The Cyc

The Boston Computer Society
TI-99/4A User Group
Software Library
Winter 1988-89

Compiled by Mike Wright
Version 19990809

Table of Contents

Introduction

This booklet is a complete listing of the software library of the Boston Computer Society's Texas Instruments TI-99/4A User Group. It contains over 100 disks. Programs have been written by members of the BCS TIUG, members of other user groups, Texas Instruments, and numerous talented individuals. All programs are on SS/SD disks, with documentation. All disks come with a printed catalog listing and label.

The catalog is divided into two parts. The first sections contains a description of every disk in the library arranged by number. This listing includes the author(s), version number, date of last update, language the program is written in, and any required cartridges. The second section is a listing of disk names and numbers by category to assist you in finding the disks you are look for. Please take the time to look through both sections as not all disks are listed in the second section.

There are several abbreviations used in this catalog. They are:

AL	assembly language
XB	Extended BASIC
EA	Editor/Assembler
TIW	TI-Writer

All disks sell for $3. A special offer is now also available. You may purchase any four disks for $10. Disks may be purchased at any BCS TIUG meeting (always the third Wednesday of the month), and at many TI Faires around the country. Disks may also be mail ordered by sending a check for the disks plus $1 for shipping and handling to:

The Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108
(617) 367-8080

Update

This update is based on the original BCS Winter 1988-89 edition.

Each disk entry now includes a full disk listing generated by the PC99 dskdir.exe utility.

Following each disk entry, are the contents of all DISPLAY/VARIABLE 80 files in order as they appear on the disk. These were created using the PC99 dskout.exe and dv802asc utilities.

Errors and typos have been fixed, and general styles have been imposed. This means that the version in this document will not always match the file on disk.

Mike Wright
August 8, 1999

BCS Background

When I joined the Boston Computer Society (BCS) it was based at One Center Plaza, Boston. The BCS catered to all computer users. A new member could elect to join two groups. A group would be, for example, the IBM PC group, the Apple group, or the TI-99/4A group.

The TI-99/4A group was run by two co-directors. They were J. Peter Hoddie and Justin Dowling. Meetings were originally held in the Boston Children Museum, then moved to the Museum of Modern Art, and finally to the Museum of Science in Waltham. It was under the BCS umbrella that the TI-99/4A group was able to hold its "fayuhs" (the word faire pronounced with a heavy Boston accent).

J. Peter Hoddie was responsible for putting together virtually all of this library. He would vet the disks and add comments to them. When Hoddie left the BCS TI-99/4A group to take up a job with Apple in Cupertino, California, this task was taken over by Justin Dowling, but the original system was disbanded. Instead, Justin began to issue Disks of the Month (DOMs).

The system of co-directors led to the unusual practice of the group having two newsletters. The BCS required that each group produce a newsletter. Hoddie would have a "meeting" newsletter at the TI-99/4A group meetings, often dashed out at the last moment. Dowling would take that newsletter and revamp it into the official newsletter. The two newsletters were never quite the same.

The BCS at one stage reported over 12,000 members. Unfortunately, it became too big for its boots by not listening to its members and, just like the Berlin Wall, was functioning in all its glory one day and simply collapsed the next.

Mike Wright
August 8, 1999

Disk 1. XB Assembly Language Routines

Version: *Author*: John Clulow
Requires: XB *Language*: AL, XB *Updated*:

Series of assembly language routines for use in XB. Source code, documentation and demo programs included for all routines. Includes sorting, screen save and restore, printer dump, and many others.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-1
Sectors total           = 360
Sectors used            = 286
Sectors available       = 72
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >003 DSRLNK        7 DIS/FIX 80   >026 006
002 >004 DSRLNK/S     12 DIS/VAR 80   >02c 011
003 >005 DUMP         11 DIS/FIX 80   >037 009 >01f 001
004 >006 DUMP/S       32 DIS/VAR 80   >040 031
005 >008 LOAD          3 PROGRAM      >022 002
006 >014 LOGO         10 DIS/FIX 80   >10e 009
007 >009 LOGO/S       26 DIS/VAR 80   >077 022 >10c 002 >117 001
008 >00d QUICK/S      15 DIS/VAR 80   >0c4 014
009 >002 READ-THIS    20 DIS/VAR 80   >024 002 >05f 017
010 >00e RECSCR        5 DIS/FIX 80   >0d2 004
011 >016 RECSCR/S     15 DIS/VAR 80   >130 014
012 >00f SAVSCR        6 DIS/FIX 80   >0d6 004 >164 001
013 >015 SAVSCR/S     15 DIS/VAR 80   >118 014
014 >01d SCREENS       3 PROGRAM      >147 002
015 >010 SCROLL        9 DIS/FIX 80   >0da 008
016 >012 SCROLL/S     32 DIS/VAR 80   >0e9 031
017 >011 SCROLLDEMO    8 PROGRAM      >0e2 007
018 >00c SORT          9 DIS/FIX 80   >0bc 008
019 >013 SORTDEMO      5 PROGRAM      >108 004
020 >019 TIME         10 PROGRAM      >13e 009
021 >01e VDP           6 DIS/FIX 80   >15f 005
022 >01c VDP/S        10 DIS/VAR 80   >14d 003 >159 006
023 >01a VDPDEMO       7 PROGRAM      >149 003 >165 003
024 >01b VENUS        10 PROGRAM      >150 009
```

Disk 1. Contents of File DSRLNK/S

```
****************************************
* Device Service Routine Link
* by Texas Instruments
*
* Modified by John Clulow, 1982
*
* Because Extended BASIC utilities don't
* include a DSRLNK, such a utility must
* be provided separately for Ext BASIC
* routines which access, for instance,
* RS232 I/O. The routine shown below was
* produced, in part, from a source pgm
* supplied by Texas Instruments and in
* part from a consideration of the
* DSRLNK utility in the Mini Memory Mod.
*
* IF the DSRLNK object code is loaded
* in the Ext BASIC program IMMEDIATELY
* after the CALL INIT statement, then
* the assembly program utilizing DSRLNK
* should use the equate DSRLNK EQU >2532
****************************************
       DEF  DSRLNK
SCLEN  EQU  >8354
SCNAME EQU  >8356
CRULST EQU  >83D0
SADDR  EQU  >83D2
GPLWS  EQU  >83E0
VSBR   EQU  >2028
FLGPTR DATA 0
SVGPRT DATA 0
SAVCRU DATA 0
SAVENT DATA 0
SAVLEN DATA 0
SAVPAB DATA 0
SAVVER DATA 0
NAMBUF DATA 0,0,0,0,0
DLNKWS DATA 0,0,0,0,0
TYPE   DATA 0,0,0,0,0,0,0,0,0,0,0
C100   DATA 100
H20    EQU  $
H2000  DATA >2000
DECMAL TEXT '.'
HAA    BYTE >AA
DSRLNK DATA DLNKWS,DLENTR
DLENTR MOV  *R14+,R5
       SZCB @H20,R15
       MOV  @SCNAME,R0
```

```
        MOV  R0,R9
        AI   R9,-8
        BLWP @VSBR
        MOVB R1,R3
        SRL  R3,8
        SETO R4
        LI   R2,NAMBUF
LNK$LP  INC  R0
        INC  R4
        C    R4,R3
        JEQ  LNK$LN
        BLWP @VSBR
        MOVB R1,*R2+
        CB   R1,@DECMAL
        JNE  LNK$LP
LNK$LN  MOV  R4,R4
        JEQ  LNKERR
        CI   R4,7
        JGT  LNKERR
        CLR  @CRULST
        MOV  R4,@SCLEN
        MOV  R4,@SAVLEN
        INC  R4
        A    R4,@SCNAME
        MOV  @SCNAME,@SAVPAB
SROM    LWPI GPLWS
        CLR  R1
        LI   R12,>0F00
NOROM   MOV  R12,R12
        JEQ  NOOFF
        SBZ  0
NOOFF   AI   R12,>0100
        CLR  @CRULST
        CI   R12,>2000
        JEQ  NODSR
        MOV  R12,@CRULST
        SBO  0
        LI   R2,>4000
        CB   *R2,@HAA
        JNE  NOROM
        A    @TYPE,R2
        JMP  SGO2
SGO     MOV  @SADDR,R2
        SBO  0
SGO2    MOV  *R2,R2
        JEQ  NOROM
        MOV  R2,@SADDR
        INCT R2
        MOV  *R2+,R9
        MOVB @SCLEN+1,R5
        JEQ  NAME2
```

```
        CB   R5,*R2+
        JNE  SGO
        SRL  R5,8
        LI   R6,NAMBUF
NAME1   CB   *R6+,*R2+
        JNE  SGO
        DEC  R5
        JNE  NAME1
NAME2   INC  R1
        MOV  R1,@SAVVER
        MOV  R9,@SAVENT
        MOV  R12,@SAVCRU
        BL   *R9
        JMP  SGO
        SBZ  0
        LWPI DLNKWS
        MOV  R9,R0
        BLWP @VSBR
        SRL  R1,13
        JNE  IOERR
        RTWP
NODSR   LWPI DLNKWS
LNKERR  CLR  R1
IOERR   SWPB R1
        MOVB R1,*R13
        SOCB @H20,R15
        RTWP
        END
```

Disk 1. Contents of File DUMP/S

```
****************************************
*
*  SINGLE-LINE DOUBLE-DENSITY GRAPHICS
*  SCREEN DUMP for TI IMPACT PRINTER
*
*  This routine is based on a screen
*  dump program by P. Swift appearing
*  in Vol. 2, No. 1 of 99er Magazine.
*
*  This DUMP routine differs from the
*  99er Magazine routine in several
*  respects. 1) It may be utilized from
*  Ext BASIC (with DSRLNK), 2) It dumps
*  a single screen line at a time which
*  allows the user to place the graphics
*  output at any location on the page by
*  preceding the CALL with a PRINT to
*  position the head, 3) It dumps in
*  double density graphics format, and
*  4) It dumps an entire graphics line
*  at once rather than only one charac-
*  ter at a time.
*
*  Modified by J. Clulow, 1982, 1984
****************************************
*
*  EQUATES FOR EXT BASIC AND MINI MEM
*
*     VSBR      >2028         >602C
*     VSBW      >2020         >6024
*     VMBR      >202C         >6030
*     VMBW      >2024         >6028
*     NUMREF    >200C         >6044
*     STRREF    >2014         >604C
*     ERR       >2034         >6050
*     ERCODE    >1E1C         >1316
*     DSRLNK    >2532         >6038
*
*  NOTES:
*
*  NUMREF, STRREF, AND ERR FOR ED/ASSM
*  ARE IN THE BSCSUP UTILITY.
*
*  FOR EXT BASIC USE THE ROUTINE DSRLNK
*  MUST BE LOADED FIRST!
*                  -----
*  To access the routine use...
*  CALL LINK("DUMP",L,"RS232.CR")
```

```
*  or other printer file specification.
*
*
       DEF  DUMP
*
VSBR   EQU  >2028    {
VSBW   EQU  >2020    {
VMBR   EQU  >202C    {
VMBW   EQU  >2024    { EXT BASIC EQUATES
NUMREF EQU  >200C    {
STRREF EQU  >2014    {
ERR    EQU  >2034    {
ERCODE EQU  >1E1C    {
DSRLNK EQU  >2532    {
WS     BSS  32
S1     BSS  2
IN     BSS  8
DO     BSS  512
SAVRTN DATA 0
MK     DATA >001F
PD     DATA >0012,>1E00,>FF00,>0000
SP     DATA 0
       BSS  24
CR     DATA >0D0A
E1     DATA >1B4C,>0002
E2     DATA >001B,>4108
D6     DATA >4019
DUMP   MOV  R11,@SAVRTN
       LWPI WS
*
* GET PRINTER SPEC
*
       LI   R0,>0017
       MOV  R0,@SP            23 BYTES MAXIMUM
       CLR  R0
       LI   R1,2              SECOND PARAMETER
       LI   R2,SP+1
       BLWP @STRREF           GET PRINTER SPEC.
*
* GET SCREEN LINE NO TO DUMP
*
       CLR  R8
       CLR  R9                STARTING SCREEN POSITION
       CLR  R0
       LI   R1,1              READ PARAMETER PASSED FROM BASIC
       BLWP @NUMREF
       CB   @>834A,@D6        IS IT OUT OF RANGE (1-24)?
       JEQ  L7
       LI   R0,ERCODE         ERROR
       BLWP @ERR
```

```
L7      C    @>834A,@D6
        JL   L6
        LI   R0,ERCODE          ERROR
        BLWP @ERR
L6      MOVB @>834B,R9
        SWPB R9                 CALCULATE STARTING POSITION
        AI   R9,-1
        SLA  R9,5
        MOVB @>9802,@S1
        SWPB @S1
        MOVB @>9802,@S1
        SWPB @S1
        DEC  @S1
*
* SET UP PAB
*
        LI   R0,>1D00
        LI   R1,PD
        MOV  @SP,R2
        AI   R2,10              NO OF PAB BYTES TO MOVE
        BLWP @VMBW              WRITE PAB TO VDP RAM
        LI   R6,>1D09
        MOV  R6,@>8356          POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK            DSRLNK TO OPEN PRINTER
        DATA 8
        JNE  GO
        MOVB @S1,@>9C02
        SWPB @S1
        MOVB @S1,@>9C02
        LI   R0,ERCODE
        SWPB R0
        BLWP @ERR
        JNE  GO                 CHECK FOR PRINTER SPEC ERROR
        BLWP @ERR               RETURN I/O ERROR
GO      LI   R0,>1D00
        LI   R1,>0300
        BLWP @VSBW              PUT WRITE OP CODE IN PAB
        LI   R0,>1D05
        LI   R1,>0400
        BLWP @VSBW              PUT LENGTH OF 4 IN PAB
        LI   R0,>1E00
        LI   R1,E2              PUT CODE FOR CARRIAGE RTN &
        LI   R2,4               8/72" VERTICAL LINE SPACING
        BLWP @VMBW              IN DATA BUFFER.
        MOV  R6,@>8356          POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK            DSRLNK-CHANGE VERT SPACING
        DATA 8
L0      MOV  R9,R0
        BLWP @VSBR              PUT BYTE OF SCREEN RAM IN R1
        SRL  R1,8               SHIFT TO LSB OF R1
        AI   R1,-128            ADJUST FOR BASIC
```

```
        SLA  R1,3             *8
        AI   R1,1024          PTRN ADDR=1024+(CHAR#-32)*8
        MOV  R1,R0
        LI   R1,IN
        LI   R2,8
        BLWP @VMBR            PUT PATTERN INTO IN
        LI   R5,128           R5 = BIT#
L3      LI   R6,128           R6 = BYTE#
        CLR  R3               R3 = OFFSET FOR IN
        CLR  R4               R4 FOR BUILDING NEXT CHAR
L2      CLR  R7
        MOVB @IN(3),R7        R7 HOLDS BYTE BEING DECODED
        SWPB R7               PUT BYTE IN LSB OF R7
        C    R7,R5            IS BIT ON?
        JLT  L1               NO
        A    R6,R4            YES, TURN OUTPUT BIT ON
        S    R5,R7            TURN OFF INPUT BIT
        SWPB R7               PUT BYTE IN MSB OF R7
        MOVB R7,@IN(3)        REWRITE TO IN
L1      INC  R3               POINT TO NEXT BYTE
        SRA  R6,1             /2
        JGT  L2               DO NEXT BYTE IF MORE
        SWPB R4               PUT OUTPUT BYTE IN MSB OF R4
        MOVB R4,@DO(8)
        INC  R8
        MOVB R4,@DO(8)        STORE AT DO
        INC  R8               POINT TO NEXT BYTE OF DO
        SRA  R5,1             /2
        JGT  L3               CONSTRUCT NEXT OUTPUT BYTE
        INC  R9               NEXT SCREEN POS
        CZC  @MK,R9           EOL?
        JNE  L0               NO, NEXT POSITION
        LI   R3,4             COUNTER
        LI   R0,>1D05         ONLY ESC K WRITE
        LI   R1,>0400
        BLWP @VSBW            PUT LENGTH OF 4 IN PAB
        LI   R0,>1E00         VDP BUFFER
        LI   R1,E1
        LI   R2,4
        BLWP @VMBW            PUT ESC K SEQ IN DATA BUFF
        LI   R6,>1D09
        MOV  R6,@>8356        POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK          DSR TO WRITE ESC K SEQUENCE
        DATA 8
        LI   R4,DO            START OF CPU GRAPHICS BUFFER
L5      LI   R2,128           QUARTER OF GRAPHICS STRING
        MOV  R4,R1
        LI   R0,>1E00         VDP ADDR
        BLWP @VMBW            PUT DO IN DATA BUFFER
        LI   R0,>1D05
```

```
        LI   R1,>8000          128 BYTES
        BLWP @VSBW
        MOV  R6,@>8356         POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK           DSR TO OUTPUT 8 CHARS
        DATA 8
        AI   R4,128            POINT TO START OF NEXT QUARTER
        DEC  R3
        JNE  L5                DO IT AGAIN FOR LAST HALF
        LI   R0,>1D05          OUTPUT CR/LF
        LI   R1,>0200
        BLWP @VSBW             PUT LENGTH OF 2 IN PAB
        LI   R0,>1E00
        LI   R1,CR
        LI   R2,2
        BLWP @VMBW
        MOV  R6,@>8356
        BLWP @DSRLNK           DSRLNK TO OUTPUT CR/LF
        DATA 8
L4      LI   R0,>1D00
        LI   R1,>0100
        BLWP @VSBW             PUT CLOSE IN PAB
        MOV  R6,@>8356
        BLWP @DSRLNK
        DATA 8
        LI   R0,4              DELAY
DEL1    LI   R1,20000
DEL2    NOP
        DEC  R1
        JNE  DEL2
        DEC  R0
        JNE  DEL1
        MOVB @S1,@>9C02
        SWPB @S1
        MOVB @S1,@>9C02
        SB   @>837C,@>837C
        LWPI >83E0
        MOV  @SAVRTN,R11
        RT
        END
```

Disk 1. Contents of File LOGO/S

```
       DEF  LOGO
*
* PRINTER DUMP FROM EXT BASIC OF
* P_FILENAME SAVED FROM LOGO I.
*
* J. Clulow, 1981
*
*      REF  DSRLNK,VMBR,VMBW,VSBW,VSBR
       AORG >A000
DSRLNK EQU  >2532
VMBR   EQU  >202C
VMBW   EQU  >2024
VSBW   EQU  >2020
VSBR   EQU  >2028
STRREF EQU  >2014
ERR    EQU  >2034
ERCODE EQU  >1C00
WS     BSS  32
S1     BSS  2
BUFFER BSS  >2000
SAVRTN DATA 0
NULL   DATA 0
DISK   DATA >0500,START,>0000,>2000
DK     DATA >000F
       BSS  15
PRINT  DATA >0010,START,>5050,>0000
PT     DATA >001A
       BSS  26
SKIPOP DATA >0D1B,>4E06
WRITE  BYTE >03
CLOSE  BYTE >01
CR     BYTE >0D
LF     BYTE >0A
START  EQU  >1000
PAB    EQU  >0F80
PNTR   EQU  >8356
LOGO   MOV  R11,@SAVRTN
       LWPI WS
*
* GET DISK FILE SPEC
*
       LI   R0,>000F
       MOV  R0,@DK
       CLR  R0
       LI   R1,1
       LI   R2,DK+1
       BLWP @STRREF
*
```

```
* GET PRINTER SPEC
*
       LI   R0,>0017
       MOV  R0,@PT
       CLR  R0
       LI   R1,2
       LI   R2,PT+1
       BLWP @STRREF
*
* INITIALIZE VDP BUFFER WITH NULLS
*
       LI   R6,START          INITIALIZE COUNTER
       LI   R7,START          INITIALIZE VDP BUFFER ADDRESS
LOOP1  MOV  R7,R0             VDP ADDRESS
       LI   R1,NULL           CPU ADDRESS
       LI   R2,2              2 BYTES TO LOAD
       BLWP @VMBW             LOAD NULL TO VDP BUFFER
       INCT R7                ADD 2 TO VDP ADDRESS
       DEC  R6                DECREMENT COUNTER
       JNE  LOOP1             IF NOT DONE GO BACK TOO LOOP1
*
* LOAD LOGO PROCEDURE FILE TO VDP RAM
*
       MOVB @>9802,@S1
       SWPB @S1
       MOVB @>9802,@S1
       SWPB @S1
       DEC  @S1
       LI   R0,PAB            LOAD ADDRESS OF PAB
       LI   R1,DISK           LOAD CPU ADDRESS OF DATA
       MOV  @DK,R2
       AI   R2,10
       BLWP @VMBW             BUILD PAB FOR FILE LOAD
       LI   R6,PAB+9          ADDRESS OF NAME LENGTH
       MOV  R6,@PNTR          MOVE TO POINTER ADDRESS
       BLWP @DSRLNK           LOAD FILE FROM DISK
       DATA 8
       JNE  GO1
       MOV  @S1,@>9C02
       SWPB @S1
       MOV  @S1,@>9C02
       LI   R0,ERCODE
       BLWP @ERR
*
* COPY VDP BUFFER TO CPU BUFFER
*
GO1    LI   R0,START          VDP BUFFER ADDRESS
       LI   R1,BUFFER         CPU BUFFER ADDRESS
       LI   R2,>2000          8K TO MOVE
       BLWP @VMBR             READ DATA
*
```

```
* COPY BACK INTO VDP ADDING LF CONTROL CHRS
*
       LI   R6,>1004          POINTER TO VDP BUFFER ADDRESS
       LI   R7,4              OFFSET FOR START OF DATA
CHECK  CB   @NULL,@BUFFER(R7) END OF DATA?
       JEQ  NEXT1             IF SO GO ON
       CB   @CR,@BUFFER(R7)   IS THE CHARACTER A CR?
       JEQ  ADD               IF SO ADD A LF
       MOV  R6,R0             CURRENT VDP ADDRESS TO WRITE TO
       MOVB @BUFFER(R7),R1    BYTE TO WRITE
       BLWP @VSBW             WRITE THE CHARACTER TO VDP ADDRESS
       INC  R6                NEXT VDP ADDRESS
       INC  R7                NEXT CHARACTER FROM CPU BUFFER
       JMP  CHECK             CHECK NEXT ONE
ADD    MOV  R6,R0             CURRENT VDP ADDRESS TO WRITE TO
       LI   R1,CR             CPU ADDRESS OF LF AND CR BYTES
       LI   R2,2              2 BYTES TO WRITE
       BLWP @VMBW             WRITE CR LF
       INCT R6                NEXT VDP ADDRESS
       INC  R7                NEXT CHARACTER FROM CPU BUFFER
       JMP  CHECK             CHECK NEXT ONE
*
* ADD SKIP OVER PERFORATION
*
NEXT1  LI   R0,>1000          VDP ADDRESS
       LI   R1,SKIPOP         CPU ADDRESS
       LI   R2,4              4 BYTES TO WRITE
       BLWP @VMBW             WRITE DATA 13 27 78 6
*
* OPEN RS232 FILE
*
       LI   R0,PAB            LOAD ADDRESS OF PAB
       LI   R1,PRINT          LOAD CPU ADDRESS OF DATA
       LI   R2,36             36 BYTES TO WRITE
       BLWP @VMBW             BUILD PAB FOR OUTPUT FILE
       LI   R6,PAB+9          ADDRESS OF NAME LENGTH
       MOV  R6,@PNTR          MOVE IT TO POINTER ADDRESS
       BLWP @DSRLNK           OPEN RS232 FILE
       DATA 8
       JNE  GO2
       MOVB @S1,@>9C02
       SWPB @S1
       MOVB @S1,@>9C02
       LI   R0,ERCODE
       BLWP @ERR
*
* INITIALIZE POINTER - CURRENT VDP RAM
*
GO2    LI   R7,>1000          START OF VDP RAM BUFFER
*
```

```
* CHANGE PAB TO WRITE OPERATION CODE
*
       LI   R0,PAB            ADDRESS OF PAB IN VDP RAM
       MOVB @WRITE,R1         LOAD OPERATION CODE FOR WRITE
       BLWP @VSBW             WRITE OPERATION CODE IN PAB
*
* BEGIN LOOP - SUCCESSIVE 80 BYTE READ/WRITES
*
READ   MOV  R7,R0             START AT CURRENT VDP ADDRESS
       BLWP @VSBR             READ DATA INTO CPU RAM BUFFER
       CB   R1,@NULL          IS FIRST BYTE NULL?
       JNE  NEXT              IF NOT GO ON
       LI   R0,PAB            PREPARE TO CLOSE RS232
       MOVB @CLOSE,R1         LOAD CLOSE OPERATION CODE
       BLWP @VSBW             MODIFY PAB FOR CLOSE
       LI   R6,PAB+9          LOAD ADDRESS OF NAME LENGTH
       MOV  R6,@PNTR          PLACE IT AT POINTER ADDRESS
       BLWP @DSRLNK           CLOSE FILE
       DATA 8
       MOVB @S1,@>9C02
       SWPB @S1
       MOVB @S1,@>9C02
       SB   @>837C,@>837C
       LWPI >83E0             LOAD GPLWS REGISTERS
       MOV  @SAVRTN,R11
       RT
NEXT   LI   R0,PAB+2          ADDRESS IN PAB OF VDP BUFFER
       LI   R1,WS+14          CURRENT VDP ADDRESS
       LI   R2,2              2 BYTES TO WRITE
       BLWP @VMBW             WRITE NEW STARTING ADDRESS TO PAB
       AI   R7,>0050          INCREMENT STARTING ADDRESS BY 80
       LI   R6,PAB+9          LOAD ADDRESS OF NAME LENGTH
       MOV  R6,@PNTR          MOVE IT TO POINTER ADDRESS
       BLWP @DSRLNK           WRITE 80 BYTES TO RS232
       DATA 8
       B    @READ             GO BACK AND READ AGAIN
       END
```

Disk 1. Contents of File QUICK/S

```
************************************
* THIS ROUTINE PERFORMS A QUICKSORT   *
* ON STRINGS. ROUTINE IS CODED FOR    *
* EXTENDED BASIC AND MEMORY EXPANSION.*
*                                     *
*   CALL LINK("SORT",S$(),N)          *
*                                     *
*  WHERE S$ IS ARRAY TO BE SORTED AND *
*  N IS LARGEST ARRAY ELEMENT.        *
*                                     *
************************************
       DEF SORT
NUMREF EQU >200C
STRREF EQU >2014
STRASG EQU >2010
ERR    EQU >2034
STATUS EQU >837C
ST     BSS 2000          STACK
B1     BSS 256           Z$
B2     BSS 256           S$(L)
B3     BSS 256           S$(U)
WS     BSS 32            USER WORKSPACE
SAVE   DATA 0            RETURN ADDRESS
ONE    DATA >0001
FF     BYTE >FF
SORT   MOV R11,@SAVE
       LWPI WS
*
*      GET N
*
       CLR R0
       LI R1,2
       BLWP @NUMREF
       LI R0,>4041
       CB @>834A,R0
       JEQ DECOD2
       SWPB R0
       CB @>834A,R0
       JEQ DECOD1
       LI R0,>1300
       BLWP @ERR         RETURN ERROR
DECOD1 LI R2,100
       CLR R3
       CLR R4
       MOVB @>834B,R3
       SWPB R3
       MPY R2,R3
       CLR R3
```

```
        MOVB @>834C,R3
        SWPB R3
        A R3,R4
        JMP  SRT
DECOD2  CLR R4
        MOVB @>834B,R4
        SWPB R4
*
*       START PROGRAM
*
SRT     CLR R10
        CLR R9
        MOV @ONE,@ST+2
        MOV R4,@ST+4
        INC R10
L940    CI R10,0
        JNE A
        B @L1230
A       DEC R10
        MOV R10,R9
        SLA R9,1
        MOV R9,R1
        SLA R1,1
        MOV @ST+2(R1),R12
        MOV @ST+4(R1),R13
        MOV R12,R0
        LI R1,1
        LI R2,B1
        MOVB @FF,@B1
        BLWP @STRREF
        MOV R12,R15
        MOV R13,R14
        INC R14
L1020   DEC R14
        C R14,R15
        JEQ L1110
        MOV R14,R0
        LI R1,1
        LI R2,B2
        MOVB @FF,@B2
        BLWP @STRREF
        LI R1,B1
        LI R2,B2
        BL @COMPAR
        CI R0,1
        JEQ X1
        JMP L1020
X1      MOV R15,R0
        LI R1,1
        LI R2,B2
        BLWP @STRASG
```

```
L1060  INC R15
       C R14,R15
       JEQ L1110
       MOV R15,R0
       LI R1,1
       LI R2,B3
       MOVB @FF,@B3
       BLWP @STRREF
       LI R1,B1
       LI R2,B3
       BL @COMPAR
       CI R0,2
       JEQ X2
       JMP L1060
X2     MOV R14,R0
       LI R1,1
       LI R2,B3
       BLWP @STRASG
       JMP L1020
L1110  MOV R15,R0
       LI R1,1
       LI R2,B1
       BLWP @STRASG
       MOV R13,R1
       S R15,R1
       CI R1,2
       JLT L1170
       MOV R10,R9
       SLA R9,1
       MOV R15,R1
       INC R1
       MOV R9,R2
       SLA R2,1
       MOV R1,@ST+2(R2)
       MOV R13,@ST+4(R2)
       INC R10
L1170  MOV R14,R1
       S R12,R1
       CI R1,2
       JLT L1220
       MOV R10,R9
       SLA R9,1
       MOV R9,R2
       SLA R2,1
       MOV R12,@ST+2(R2)
       MOV R14,R1
       DEC R1
       MOV R1,@ST+4(R2)
       INC R10
L1220  B @L940
```

```
L1230  CLR R0
       MOV R0,@STATUS
       LWPI >83E0        GPLWS
       MOV @SAVE,R11
       RT                RETURN TO BASIC
COMPAR CLR R3
       MOVB *R1,R3
       MOV R2,R0
       CB *R2,*R1
       JHE N2
       MOVB *R2,R3
N2     SWPB R3
LOOP   INC R1
       INC R2
       CB *R2,*R1
       JL LESS
       JH GRTR
       DEC R3
       JGT LOOP
       MOV R0,R2
       CB *R2,@B1
       JL LESS
       JH GRTR
       LI R0,3
       B *R11
LESS   LI R0,1
       B *R11
GRTR   LI R0,2
       B *R11
       END
```

Disk 1. Contents of File READ-THIS

Assembly Language routines for Extended BASIC from the New Horizons User Group

LOAD (J. Clulow)

Provides menu access to demos for all routines on the disk.

DSRLNK (Texas Instruments)

This routine is a stand alone Device Service Routine Link for the Disk, RS232, etc. Modified from a source program supplied by TI, this version duplicates the Mini-Memory and Editor/Assembler versions. You may use it in you assembly programs using the EQUate >2532. For this EQUate to be valid, DSRLNK must be the first CALL LOAD item after CALL INIT is the Ext. BASIC program. DSRLNK is used in many of the routines described below and IT MUST ALWAYS BE LOADED FIRST!

DUMP (J. Clulow)

Load as follows:

```
CALL LOAD("DSK1.DSRLNK","DSK1.DUMP")
```

This routine produces a high res printer dump (TI Impact, Epson, Gemini, etc.) of a single graphics line from the screen. Only one line is dumped at a time so that a separate file may be opened to the printer and the print head positioned and/or additional data printer, if desired, prior to printing the graphics line. This technique could be used to label coordinates of a graph, for example. The line desired is passed as the first parameter in the link:

```
CALL LINK("DUMP",L,"PIO.CR")
```

If you use an RS232 spec, simply substitute it in the LINK statement for PIO. Make sure your spec ends with .CR!

QUICKSORT (David R. Romer, J. Clulow)

The SORT routine does a full ASCII sort on any string array. OPTION BASE 1 should be in effect. To load the routine use: CALLS LOAD("DSK1.SORT").To link to it, with your data in an array — e.g., A$() — use CALL LINK("SORT",A$(),n) where N is the number of items to be sorted (the highest array element used).

SCREEN SAVE PACKAGE (Ken Hopkins, TIUP)

These utilities allow the entire graphics portion of VDP RAM to be saved on disk and re-loaded in only a couple of seconds. The effect is to rapidly recreate motion, etc. By saving screens on disk, programs can generate graphics much faster than via ordinary BASIC instructions while at the same time drastically reducing associated program memory requirements. Once loaded from disk. all graphics may be treated within the program exactly as if they had been generated by the program.

To load the routines you must load DSRLNK first! Then either or both of the screen utilities.

```
CALLS LOAD("DSK1.DSRLNK","DSK1.SAVSCR","DSK1.RECSCR")
```

To save a screen use CALL LINK("SAVSCR","DSK1.NAME",SC) where DSK1.NAME is the output file name and SC is the screen color to be used with the graphics screen.

To load a screen back in use: "CALL LINK("RECSCR","DSK1.NAME").A screen takes about 3 seconds to load.

VPOKE/VPEEK (J. Clulow)

VPOKE and VPEEK allow access to VDP RAM from within an Extended BASIC program thus allowing direct modification of sprite, color, character definition, screen and other tables. To load the utilities use: CALL LOAD("DSK1.VDP").To execute them use:

```
CALL LINK("VPEEK",MEM,N,A$)
CALL LINK("VPOKE",MEM,A$)
```

There routines differ from those of the E/A or Mini Mem in that data are transferred via a string of bytes. The byte value 65 would be represented as CHR$(65) and 235 as CHR$(235), for example. Strings may be up to 255 bytes long allowing for rapid transfer for a large number of bytes.

In the example above, MEM is the starting VDP RAM location to be used in writing or reading data. N is the number of bytes to be read, and A$ is the return string for VPEEK and the data string for VPOKE.

SCROLL (J. Clulow)

The SCROLL routine allows you to scroll any string horizontally across any screen line. The utility is loaded with: CALL LOAD("DSK1.SCROLL"). The link statement is: CALL LINK("SCROLL",X$,L).Where X$ may be any valid string expression (other than an array) and L any numeric expression which evaluates to between 1 and 24.

LOGO I PRINT (J. Clulow)

This routine will provide a printout of a LOGO I procedure file. To LOAD the routine use: CALL LOAD("DSK1.DSRLNK","DSK1.LOGO")To print a LOGO procedure file, use: CALL LINK("LOGO","DSK1.P_FILE","PIO")The file name and printer specs may be changed as required. However, it should be noted that all LOGO procedure files are prefixed with P_.

Disk 1. Contents of File RECSCR/S

```
*RECALL SCREEN ROUTINE
*BY KEN HOPKINS, TIUP WESTERN AUSTRALIA
* modified for ext BASIC 9/4/83 J. Clulow
*
*
       TITL  'RECALL SAVED SCREEN ROUTINE JUNE 1983'
       DEF  RECSCR
*
*      REF  ERR,STRREF,VMBR,VSBR,VSBW,VMBW,DSRLNK,VWTR
ERR    EQU >2034
STRREF EQU >2014
DSRLNK EQU >2532
VMBR   EQU >202C
VSBR   EQU >2028
VSBW   EQU >2020
VWTR   EQU >2030
VMBW   EQU >2024
WS     EQU >20BA
FNLENB BYTE >0F
PAB    EQU  >0F80
PABERR EQU  >831C
PDATA  DATA >0500,>1000,>0000,>0823
       BYTE >00
FILNAM BSS  >0010             ;FILENAME CAN BE 15 BYTES + LENGTH BYTE
       EVEN
SAVRTN DATA 0
PNTR   EQU  >8356
TABLES BSS  >0820
NSPR   BYTE >00
SCRCOL BYTE >07
MAGFAC BYTE >00
RECSCR MOV  R11,@SAVRTN
       LWPI WS
       MOVB @FNLENB,@FILNAM   ;LENGTH OF BUFFER IN LENGTH BYTE
       LI   R0,>0000          ;SCALAR NOT STRING ARRAY
       LI   R1,>0001          ;FIRST PARAMETER IN LINK CALL
       LI   R2,FILNAM         ;ADDRESS FOR STRING TO GO TO
       BLWP @STRREF           ;GET STRING PARAMETER
       LI   R0,PAB            ;VDP PERIPHERAL ACCESS BLOCK ADDRESS
       LI   R1,PDATA          ;CPU BUFFER TO WRITE
       LI   R2,>0A00          ;10 IN HIGH BYTE
       AB   @FILNAM,R2        ;ADD FILENAME LENGTH TO HIGH BYTE
       SWPB R2                ;MOVE BYTES TO WRITE TO LOW BYTE
       BLWP @VMBW             ;WRITE PAB
       LI   R6,PAB+9          ;SET POINTER TO NAME LENGTH
       MOV  R6,@PNTR          ;STORE IN >8356 & >8357
       BLWP @DSRLNK           ;COPY DATA TO VDP BUFFER
       DATA 8                 ;THIS IS A DSRLNK CALL
```

```
       LI   R0,PAB+1          ;THIS BYTE OF PAB GETS THE ERROR CODE
       CLR  R1                ;GET READY TO RECEIVE CODE
       BLWP @VSBR             ;READ THE CODE FROM PAB
       ANDI R1,>C000          ;DISCARD ALL BUT LEFT TWO BITS
       CI   R1,>0000          ;WAS THERE AN ERROR?
       JEQ  CONTIN            ;NO KEEP GOING
       SRL  R1,6              ;MOVE BITS TO WHERE THEY ARE A NUMBER
       MOV  R1,R0             ;ERR WANTS IT IN R0
       LI   R4,PAB            ;ADDRESS OF PAB IN R0
       MOV  R4,@PABERR        ;PUT IT WHERE ERR ROUTINE WANTS IT
       BLWP @ERR              ;TELL BASIC ABOUT NAUGHTINESSS
CONTIN LI   R0,>1000          ;PREPARE TO COPY FROM VDP TO TABLES
       LI   R1,TABLES         ;CPU BUFFER ADDRESS
       LI   R2,>0823          ;2664 BYTES TO WRITE
       BLWP @VMBR             ;COPY THE BUFFER
       MOVB @NSPR,@>837A      ;N SPRITES IN MOTION
       CLR  R0
       MOVB @SCRCOL,R0
       SWPB R0
       AI   R0,>0700
       BLWP @VWTR             ;RESTORE SCREEN COLOR
       CLR  R0
       MOVB @MAGFAC,R0
       MOVB @MAGFAC,@>83D4
       SWPB R0
       AI   R0,>0100
       BLWP @VWTR
       LI   R0,>0000          ;NOW COPY TO PATTERN TABLE
       LI   R1,TABLES         ;ADDRESS OF CPU BUFFER
       LI   R2,>0820          ;1536 BYTES TO COPY
       BLWP @VMBW             ;COPY TO TABLE
       LWPI >83E0
       MOV  @SAVRTN,R11
       B    *R11              ;RETURN TO CALLING PROGRAM
       END
```

Disk 1. Contents of File SAVSCR/S

```
*SAVE SCREEN ROUTINE
*ADAPTED FROM "THE MAGIC CRAYON"
*BY JOHN CLULOW
* 99'ER MAGAZINE VOL. 1 NO 6.
*BY ken hopkins
* modified for Ext BASIC 9/4/83
*
       TITL  'SAVE SCREEN ROUTINE JUNE 1983'
*
       DEF  SAVSCR
*      REF  ERR,STRREF,VMBR,VSBR,VMBW,DSRLNK
ERR    EQU  >2034
STRREF EQU  >2014
NUMREF EQU  >200C
VMBR   EQU  >202C
VSBR   EQU  >2028
VMBW   EQU  >2024
DSRLNK EQU  >2532
FAC    EQU  >834A
WS     BSS  32
FNLENB BYTE >0F
PAB    EQU  >0F80
PABERR EQU  >831C
PDATA  DATA >0600,>1000,>0000,>0823
       BYTE >00
FILNAM BSS  >0010             ;FILENAME CAN BE 15 BYTES +LENGTH BYTE
       EVEN
SAVRTN DATA 0
PNTR   EQU  >8356
TABLES BSS  >0820
NSPR   BYTE >00
SCRCOL BYTE >07
MAGFAC BYTE >00
SAVSCR MOV  R11,@SAVRTN
       LWPI WS
       MOVB @FNLENB,@FILNAM   ;LENGTH OF BUFFER IN LENGTH BYTE
       LI   R0,>0000          ;SCALAR STRING NOT ARRAY
       LI   R1,>0001          ;FIRST PARAMETER IN LINK CALL
       LI   R2,FILNAM         ;ADDRESS FOR STRING TO GO TO
       BLWP @STRREF           ;GET STRING PARAMETER  FOR FILENAME
       LI   R0,>0000          ;BEGINNING OF SCREEN TABLE
       LI   R1,TABLES         ;CPU BUFFER TO COPY TO
       LI   R2,>0820          ;2080 BYTES TO WRITE
       BLWP @VMBR             ;COPY SCREEN TABLES TO CPU BUFFER
       MOVB @>837A,@NSPR      ;N ACTIVE SPRITES
       CLR  R0                ;OBTAIN SCREEN COLOR FROM LINK
       LI   R1,2
       BLWP @NUMREF
```

```
        MOV  @FAC,R1
        CI   R1,>4001
        JL   SK1
        CI   R1,>4010
        JH   SK1
        MOVB @FAC+1,R0
        SWPB R0
        DEC  R0
        SWPB R0
        MOVB R0,@SCRCOL
SK1     MOVB @>83D4,@MAGFAC     ;GET MAG FACTOR
        LI   R0,>1000           ;PREPARE TO MOVE TABLES TO VDP BUFFER
        LI   R1,TABLES          ;CPU BUFFER ADDRESS
        LI   R2,>0823           ;2064  BYTES TO MOVE
        BLWP @VMBW              ;WRITE DATA
        LI   R0,PAB             ;VDP PERIPHERAL ACCESS BLOCK ADDRESS
        LI   R1,PDATA           ;CPU BUFFER TO BE WRITTEN TO VDP
        LI   R2,>000A           ;  10 IN HIGH BYTE
        CLR  R4
        MOVB @FILNAM,R4
        SWPB R4
        A    R4,R2              ;10 +LENGTH OF FILENAME IN HIGH BYTE
        BLWP @VMBW              ;WRITE THE PAB
        LI   R6,PAB+9           ;SET POINTER TO NAME LENGTH
        MOV  R6,@PNTR           ;STORE IN >8356 & >8357
        BLWP @DSRLNK            ;EXECUTE THE SAVE
        DATA 8                  ;THIS IS A DSRLNK CALL
        LI   R0,PAB+1           ;THIS BYTE OF PAB GETS ERROR CODES
        CLR  R1                 ;GET READY FOR CODE
        BLWP @VSBR              ;READ THE BYTE FROM PAB
        ANDI R1,>C000           ;DISCARD ALL BUT LEFT 2 BITS
        CI   R1,>0000           ;WAS THERE AN ERROR?
        JEQ  BASRET             ;NO, ALL IS O.K.
        SRL  R1,6               ;MOVE TO WHERE THEY MEAN NUMBER FOR ERR CODE
        MOV  R1,R0              ;ERR WANTS IT IN R0
        LI   R4,PAB             ;ADDRESS OF PAB IN R0
        MOV  R4,@PABERR         ;INTO ADDRESS WHERE ERR WANTS IT
        BLWP @ERR               ;CALL ERROR ROUTINE
BASRET  LWPI >83E0
        MOV  @SAVRTN,R11
        B    *R11               ;RETURN TO CALLING PROGRAM
        END
```

Disk 1. Contents of File SCROLL/S

```
*************************************************************
*                                                           *
*               LINE SCROLLING UTILITY                      *
*
*               By John Clulow, 1983
*
*************************************************************
       DEF SCROLL
*
* EXT BASIC EQUATES
*
SW     EQU >2020 VSBW
MW     EQU >2024 VMBW
MR     EQU >202C VMBR
SR     EQU >2028 VSBR
NR     EQU >200C NUMREF
ER     EQU >2034 ERR
BV     EQU >1E00 BAD VALUE CODE
FC     EQU >834A FLOATING POINT ACCUMULATOR
SP     EQU >837A # SPRITES IN MOTION
ST     EQU >2014 STRING REFERENCE
*
* --- ASSEMBLER DIRECTIVES ---
*
* B1 - BUFFER FOR CURRENT DISPLAY LINE
* B2 - BUFFER FOR CHAR PATTERNS 34 CHARS
* B3 - BUFFER FOR STRING FROM BASIC
*
B1     BSS 32
B2     BSS 272
B3     BSS 256
WS     BSS 32  WORKSPACE
*
*
H1     DATA >0100 1 FOR SOCB BIT ADDITION.
HO     DATA >6000 ASCII OFFSET FROM BASIC. LSB IS FLAG.
CK     DATA >4018 BYTES 64 AND 24 FOR ERROR CHECK
SV     DATA >0000 SAVE BASIC RTN ADDRESS
SN     BYTE >00   NUMBER OF MOVING SPRITES AT CALL
*
* MAKE COLORS FOR SETS 15,16 = SET 14
*
SCROLL MOV  R11,@SV
       LWPI WS
       LI   R0,>081D
       BLWP @SR
       LI   R0,>081E
       BLWP @SW
```

```
       LI   R0,>081F
       BLWP @SW
       MOVB @SP,@SN      SAVE @ ACTIVE SPRITES
       MOVB @H1+1,@SP    NO SPRITES IN MOTION!
*
* EVALUATE SCREEN LINE NUMBER
*
       LI   R0,0         NOT AN ARRAY
       LI   R1,2         SECOND LINK PARAMETER
       BLWP @NR          GET RADIX 100 NUMBER
       CB   @FC,@CK      FIRST BYTE MUST BE 64
       JNE  BD
       CB   @FC+1,@CK+1  SECOND BYTE MUST BE <25
       JH   BD
       CLR  R9           INITIALIZE LINE # REGISTER
       MOVB @FC+1,R9
       SWPB R9           MAKE LSB TO CALC SCREEN OFFSET
       DEC  R9           SUBTRACT 1
       SLA  R9,5         MULTIPLY BY 32
       JMP  L0
BD     LI   R0,BV        ERROR CODE BAD VALUE
       BLWP @ER          RETURN CODE TO BASIC
*
* SAVE CURRENT DISPLAY LINE
*
L0     MOV  R9,R0        BEGINNING OF SCREEN LINE, VDP
       LI   R1,B1        START OF CPU SAVE BUFFER
       LI   R2,32        32 BYTES TO TRANSFER
       BLWP @MR          READ INTO BUFFER
*
* PUT CORRESPONDING PATTERNS IN CHARS 128-159
*     START WITH POSITION 32 (AT EXTREME RIGHT)
*
       LI   R3,31        COUNTER
L1     CLR  R4           INITIALIZE
       MOVB @B1(R3),R4   GET CHAR CODE
       SWPB R4           MAKE IT LSB
       SLA  R4,3         MPY BY 8 TO GET VDP PATTERN OFFSET
       MOV  R3,R5        COPY COUNTER
       SLA  R5,3         MPY BY 8 FOR B2 OFFSET
       MOV  R4,R0        VDP STARTING ADDRESS FOR PATTERN
       MOV  R5,R1        B2 OFFSET
       AI   R1,B2+8      ADD B2+1 TO STARTING ADDRESS
       LI   R2,8         8 BYTES TO GET FOR CHAR PATTERN
       BLWP @MR          READ INTO BUFFER
       DEC  R3           SUBTRACT 1 FROM R3
       JLT  L2           IF DONE GO ON
       JMP  L1           IF NOT, DO NEXT CHAR
L2     LI   R0,1792      VDP ST OF PATTERN FOR CH 128 (224)
       LI   R1,B2+8      CPU START OF PATTERNS ROW 23
```

```
       LI   R2,256       256 BYTES FOR 32 PATTERNS
       BLWP @MW          WRITE PATTERNS
*
* NOW DISPLAY CHARS 128 TO 159 AT ROW 23 ON SCREEN
*
       MOV  R9,R4        FIRST CHAR ON LINE 23, VDP RAM
       LI   R5,224       ASCII CODE FOR CHAR 128 (COUNTER)
       SWPB R5           MAKE MSB
L3     MOVB R5,R1        BYTE TO WRITE
       MOV  R4,R0        VDP LOCATION
       BLWP @SW          DISPLAY CHARACTER
       INC  R4           NEXT VDP MEMORY LOCATION
       AI   R5,>0100     NEXT CHAR
       JNE  L3           IF NOT DONE, DISPLAY NEXT CHAR
*
* PUT STRING FROM BASIC INTO B3
*
       LI   R0,0         NOT AN ARRAY
       LI   R1,1         FIRST PARAMETER IN CALL LINK
       LI   R2,B3        STARTING ADDRESS OF CPU BUFFER
       LI   R3,>FF00     MAX STRING LENGTH
       MOVB R3,@B3       STORE AS FIRST BYTE OF BUFFER
       BLWP @ST          PUT STRING IN BUFFER
*
* PUT STRING LENGTH IN R5 AND SET CHAR POINTER
*
       CLR  R5           INITIALIZE
       MOVB @B3,R5       GET LENGTH BYTE
       SWPB R5           MAKE LSB
       LI   R6,B3+1      POINTER TO FIRST (NEXT) STRING CHAR
*
* ADD OFFSET TO EACH ASCII CODE IN B3
*
       MOV  R5,R2        R2 IS BYTE COUNTER
       MOV  R6,R1        R1 IS BYTE POINTER
M1     AB   @HO,*R1+     ADD 96 TO ASCII CODE
       DEC  R2           DEC BYTE POINTER
       JNE  M1           NEXT CHAR
*
* SET FLAG TO ZERO FOR FIRST TIME THROUGH
*
       MOVB @HO+1,@H1+1
*
* ------- BEGIN LOOP TO SCROLL PIXELS --------
*
* R3 = COUNTER FOR # BYTES TO BE SHIFTED
* R4 = POINTER TO PATTERN BYTE IN B2 TO BE SHIFTED
* R5 = COUNTER FOR STRING LENGTH
* R6 = POINTER TO NEXT STRING CHARACTER
* R7 = CYCLE COUNTER (8 PER BYTE)
* R8 = USED FOR SHIFT OPERATION (ON BYTE)
```

```
*
L4      CLR  R0           INITIALIZE
        MOVB *R6+,R0      PUT NEXT CHAR IN R0
        SWPB R0           MAKE LSB
        SLA  R0,3         MPY BY 8 FOR VDP OFFSET
        LI   R1,B2+264    CPU BUFFER
        LI   R2,8         8 BYTES FOR PATTERN
        BLWP @MR          READ PATTERN
        LI   R7,8         CYCLE COUNTER
L5      LI   R4,B2+8      PATTERN BYTE POINTER
        LI   R3,264       COUNTER FOR BYTES TO BE SHIFTED
        CLR  R8           INITIALIZE SHIFT REGISTER
L6      MOVB *R4,R8       PUT PATTERN BYTE IN MSB R8
        SLA  R8,1         SHIFT ONE BIT RIGHT
        JOC  L7           IF A BIT WASN'T SHIFTED OUT, GO ON
        JMP  L8           IF NOT, GO ON
L7      SOCB @H1,@-8(R4)  IF BIT SHIFTED, SET IN PRECEDING
L8      MOVB R8,*R4       PUT BYTE BACK
        INC  R4           POINT TO NEXT BYTE
        DEC  R3           DEC SHIFT BYTE COUNTER
        JNE  L6           NEXT BYTE
        LI   R0,1792      VDP PATTERN BUFFER
        LI   R1,B2+8      CPU PATTERN BUFFER
        LI   R2,256       256 BYTES TO WRITE
        BLWP @MW          WRITE MODIFIED PATTERN TO VDP
        DEC  R7           DEC SHIFTS PER BYTE
        JNE  L5           IF NOT DONE, ANOTHER SHIFT
        DEC  R5           DEC CHAR COUNTER
        JNE  L4           IF NOT DONE, NEXT CHAR
*
* IS FLAG BYTE 1? IF NOT, DO 32 SPACES
*
        CB   @HO+1,@H1+1  IS FLAG 0?
        JNE  M2           IF NOT, FINISH
        LI   R0,32        COUNTER
        LI   R1,B3+1      B3 POINTER
        LI   R2,>8000     128 (SPACE + 96)
L9      MOVB R2,*R1+      STORE TWO SPACES
        DEC  R0           DEC COUNTER
        JNE  L9           IF NOT DONE, DO TWO MORE
        LI   R5,32        SET NEW STRING LENGTH IN COUNTER
        LI   R6,B3+1      POINT TO START OF NEW STRING
        MOVB @H1,@H1+1    SET FLAG BYTE
        B    @L4          SCROLL LINE OF SPACES
*
* RESTORE ORIGINAL LINE AND RETURN
*
M2      MOV  R9,R0        VDP START OF LINE 23
        LI   R1,B1        ORIGINAL SCREEN LINE BUFFER
        LI   R2,32        32 BYTES TO MOVE
```

```
BLWP @MW          RESTORE SCREEN LINE 23
MOVB @H1+1,@>837C CLR GPL STATUS
MOVB @SN,@SP      RESTORE ACTIVE SPRITE NO.
LWPI >83E0        LOAD GPL WS
MOV  @SV,R11      RESTORE RTN ADDRESS
RT                RETURN TO BASIC
END
```

Disk 1. Contents of File VDP/S

```
       DEF  VPEEK,VPOKE
STRREF EQU  >2014
STRASG EQU  >2010
ERR    EQU  >2034
VDPRD  EQU  >8800
VDPWA  EQU  >8C02
VDPWD  EQU  >8C00
GPLWS  EQU  >83E0
XMLLNK EQU  >2018
NUMREF EQU  >200C
STATUS EQU  >837C
FAC    EQU  >834A
WS     BSS  32
ST     BSS  256
SAVRTN DATA 0
FLAG   DATA 0
EMSG   DATA >1E00
H255   DATA >00FF
TOP    DATA >3FFF
VPEEK  CLR  @FLAG       FLAG =0 FOR PEEK
       JMP  START
VPOKE  MOV  @EMSG,@FLAG FLAG<>0 FOR POKE
START  MOV  R11,@SAVRTN
       LWPI WS
       CLR  R0
       LI   R1,1
       BLWP @NUMREF     GET VDP ADDR
       BLWP @XMLLNK     CVT TO INTEGER
       DATA >12B8
       MOV  @FAC,R3
       MOV  R3,R3
       JLT  ERROR       ERROR IF <0
       MOV  @FLAG,@FLAG
       JEQ  N1
       LI   R1,2        IF VPOKE, 1ST
       MOVB @H255+1,@ST BYTE OF STRING
       LI   R2,ST       IS LENGTH
       BLWP @STRREF
       CLR  R4
       MOVB @ST,R4      LENGTH INTO R4
       SWPB R4
       JMP  N2
N1     LI   R1,2        GET # OF BYTES
       BLWP @NUMREF     FOR VPEEK ROUTN
       BLWP @XMLLNK
       DATA >12B8
       MOV  @FAC,R4
N2     MOV  R4,R4       IF # BYTES=0 ...
```

```
        JEQ  ERROR
        C    R4,@H255    OR >255, ERROR
        JH   ERROR
        MOV  R3,R5       IF LAST ADDR IS
        A    R4,R5       PAST VDP RAM TOP
        C    R5,@TOP     THEN ERROR
        JH   ERROR
        MOV  @FLAG,@FLAG
        JEQ  N3
        AI   R3,>4000    IF POKE, WRITE
N3      SWPB R4
        MOVB R4,@ST      INDICATE # BYTES
        SWPB R3
        MOVB R3,@VDPWA
        SWPB R3
        MOVB R3,@VDPWA
        SWPB R4
        LI   R2,ST+1     POINTER TO STRNG
        MOV  @FLAG,@FLAG
        JEQ  N5
N4      MOVB *R2+,@VDPWD WRITE TO VDP RAM
        DEC  R4
        JNE  N4
        JMP  N6
N5      MOVB @VDPRD,*R2+ READ VDP RAM
        DEC  R4
        JNE  N5
        CLR  R0           WRITE STRING
        LI   R1,3         BACK TO BASIC
        LI   R2,ST
        BLWP @STRASG
N6      MOVB R0,@STATUS   CLEAN UP FOR
        LWPI GPLWS        RETURN TO BASIC
        MOV  @SAVRTN,R11
        B    *R11
ERROR   MOV  @EMSG,R0
        BLWP @ERR
        END
```

Disk 2. Disk Copy and Disassemble

Version: *Author*: John Clulow
Requires: EA *Language*: AL *Updated*:

Copy utility similar to Masscopy, but can initialize the target disk and handle all disk formals. Very fast dis-assembler with source code for XB, EA, and Mini-Mem — adds labels in a second pass.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-2
Sectors total           = 360
Sectors used            = 232
Sectors available       = 126
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00e COPY         48 DIS/FIX 80   >0d6 033 >116 009 >010 005
002 >005 DISASSM/S    80 DIS/VAR 80   >073 020 >088 058 >0d4 001
003 >00c DSRLNK        7 DIS/FIX 80   >05a 006
004 >00d E/ADIS       20 DIS/FIX 80   >060 019
005 >006 MMDIS        20 DIS/FIX 80   >0c2 018 >0d5 001
006 >00f READ-THIS    17 DIS/VAR 80   >022 014 >056 002
007 >003 XBDIS        40 DIS/FIX 80   >030 038 >087 001
```

Disk 2. Contents of File DISASSM/S

```
****************************************
*
* DISASSEMBLER- Modified from TI original
*               5/19/84 J. Clulow - NHUG
*
* E/A version - use REFs and delete
*               lines 33-44
*
* MINI-MEM    - No changes required
*
* Ext BASIC   - use Ext BASIC EQUates
*               on 33-44 and remove *
*               from lines 217-222
*
*  Note: Use LOAD AND RUN option with
*        E/A and MINIMEM with Program
*        Name START
*
*  Ext BASIC: CALL INIT::CALL LOAD("DSK1
*             .DSRLNK","DSK1.XBDIS")
*
*             CALL LINK("START")
*
*----------------------------------------
*
* Use REF statements for ED/ASSM version
*
*      REF  KSCAN,VDPWA,VDPWD,GPLWS,VDPRD
*      REF  DSRLNK,VSBW,VMBW,VMBR,GRMRA,GRMWA
       DEF  START
*
* USE THE EQU's FOR XBASIC AND MINIMEM
*
KSCAN  EQU  >6020 }M >201C }E
VDPWA  EQU  >8C02 }I >8C02 }X
VDPWD  EQU  >8C00 }N >8C00 }T
GPLWS  EQU  >83E0 }I >83E0 }
VDPRD  EQU  >8800 }  >8800 }B
DSRLNK EQU  >6038 }M >2532 }A ->X BASIC
VSBW   EQU  >6024 }E >2020 }S MUST LOAD
VMBW   EQU  >6028 }M >2024 }I DSRLNK
VMBR   EQU  >6030 }O >202C }C FIRST!!
GRMRA  EQU  >9802 }R >9802 }
GRMWA  EQU  >9C02 }Y >9C02 }
NAM1
       TEXT 'LI  '       0200
       TEXT 'AI  '       0220
       TEXT 'ANDI'       0240
```

```
        TEXT 'ORI '       0260
        TEXT 'CI  '       0280
        TEXT 'STWP'       02A0
        TEXT 'STST'       02C0
        TEXT 'LWPI'       02E0
        TEXT 'LIMI'       0300
        TEXT 'LMF '       0320
        TEXT 'IDLE'       0340
        TEXT 'RSET'       0360
        TEXT 'RTWP'       0380
        TEXT 'CKON'       03A0
        TEXT 'CKOF'       03C0
        TEXT 'LREX'       03E0
NAM2
        TEXT 'BLWP'       0400
        TEXT 'B   '       0440
        TEXT 'X   '       0480
        TEXT 'CLR '       04C0
        TEXT 'NEG '       0500
        TEXT 'INV '       0540
        TEXT 'INC '       0580
        TEXT 'INCT'       05C0
        TEXT 'DEC '       0600
        TEXT 'DECT'       0640
        TEXT 'BL  '       0680
        TEXT 'SWPB'       06C0
        TEXT 'SETO'       0700
        TEXT 'ABS '       0740
        TEXT 'LDS '       0780
        TEXT 'LDD '       07C0
NAM3
        TEXT 'SRA '       0800
        TEXT 'SRL '       0900
        TEXT 'SLA '       0A00
        TEXT 'SRC '       0B00
NAM4
        TEXT 'JMP '       1000
        TEXT 'JLT '       1100
        TEXT 'JLE '       1200
        TEXT 'JEQ '       1300
        TEXT 'JHE '       1400
        TEXT 'JGT '       1500
        TEXT 'JNE '       1600
        TEXT 'JNC '       1700
        TEXT 'JOC '       1800
        TEXT 'JNO '       1900
        TEXT 'JL  '       1A00
        TEXT 'JH  '       1B00
        TEXT 'JOP '       1C00
        TEXT 'SBO '       1D00
```

```
       TEXT 'SBZ '       1E00
       TEXT 'TB  '       1F00
NAM5
       TEXT 'COC '       2000
       TEXT 'CZC '       2400
       TEXT 'XOR '       2800
       TEXT 'XOP '       2C00
       TEXT 'LDCR'       3000
       TEXT 'STCR'       3400
       TEXT 'MPY '       3800
       TEXT 'DIV '       3C00
NAM6
       TEXT 'SZC '       4000
       TEXT 'SZCB'       5000
       TEXT 'S   '       6000
       TEXT 'SB  '       7000
       TEXT 'C   '       8000
       TEXT 'CB  '       9000
       TEXT 'A   '       A000
       TEXT 'AB  '       B000
       TEXT 'MOV '       C000
       TEXT 'MOVB'       D000
       TEXT 'SOC '       E000
       TEXT 'SOCB'       F000
AORG   TEXT '       AORG >'
END    TEXT '       END'
*
STATUS EQU  >837C
KEY    EQU  >8375
RESET  EQU  >6A
PABPTR EQU  >8356
PAB    EQU  >1000
*******************************
* DATA
*******************************
FIRST  DATA  0
LAST   DATA >80
SAVG   DATA 0
FFLAG  DATA 0
FLOOP  DATA 0
LBL    DATA 0
SAVRTN DATA 0
COMMA  BYTE ','
STAR   BYTE '*'
AT     BYTE '@'
PLUS   BYTE '+'
MINUS  BYTE '-'
GREAT  BYTE '>'
OPEN   BYTE '('
CLOSE  BYTE ')'
SPACE  BYTE ' '
```

```
CH1    BYTE '1'
SMALLZ BYTE 'z'
PABC   DATA >0012
       DATA PAB+33  80 BYTE VDP RAM BUFFER
       BYTE 80,80
       DATA 0,0
BLANKS TEXT '    '
DATA   TEXT 'DATA'
SCRBUF BSS  40      SCROLL BUFFER
NAMBUF BSS  80
WS     BSS  32
HEX    TEXT '0123456789ABCDEF'
TITLE  TEXT '9900 DISASSEMBLER'
H00    BYTE 0
COPY   TEXT '(C) 1981 BY TEXAS INSTRUMENTS'
       BYTE 0
PRM1   TEXT 'START ADDRESS:>'
       BYTE 0
PRM2   TEXT 'END ADDRESS:  >'
       BYTE 0
PRM3   TEXT 'OUTPUT DEVICE NAME OR <CR>'
       BYTE 0
FAIL   TEXT 'CANNOT OPEN DEVICE'
       BYTE 0
*
* START OF PROGRAM
*
       EVEN
START
* SAVE GROM ADDRESS, IN CASE OF RS232
       MOVB @GRMRA,@SAVG
       MOVB @GRMRA,@SAVG+1
       DEC  @SAVG
       MOV  R11,@SAVRTN
       LWPI WS
       LI   R15,VDPWD
       CLR  @FLOOP       FIRST ITERATION
       LI   R1,40*7+2
       MOV  R1,@START3   MODIFY CODE
       LI   R1,40*8+2
       MOV  R1,@START4+2 MODIFY CODE
* CLEAR SCREEN
START2 LI   R5,>4000
       BL   @VAD
       LI   R1,40*24
CS1    MOVB @SPACE,*R15
       DEC  R1
       JGT  CS1
       MOV  @FLOOP,@FLOOP
       JNE  PCHX
```

```
* SELECT TEXT MODE
       LI   R1,>F081
       MOVB R1,@>83D4     FOR KEYSCAN
       MOVB R1,@VDPWA
       SWPB R1
       MOVB R1,@VDPWA
       LI   R1,>1787      COLOR
       MOVB R1,@VDPWA
       SWPB R1
       MOVB R1,@VDPWA
       LI   R1,>0184
       MOVB R1,@VDPWA
       SWPB R1
       MOVB R1,@VDPWA
*
* USE FOLLOWING 6 LINES FOR XBASIC ONLY
*
*      LI   R0,>400     ; LOAD CHAR DEFS
*      LI   R1,OPER     ;
*      LI   R2,1024     ; - EXT BASIC-
*      BLWP @VMBR       ;
*      LI   R0,>900     ;    ONLY
*      BLWP @VMBW       ;
*
       BLWP @LBLST  - INITIALIZE ARRAYS
* PUT UP TITLE
PCHX   BL   @MESG
       DATA 48,TITLE
* PUT UP COPYRIGHT
       BL   @MESG
       DATA 40*23+5,COPY
* PROMPT FOR START ADDRESS
       BL   @MESG
       DATA 40*3+2,PRM1
       MOV  @FLOOP,@FLOOP
       JEQ  STD1
       LI   R5,40*3+17
       ORI  R5,>4000
       BL   @VAD
       MOV  @FIRST,R3
       BL   @PUTHX2
       JMP  STD2
STD1   BL   @INNUM
       DATA 40*3+17,FIRST
* PROMPT FOR ENDING ADDRESS
STD2   BL   @MESG
       DATA 40*5+2,PRM2
       MOV  @FLOOP,@FLOOP
       JEQ  STD3
       LI   R5,40*5+17
       ORI  R5,>4000
```

```
       BL   @VAD
       MOV  @LAST,R3
       BL   @PUTHX2
       JMP  STD4
STD3   BL   @INNUM
       DATA 40*5+17,LAST
       JMP  STD5
STD4   BL   @MESG
       DATA 40*7+2,PRM3
       LI   R0,40*8+2
       LI   R1,NAMBUF
       LI   R2,80
       BLWP @VMBW
* PROMPT FOR OUTPUT DEVICE
STD5   BL   @MESG
START3 DATA 40*7+2,PRM3
       LI   R1,NAMBUF
       LI   R2,80
       LI   R3,>2020
* CLEAR FILE NAME
FN1    MOV  R3,*R1+
       DECT R2
       JGT  FN1
START4 LI   R0,40*8+2
       LI   R1,NAMBUF
       LI   R2,80
FN2    CLR  R3
       CLR  R4
FN3    BLWP @VMBW
       BLWP @KSCAN
       MOVB @STATUS,R4
       JEQ  FN3
       MOVB @KEY,R4
       CI   R4,>0D00   ENTER
       JEQ  FN9
       CI   R4,>0800   BACKSPACE
       JNE  FN4
       DEC  R3
       JLT  FN2        LEFT MARGIN
* MUST BACK UP
       MOVB @SPACE,@NAMBUF(R3)
       JMP  FN3
FN4    MOVB R4,@NAMBUF(R3)
       CI   R3,79
       JHE  FN3
       INC  R3
       JMP  FN3
* FINISHED NAME
FN9    MOV  R3,@FFLAG    NONZERO MEANS FILE IN USE
       JEQ  XLATE
```

```
       SWPB R3
       MOVB R3,@PABC+9
       LI   R0,PAB
       LI   R1,PABC
       LI   R2,10
       BLWP @VMBW          COPY TO VDP
       LI   R0,PAB+10      MOVE NAME ALSO
       LI   R1,NAMBUF
       MOV  @FFLAG,R2
       BLWP @VMBW
       LI   R0,PAB+9
       MOV  R0,@PABPTR
* TRY TO OPEN FILE
       BLWP @DSRLNK
       DATA 8
       JEQ  FAILED
       LI   R0,PAB+33
       LI   R1,AORG
       LI   R2,13
       BLWP @VMBW
       LI   R5,PAB+46+>4000
       BL   @VAD
       MOV  @FIRST,R3
       BL   @PUTHX2
       MOV  @FIRST,R9
       B    @WRITE
* FAILED
FAILED BL   @MESG
       DATA 40*12+2,FAIL
       MOV  @CH1,@FLOOP
       B    @WAIT
*
XLATE  MOV  @FIRST,R9

TOP    C    R9,@LAST
       JLE  TOP1
       ABS  @FFLAG
       JEQ  WAIT
       BL   @NEWLIN
       LI   R0,PAB+38
       LI   R1,END
       LI   R2,10
       BLWP @VMBW
       LI   R1,>0300
       LI   R0,PAB
       BLWP @VSBW
       AI   R0,9
       MOV  R0,@PABPTR
       BLWP @DSRLNK
       DATA 8
* GO BACK
```

```
WAIT   LI   R1,>0100   CLOSE
       LI   R0,PAB
       BLWP @VSBW
       AI   R0,9
       MOV  R0,@PABPTR
       BLWP @DSRLNK
       DATA 8       (IGNORE ERROR)
WT1    BLWP @KSCAN
       MOVB @STATUS,R0
       JEQ  WT1
       MOV  @FLOOP,@FLOOP
       JNE  STD0
       INC  @FLOOP
       LI   R1,40*11+2
       MOV  R1,@START3    MODIFY CODE
       LI   R1,40*12+2
       MOV  R1,@START4+2  MODIFY CODE
       B    @START2
* RESTORE PATERN MODE
STD0   LI   R1,>E081
       MOVB R1,@>83D4     FOR KEYSCAN
       MOVB R1,@VDPWA
       SWPB R1
       MOVB R1,@VDPWA
* RESTORE GROM ADDRESS
       MOVB @SAVG,@GRMWA
       MOVB @SAVG+1,@GRMWA
       CLR  R1
       MOVB R1,@STATUS
       LWPI GPLWS
       MOV  @SAVRTN,R11
       B    *R11
TOP1
       MOV  R9,R8         SAVE POINTER
       BL   @NEWLIN
       MOV  R9,R3
       MOV  @FLOOP,@FLOOP
       JEQ  PCH2
       BLWP @LBLCK
       MOV  @LBL,@LBL
       JEQ  PCH3
       MOVB @LBL,*R15
       SWPB @LBL
       MOVB @LBL,*R15
       BL   @SP5
       JMP  PCH4
PCH3   BL   @SPLBL
       JMP  PCH4
PCH2   BL   @PUTHX2    WRITE LOCATION
       LI   R0,>2000
```

```
       MOVB R0,*R15    SKIP SOME SPACE
       MOVB R0,*R15
       MOVB R0,*R15
PCH4   MOV  *R9+,R2    FETCH INSTRUCTION
       CI   R2,>4000   FORMAT1?
       JL   TOP2
* FORMAT1 HERE
       MOV  R2,R3
       SRL  R3,12      SAVE 4 BITS
       SLA  R3,2       MPY BY 4
       AI   R3,NAM6-16    R3 POINTS TO MNEMONIC
       BL   @PUTOP
       BL   @GS
       MOVB @COMMA,*R15
       BL   @GD
       B    @NEXT
TOP2
       CI   R2,>2000    FORMATS 3,4,9?
       JL   TOP3
* FORMATS 3,4,9 ARE THE SAME
       MOV  R2,R3
       SRL  R3,10      SAVE 6 BITS (TOP TWO 0)
       SLA  R3,2       MPY BY 4
       AI   R3,NAM5-32   R3 POINTS TO MNEMONIC
       BL   @PUTOP
       BL   @GS
       MOVB @COMMA,*R15
       BL   @RD
       B    @NEXT
TOP3
       CI   R2,>1000    FORMAT2
       JL   TOP4
* FORMAT 2 - JUMPS AND BITS
       MOV  R2,R3
       SRL  R3,8        SAVE 8 BITS (TOP 3 ARE ZERO)
       SLA  R3,2        MPY BY 4
       AI   R3,NAM4->40  R3 POINTS TO MNEMONIC
       BL   @PUTOP
       MOV  R2,R3
       SLA  R3,8
       SRA  R3,8        SIGN EXTEND
       CI   R2,>1D00    JUMP?
       JL   TOP35
* BIT OPERATION HERE
       BL   @PUTSGN
       B    @NEXT
* JUMP HERE
TOP35
       SLA  R3,1        FOR BYTE OFFSET
       A    R9,R3       RELATIVE TO NEXT INSTRUCTION
       MOV  @FLOOP,@FLOOP
```

```
       JEQ  PCH5
       BLWP @LBLCK
       MOV  @LBL,@LBL
       JEQ  TOP36
       MOVB @LBL,*R15
       SWPB @LBL
       MOVB @LBL,*R15
       B    @NEXT
PCH5   BLWP @LBLAD
TOP36  BL   @PUTHEX
       B    @NEXT
TOP4
       CI   R2,>800     FORMAT5?
       JL   TOP5
* FORMAT5 HERE - SHIFTS AND UNKNOWN
       CI   R2,>C00
       JHE  TOP7         UNKNOWN
       MOV  R2,R3
       SRL  R3,8         SAVE 8 BITS (TOP 4 ZERO)
       SLA  R3,2         MPY BY 4
       AI   R3,NAM3->20  R3 POINTS TO MNEMONIC
       BL   @PUTOP
       BL   @RS           REGISTER OPERAND
       MOVB @COMMA,*R15
       SLA  R2,2          TRICK TO MAKE COUNT LOOK LIKE REGISTER DEST
       BL   @RD
       B    @NEXT
TOP5
       CI   R2,>400      FORMAT6?
       JL   TOP6
* FORMAT 6 HERE
       MOV  R2,R3
       SRL  R3,6
       SLA  R3,2
       AI   R3,NAM2->40
       BL   @PUTOP
       BL   @GS
       B    @NEXT
TOP6
       CI   R2,>200     ANYTHING ELSE?
       JL   TOP7        FOR DATA
* FORMATS 7,8,AND 10
       MOV  R2,R3
       SRL  R3,5
       SLA  R3,2
       AI   R3,NAM1->40
       BL   @PUTOP
       CI   R2,>340     FORMAT7?
       JHE  NEXT      YES - NO OPERAND
*      CI   R2,>320     FORMAT10?
```

```
*      JHE  ?????     YES - TREAT SPECIALLY
* ALL OF FORMAT 8 EXCEPT LWPI AND LIMI HAVE WORKSPACE "SOURCE"
       CI   R2,>2E0
       JHE  TOP63
       BL   @RS
* STWP AND STST HAVE NO IMMEDIATE OPERAND
       CI   R2,>2A0
       JHE  NEXT
       MOVB @COMMA,*R15
TOP63
       MOV  *R9+,R3      GET IMMEDIATE OPERAND
       MOV  @FLOOP,@FLOOP
       JEQ  PCH7
       BLWP @LBLCK
       MOV  @LBL,@LBL
       JEQ  TOP64
       MOVB @LBL,*R15
       SWPB @LBL
       MOVB @LBL,*R15
       JMP  NEXT
PCH7   BLWP @LBLAD
TOP64  BL   @PUTHEX
       JMP  NEXT
TOP7   LI   R3,DATA      UNRECOGNIZED -- DISPLAY AS DATA
       BL   @PUTOP
       MOV  R2,R3
       BL   @PUTHEX
NEXT
* PUT OUT FIRST WORD OF INSTRUCTION
       LI   R5,40*23+>4000+34
       BL   @VAD
       MOV  *R8,R3
       BL   @PUTHEX
* NOW WRITE THE RECORD, IF FILE IS OPEN
       ABS  @FFLAG
       JEQ  NEXT0
       LI   R0,40*23
       LI   R1,SCRBUF
       LI   R2,33
       BLWP @VMBR
       LI   R0,PAB+33
       BLWP @VMBW
       LI   R5,>4000+PAB+67
       BL   @VAD
       MOV  R8,R3
       BL   @PUTHEX
       BL   @SP3
       MOV  R8,R4
NPCH1  MOV  *R4+,R3
       BL   @PUTHEX
       MOVB @SPACE,*R15
```

```
        C    R4,R9
        JNE  NPCH1
        LI   R5,>4000+PAB+96
        BL   @VAD
        MOV  R8,R4
NPCH5   MOV  *R4+,R3
        LI   R1,2
NPCH4   CB   R3,@SPACE
        JL   NPCH2
        CB   R3,@SMALLZ
        JH   NPCH2
        MOVB R3,*R15
        JMP  NPCH3
NPCH2   MOVB @STAR,*R15
NPCH3   SWPB R3
        DEC  R1
        JNE  NPCH4
        C    R4,R9
        JNE  NPCH5
WRITE   LI   R1,>0300   WRITE
        LI   R0,PAB
        BLWP @VSBW
        AI   R0,9
        MOV  R0,@PABPTR
        BLWP @DSRLNK    WRITE THE RECORD
        DATA 8
        JNE  NEXT0
        MOVB @CH1,@FLOOP
        B    @WAIT       ERROR EXIT (FOR NOW)
NEXT0   BLWP @KSCAN
        MOVB @STATUS,R0
        JEQ  NEXT2
*PAUSE
NEXT1   BLWP @KSCAN
        MOVB @STATUS,R0
        JEQ  NEXT1
NEXT2   B    @TOP  COULD DO MORE, LIKE PUT LINE TO DISK
NEWLIN
        LI   R4,23  LINES TO SCROLL
        LI   R5,40  FIRST READ LOCATION
        MOV  R11,R10
NEW1
        BL   @VAD
        LI   R6,SCRBUF
        LI   R7,40
NEW2    MOVB @VDPRD,*R6+
        DEC  R7
        JGT  NEW2
        AI   R5,>4000-40
        BL   @VAD
```

```
       AI   R5,->4000+80  NEXT LINE
       LI   R6,SCRBUF
       LI   R7,40
NEW3   MOVB *R6+,*R15
       DEC  R7
       JGT  NEW3
       DEC  R4        ALL DONE?
       JGT  NEW1
* NOW THE BLANK LINE
       LI   R5,40*23+>4000
       BL   @VAD
       LI   R6,>2000
       LI   R7,40
NEW4   MOVB R6,*R15
       DEC  R7
       JGT  NEW4
       LI   R5,PAB+33+>4000
       BL   @VAD
       LI   R6,>2000
       LI   R7,80
NEW5   MOVB R6,*R15
       DEC  R7
       JGT  NEW5
* FINALLY, SET UP TO WRITE LAST LINE
       LI   R5,40*23+>4000
       BL   @VAD
       B    *R10
VAD
       SWPB R5
       MOVB R5,@VDPWA
       SWPB R5
       MOVB R5,@VDPWA
       RT
PUTOP
* MOVE 4 BYTES POINTED TO BY R3 TO SCREEN, FOLLOWED BY A BLANK
       LI   R4,4
PO2    MOVB *R3+,*R15
       DEC  R4
       JGT  PO2
       MOVB @SPACE,*R15
       RT
PUTHEX
       MOVB @GREAT,*R15
PUTHX2
       LI   R7,4
PH1    SRC  R3,12
       MOV  R3,R5
       ANDI R5,>F
       MOVB @HEX(R5),*R15
       DEC  R7
       JGT  PH1
```

```
       RT
GS
       MOV  R2,R3
       ANDI R3,>30    TS
       SRL  R3,3      DOUBLED TS
GS0    MOV  R2,R4
       ANDI R4,>F     S
GS1    MOV  @GSTAB(R3),R5
       B    *R5
GSTAB  DATA GSR,GSRI,GSL,GSII
* REGISTER SOURCE
GSR
       CI   R4,9
       JLE  GSR1
       MOVB @CH1,*R15
       AI   R4,-10
GSR1   AI   R4,>30
       SWPB R4
       MOVB R4,*R15
       RT
GSRI   MOVB @STAR,*R15
       JMP  GSR
GSL    MOV  *R9+,R3     PICK UP ADDRESS OF SOURCE
       MOV  R11,R10
       MOVB @AT,*R15
       MOV  @FLOOP,@FLOOP
       JEQ  PCH9
       BLWP @LBLCK
       MOV  @LBL,@LBL
       JEQ  GSL2
       MOVB @LBL,*R15
       SWPB @LBL
       MOVB @LBL,*R15
       JMP  GSL3
PCH9   BLWP @LBLAD
GSL2   BL   @PUTHEX
GSL3   MOV  R4,R4      INDEXED?
       JEQ  GSL1
       MOVB @OPEN,*R15
       BL   @GSR
       MOVB @CLOSE,*R15
GSL1   B    *R10       RETURN
GSII   MOV  R11,R10
       MOVB @STAR,*R15
       BL   @GSR
       MOVB @PLUS,*R15
       B    *R10       RETURN
* GENERAL DESTINATION
GD
       MOV  R2,R3
```

```
       ANDI R3,>C00     TD
       SRL  R3,9        TD*2
GD1    MOV  R2,R4
       ANDI R4,>3C0      D
       SRL  R4,6
       B    @GS1        SHARE CODE
* REGISTER SOURCE
RS
       CLR  R3         FORCE TS=0
       B    @GS0       SHARE CODE
* REGISTER DESTINATION
RD
       CLR  R3
       B    @GD1       SHARE CODE
* PUT A SIGNED BYTE NUMBER
PUTSGN
       MOV  R3,R3
       JLT  PS2
PS1    MOVB @GREAT,*R15
       MOV  R3,R4
       SRL  R4,4    FIRST NYBBLE
       MOVB @HEX(R4),*R15
       ANDI R3,>F
       MOVB @HEX(R3),*R15
       RT
PS2    MOVB @MINUS,*R15
       NEG  R3
       JMP  PS1
*
* PRINT A MESSAGE
*
MESG
       MOV  *R11+,R5   VDP ADDRESS
       MOV  *R11+,R2   MESSAGE ADDRESS
       MOV  R11,R10    SAVE RETURN
       ORI  R5,>4000
       BL   @VAD
MSG1   CB   *R2,@H00
       JEQ  MSG2
       MOVB *R2+,*R15
       JMP  MSG1
MSG2   B    *R10
*
* GET A NUMBER (IN HEX)
*
INNUM
       MOV  *R11+,R5
       MOV  *R11+,R4   DESTINATION
       MOV  R11,R10   SAVE RETURN
       ORI  R5,>4000
       CLR  R0
```

```
       CLR  R1
INN1   BLWP @KSCAN
       MOVB @STATUS,R0
       JEQ  INN1
       MOVB @KEY,R0
       CI   R0,>D00      ENTER?
       JEQ  INN9
       CI   R0,>3000
       JL   INN1        < '0'
       CI   R0,>3900
       JH   INN3
* VALID HEX DIGIT
       BL   @VAD
       MOVB R0,*R15    ECHO TO SCREEN
       SRL  R0,8
       AI   R0,->30
INN2   SLA  R1,4
       A    R0,R1
       CLR  R0
       INC  R5
       JMP  INN1
INN3   CI   R0,>4100
       JL   INN1         < 'A'
       CI   R0,>4600
       JH   INN1         > 'F'
       BL   @VAD
       MOVB R0,*R15
       SRL  R0,8
       AI   R0,->37
       JMP  INN2
INN9   MOV  R1,*R4
       B    *R10
SPLBL  MOVB @SPACE,*R15
       MOVB @SPACE,*R15
SP5    MOVB @SPACE,*R15
       MOVB @SPACE,*R15
SP3    MOVB @SPACE,*R15
       MOVB @SPACE,*R15
       MOVB @SPACE,*R15
       B    *R11
*
* BLWP UTILITIES
*
LBLST  DATA LBWS
       DATA LBST
LBLAD  DATA LBWS
       DATA LBAD
LBLCK  DATA LBWS
       DATA LBCK
LZ     BYTE >5A
```

```
L9     BYTE >39
LIMIT  DATA 260
COUNT  DATA 0
OPER   BSS  520
LABEL  BSS  520
LBWS   BSS  32
*
* INITIALIZE
*
LBST   CLR  R1
       MOV  R1,@COUNT
       LI   R6,518
       LI   R2,26
       MOV  @LZ,R4
LS1    LI   R3,10
LS2    MOV  R4,@LABEL(R6)
       DECT R6
       DEC  R4
       DEC  R3
       JNE  LS2
       SWPB R4
       MOVB @L9,R4
       DEC  R4
       SWPB R4
       DEC  R2
       JNE  LS1
LS4    LI   R3,>FFFF
       LI   R1,260
       LI   R2,OPER
LS3    MOV  R3,*R2+
       DEC  R1
       JNE  LS3
       RTWP
*
* ADD LABELS
*
LBAD   MOV  @6(R13),R3
       C    R3,@FIRST
       JL   NOAD
       C    R3,@LAST
       JH   NOAD
       MOV  @COUNT,@COUNT
       JEQ  AD
       C    @COUNT,@LIMIT
       JEQ  NOAD
       MOV  @COUNT,R2
       CLR  R1
LA1    C    R3,@OPER(R1)
       JEQ  NOAD
       INCT R1
       DEC  R2
```

```
       JNE  LA1
AD     MOV  @COUNT,R1
       SLA  R1,1
       MOV  R3,@OPER(R1)
       INC  @COUNT
NOAD   RTWP
*
* CHECK FOR LABEL
*
LBCK   MOV  @6(R13),R3
       C    R3,@FIRST
       JL   NOFND
       C    R3,@LAST
       JH   NOFND
       CLR  R1
       MOV  @COUNT,R2
LC1    C    R3,@OPER(R1)
       JEQ  FNDIT
       INCT R1
       DEC  R2
       JNE  LC1
NOFND  CLR  @LBL
       RTWP
FNDIT  MOV  @LABEL(R1),@LBL
       RTWP
       END
```

Disk 2. Contents of File READ-THIS

Extended BASIC A/L Routines — New Horizons Users Group

DISASSEMBLER — (Texas Instruments) (mod. J. Clulow)

Given a starting and ending address in hexadecimal, the program determines assembly language equivalents of machine code. On pass one the result is displayed on the screen and optionally transferred to disk or printer. The left column contains the hex memory location in which the instruction starts. This is followed by the corresponding assembly language mnemonics and operands. The last column contains the first word of machine code at the address.

A second pass option was added to this program in which labels are inserted in the output. Up to 260 labels are available (A0 - A9 ... Z9). This allows easier modification of output for re-assembly if the objective is to produce relocatable code. However, it is unlikely that disassembled code will re-assemble without error. If an operand is in the range of addresses to be disassembled but if it does not represent the first word of an instruction, then it will be an undefined symbol. After the first attempt at assembly, it is an easy matter to edit the source file, and make the necessary operand changes from the machine code in the comment field.

It is for this reason that the program was also modified to output full machine code in addition to the starting address in both passes and, in the right column, any displayable ASCII characters are printed — non-displayable characters are indicated with an asterisk *.

Three object programs are provided for various configurations: Mini Memory (MMDIS), Extended BASIC (XBDIS), and Editor/Assembler (E/ADIS). In the E/A or Mini Memorycases, the appropriate object file should be loaded with the LOAD AND RUN option and executed with the program name START. With Extended BASIC the load sequence is:

```
CALL INIT
CALL LOAD("DSK1.DSRLNK","DSK1.XBDIS")
CALL LINK("START")
```

The source program is DISASSM/S. When assembled, this version yields the Mini Memory object file. The source code contains full instructions for producing the Ext BASIC and E/A versions if required.

The MMDIS version will fit in the Mini Memory. It can then be used to disassemble programs loaded into the memory expansion. First initialize the Mini Memory. Then in TI BASIC enter the statement:

```
CALL LINK(28706,0,0,0,0,0,0,0,0)
```

Now load the program with the LOAD AND RUN option. Finally, enter the TI BASIC statement:

```
CALL LINK(28706,160,0,255,224,32,0,63,255)
```

The first CALL LINK forces the loader to load MMDIS into the Mini Memory module, and the second CALL LINK restores the pointers used by the loader to allow subsequent code to go into the 32K memory expansion unit.

Disk 3. Printer Utility Routines

Version: *Author*: Many
Requires: XB, TIW *Language*: XB, AL *Updated*:

Includes improved version of printer dump from disk 1, files of transliterate commands for TI-Writer, program to address envelopes in multiple type-faces, a program to list entire symbol table to your printer, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-3
Sectors total           = 360
Sectors used            = 141
Sectors available       = 217
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >006 DUMP-XB       4 DIS/VAR163   >05b 003
002 >005 DUMPJPH      18 DIS/FIX 80   >04c 015 >096 002
003 >004 DUMPJPH/S    44 DIS/VAR 80   >02a 034 >08d 009
004 >007 ENVELOPE     22 PROGRAM      >05e 021
005 >00c LISTING      12 DIS/VAR 80   >0a4 011
006 >009 READ-THIS    10 DIS/VAR 80   >081 009
007 >00a SECTOR-ASM    6 DIS/VAR 80   >08a 003 >098 002
008 >008 SYMBOLDUMP   15 PROGRAM      >073 014
009 >002 TYPE          3 DIS/VAR 80   >022 002
010 >003 TYPE2         7 DIS/VAR 80   >024 006
```

Disk 3. Contents of File DUMPJPH/S

```
       DEF  DUMP,START,INIT
*
NUMREF EQU  >200C    {
STRREF EQU  >2014    {
ERR    EQU  >2034    {
ERCODE EQU  >1E1C    {
WS     BSS  32
S1     BSS  2
IN     BSS  8
DO     BSS  512
LINE   DATA 0            JPH DATA HOLDER
SAVRTN DATA 0
MK     DATA >001F
PD     DATA >0012,>1E00,>FF00,>0000
SP     DATA >0006
       TEXT 'PIO.CR'
       EVEN
CR     DATA >0D0A
E1     DATA >1B4C,>0002
E2     DATA >001B,>4108
D6     DATA >4019

************************
* GET PRINTER SPEC
*
INIT   MOV  R11,@SAVRTN
       LI   R0,>0017
       MOV  R0,@SP           23 BYTES MAXIMUM
       CLR  R0
       LI   R1,1             SECOND PARAMETER
       LI   R2,SP+1
       BLWP @STRREF          GET PRINTER SPEC.
       MOV  @SAVRTN,R11
       RT

************************

START
       LI   R1,INTER
       MOV  R1,@>83C4
       RT

INTER
       MOV  R11,@SAVRTN
       LIMI 0
       LWPI WS

       LI   R0,>0E00
```

```
       CB   R0,@>8375
       JNE  EXIT

       SETO R0
       MOVB R0,@>8375    CLEAR BYTE OF KEY PRESSED
       JMP  DUMP0

EXIT   LWPI >83E0
       MOV  @SAVRTN,R11
       RT

************************

QUIT   B    @CLOSE

*******

DUMP   MOV  R11,@SAVRTN
       LIMI 0

DUMP0  LWPI WS

       CLR  R8
       CLR  R9                STARTING SCREEN POSITION
       CLR  R0
       LI   R9,-1         JPH WAS HERE!
       MOV  R9,@LINE

*
* SET UP PAB
*
NEXTL  LI   R0,>1D00
       LI   R1,PD
       MOV  @SP,R2
       AI   R2,10             NO OF PAB BYTES TO MOVE
       BLWP @VMBW             WRITE PAB TO VDP RAM
       LI   R6,>1D09
       MOV  R6,@>8356         POINT TO DEVICE NAME LENGTH
       BLWP @DSRLNK           DSRLNK TO OPEN PRINTER
       DATA 8
       JNE  GO
       MOVB @S1,@>9C02
       SWPB @S1
       MOVB @S1,@>9C02
       LI   R0,ERCODE
       SWPB R0
       BLWP @ERR
       JNE  GO                CHECK FOR PRINTER SPEC ERROR
       BLWP @ERR              RETURN I/O ERROR

GO     INC  @LINE
```

```
       MOV  @LINE,R9
       CI   R9,24          JPH WAS HERE
       JEQ  QUIT
       CLR  R8
       CLR  R0

       SLA  R9,5
       MOVB @>9802,@S1
       SWPB @S1
       MOVB @>9802,@S1
       SWPB @S1
       DEC  @S1

       LI   R0,>1D00
       LI   R1,>0300
       BLWP @VSBW              PUT WRITE OP CODE IN PAB
       LI   R0,>1D05
       LI   R1,>0400
       BLWP @VSBW              PUT LENGTH OF 4 IN PAB
       LI   R0,>1E00
       LI   R1,E2              PUT CODE FOR CARRIAGE RTN &
       LI   R2,4               8/72" VERTICAL LINE SPACING
       BLWP @VMBW              IN DATA BUFFER.
       MOV  R6,@>8356          POINT TO DEVICE NAME LENGTH
       BLWP @DSRLNK            DSRLNK-CHANGE VERT SPACING
       DATA 8
L0     MOV  R9,R0
       BLWP @VSBR              PUT BYTE OF SCREEN RAM IN R1
       SRL  R1,8               SHIFT TO LSB OF R1
       AI   R1,-128            ADJUST FOR BASIC
       SLA  R1,3               *8
       AI   R1,1024            PTRN ADDR=1024+(CHAR#-32)*8
       MOV  R1,R0
       LI   R1,IN
       LI   R2,8
       BLWP @VMBR              PUT PATTERN INTO IN
       LI   R5,128             R5 = BIT#
L3     LI   R6,128             R6 = BYTE#
       CLR  R3                 R3 = OFFSET FOR IN
       CLR  R4                 R4 FOR BUILDING NEXT CHAR
L2     CLR  R7
       MOVB @IN(3),R7          R7 HOLDS BYTE BEING DECODED
       SWPB R7                 PUT BYTE IN LSB OF R7
       C    R7,R5              IS BIT ON?
       JLT  L1                 NO
       A    R6,R4              YES, TURN OUTPUT BIT ON
       S    R5,R7              TURN OFF INPUT BIT
       SWPB R7                 PUT BYTE IN MSB OF R7
       MOVB R7,@IN(3)          REWRITE TO IN
L1     INC  R3                 POINT TO NEXT BYTE
```

```
        SRA  R6,1             /2
        JGT  L2               DO NEXT BYTE IF MORE
        SWPB R4               PUT OUTPUT BYTE IN MSB OF R4
        MOVB R4,@DO(8)
        INC  R8
        MOVB R4,@DO(8)        STORE AT DO
        INC  R8               POINT TO NEXT BYTE OF DO
        SRA  R5,1             /2
        JGT  L3               CONSTRUCT NEXT OUTPUT BYTE
        INC  R9               NEXT SCREEN POS
        CZC  @MK,R9           EOL?
        JNE  L0               NO, NEXT POSITION
        LI   R3,4             COUNTER
        LI   R0,>1D05         ONLY ESC K WRITE
        LI   R1,>0400
        BLWP @VSBW            PUT LENGTH OF 4 IN PAB
        LI   R0,>1E00         VDP BUFFER
        LI   R1,E1
        LI   R2,4
        BLWP @VMBW            PUT ESC K SEQ IN DATA BUFF
        LI   R6,>1D09
        MOV  R6,@>8356        POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK          DSR TO WRITE ESC K SEQUENCE
        DATA 8
        LI   R4,DO            START OF CPU GRAPHICS BUFFER
L5      LI   R2,128           QUARTER OF GRAPHICS STRING
        MOV  R4,R1
        LI   R0,>1E00         VDP ADDR
        BLWP @VMBW            PUT DO IN DATA BUFFER
        LI   R0,>1D05
        LI   R1,>8000         128 BYTES
        BLWP @VSBW
        MOV  R6,@>8356        POINT TO DEVICE NAME LENGTH
        BLWP @DSRLNK          DSR TO OUTPUT 8 CHARS
        DATA 8
        AI   R4,128           POINT TO START OF NEXT QUARTER
        DEC  R3
        JNE  L5               DO IT AGAIN FOR LAST HALF
        LI   R0,>1D05         OUTPUT CR/LF
        LI   R1,>0200
        BLWP @VSBW            PUT LENGTH OF 2 IN PAB
        LI   R0,>1E00
        LI   R1,CR
        LI   R2,2
        BLWP @VMBW
        MOV  R6,@>8356
        BLWP @DSRLNK          DSRLNK TO OUTPUT CR/LF
        DATA 8

L4      LI   R0,>1D00
        LI   R1,>0100
```

```
       BLWP @VSBW             PUT CLOSE IN PAB
       MOV  R6,@>8356
       BLWP @DSRLNK
       DATA 8
       LI   R0,4              DELAY
DEL1   LI   R1,20000
DEL2   NOP
       DEC  R1
       JNE  DEL2
       DEC  R0
       JNE  DEL1
       MOVB @S1,@>9C02
       SWPB @S1
       MOVB @S1,@>9C02
       CLR  R0
       MOVB R0,@>837C
       B    @NEXTL

CLOSE  LWPI >83E0
       MOV  @SAVRTN,R11
       RT

SCLEN  EQU  >8354
SCNAME EQU  >8356
CRULST EQU  >83D0
SADDR  EQU  >83D2
GPLWS  EQU  >83E0
FLGPTR DATA 0
SVGPRT DATA 0
SAVCRU DATA 0
SAVENT DATA 0
SAVLEN DATA 0
SAVPAB DATA 0
SAVVER DATA 0
NAMBUF DATA 0,0,0,0,0
DLNKWS DATA 0,0,0,0,0
TYPE   DATA 0,0,0,0,0,0,0,0,0,0,0
C100   DATA 100
H20    EQU  $
H2000  DATA >2000
DECMAL TEXT '.'
HAA    BYTE >AA
DSRLNK DATA DLNKWS,DLENTR

JMPLNK B    @LNKERR

DLENTR MOV  *R14+,R5
       SZCB @H20,R15
       MOV  @SCNAME,R0
       MOV  R0,R9
```

```
       AI   R9,-8
       BLWP @VSBR
       MOVB R1,R3
       SRL  R3,8
       SETO R4
       LI   R2,NAMBUF
LNK$LP INC  R0
       INC  R4
       C    R4,R3
       JEQ  LNK$LN
       BLWP @VSBR
       MOVB R1,*R2+
       CB   R1,@DECMAL
       JNE  LNK$LP
LNK$LN MOV  R4,R4
       JEQ  JMPLNK
       CI   R4,7
       JGT  JMPLNK
       CLR  @CRULST
       MOV  R4,@SCLEN
       MOV  R4,@SAVLEN
       INC  R4
       A    R4,@SCNAME
       MOV  @SCNAME,@SAVPAB
SROM   LWPI GPLWS
       CLR  R1
       LI   R12,>0F00
NOROM  MOV  R12,R12
       JEQ  NOOFF
       SBZ  0
NOOFF  AI   R12,>0100
       CLR  @CRULST
       CI   R12,>2000
       JEQ  NODSR
       MOV  R12,@CRULST
       SBO  0
       LI   R2,>4000
       CB   *R2,@HAA
       JNE  NOROM
       A    @TYPE,R2
       JMP  SGO2
SGO    MOV  @SADDR,R2
       SBO  0
SGO2   MOV  *R2,R2
       JEQ  NOROM
       MOV  R2,@SADDR
       INCT R2
       MOV  *R2+,R9
       MOVB @SCLEN+1,R5
       JEQ  NAME2
       CB   R5,*R2+
```

```
       JNE  SGO
       SRL  R5,8
       LI   R6,NAMBUF
NAME1  CB   *R6+,*R2+
       JNE  SGO
       DEC  R5
       JNE  NAME1
NAME2  INC  R1
       MOV  R1,@SAVVER
       MOV  R9,@SAVENT
       MOV  R12,@SAVCRU
       BL   *R9
       JMP  SGO
       SBZ  0
       LWPI DLNKWS
       MOV  R9,R0
       BLWP @VSBR
       SRL  R1,13
       JNE  IOERR
       RTWP
NODSR
       LWPI DLNKWS
LNKERR CLR  R1
IOERR  SWPB R1
       MOVB R1,*R13
       SOCB @H20,R15
       RTWP

VDPRD  EQU  >8800        VDP read data address
VDPWD  EQU  >8C00        VDP write data address
VDPWA  EQU  >8C02        VDP write address address

*** General utility workspace registers (Overlaps next WS)
UTILWS DATA 0,0
       BYTE 0
R2LB   BYTE 0
       BSS  28
*
* Utility Vectors
*
VSBW   DATA UTILWS,VSBWEN     VDP single byte write
VMBW   DATA UTILWS,VMBWEN     VDP multiple byte write
VSBR   DATA UTILWS,VSBREN     VDP single byte read
VMBR   DATA UTILWS,VMBREN     VDP multiple byte read
VWTR   DATA UTILWS,VWTREN     VDP write to register
*
*=============================================================
*      VDP UTILITIES
*
** VDP single byte write
```

```
*
VSBWEN BL   @WVDPWA          Write out address
       MOVB @2(R13),@VDPWD   Write data
       RTWP                  Return to calling program
*
** VDP multiple byte write
*
VMBWEN BL   @WVDPWA          Write out address
VWTMOR MOVB *R1+,@VDPWD      Write a byte
       DEC  R2               Decrement byte count
       JNE  VWTMOR           More to write?
       RTWP                  Return to calling Program
*
** VDP single byte read
*
VSBREN BL   @WVDPRA          Write out address
       MOVB @VDPRD,@2(R13)   Read data
       RTWP                  Return to calling program
*
** VDP multiple byte read
*
VMBREN BL   @WVDPRA          Write out address
VRDMOR MOVB @VDPRD,*R1+      Read a byte
       DEC  R2               Decrement byte count
       JNE  VRDMOR           More to read?
       RTWP                  Return to calling program
*
** VDP write to register
*
VWTREN MOV  *R13,R1          Get register number and value
       MOVB @1(R13),@VDPWA   Write out value
       ORI  R1,>8000         Set for register write
       MOVB R1,@VDPWA        Write out register number
       RTWP                  Return to calling program
*
** Set up to write to VDP
*
WVDPWA LI   R1,>4000
       JMP  WVDPAD
*
** Set up to read VDP
*
WVDPRA CLR  R1
*
** Write VDP address
*
WVDPAD MOV  *R13,R2          Get VDP address
       MOVB @R2LB,@VDPWA     Write low byte of address
       SOC  R1,R2            Properly adjust VDP write bit
       MOVB R2,@VDPWA        Write high byte of address
       MOV  @2(R13),R1       Get CPU RAM address
```

```
       MOV  @4(R13),R2       Get byte count
       RT                    Return to calling routine
*
       END
```

Disk 3. Contents of File LISTING

```
      Block 1

32000 OPEN #1:"PIO",OUTPUT,VARIABLE 255 :: PRINT #1:CHR$(27);"A";CHR$(8)
32001 Y1=0

      Block 2

32002 Y1=Y1+1 :: Q$="" :: T$="" :: S$=""
32003 FOR Z=3 TO 29 :: CALL GCHAR(Y1,Z,G):: Q$=Q$&CHR$(G):: NEXT Z
32004 FOR Z=1 TO LEN(Q$)
32005 G=ASC(SEG$(Q$,Z,1))
32006 CALL CHARPAT(G,A$)
32007 CALL DUMP(A$,S$)
32008 T$=T$&S$
32009 NEXT Z
32010 PRINT #1:CHR$(27);"K";CHR$(LEN(Q$)*8);CHR$(0);T$
32011 GOTO 32002

      Block 3

32012 SUB DUMP(C$,S$)
32013 IF C$=RPT$("0",16)THEN S$=RPT$(CHR$(0),8):: SUBEXIT
32014 S$=""
32015 B$="0000000100100011010001010110011111000100110101011110011011110111 1"
32016 H$="0123456789ABCDEF"

      Block 4

32017 FOR X=1 TO 8 :: FOR Y=1 TO 2
32018 P=POS(H$,SEG$(C$,(X-1)*2+Y,1),1)
32019 Z$=SEG$(B$,(P-1)*4+1,4)
32020 FOR Z=1 TO 4 :: G(X,(Y-1)*4+Z)=ASC(SEG$(Z$,Z,1))-48
32021 NEXT Z
32022 NEXT Y :: NEXT X

      Block 5

32023 FOR Y=1 TO 8 :: T=0 :: FOR X=1 TO 8
32024 T=G(X,Y)*2^(8-X)+T
32025 NEXT X :: S$=S$&CHR$(T):: NEXT Y
32026 SUBEND
```

Block 1: Initialization. Opens printer for output and sets up printer to use 8/72 of an inch line spacing.

Block 2: Screen scan. Line 32003 puts the horizontal screen line (Y1) into the string Q$.32004 to 32009 then convert this to output for the printer by getting the character codes for each character and passing them to the subprogram DUMP. Line 32010 outputs the line of graphics data to the printer with the ESCAPE K sequence followed by two bytes describing the length of the data.

Block 3: Initialization of the DUMP subprogram. C$ is the character pattern passed to DUMP and S$ is the printer data returned by DUMP. Line 32013 checks to see if the character is a space. If it is then DUMP gives it the printer code immediately and returns to save time. Line 32015 contains the Binary number system from 0 to 15 stored in a string. Line 32016 is the Hexadecimal system from 0 to 15.

Block 4: These lines unpack the character pattern (C$) into the array G(,) as a series of ones and zeros.

Block 5: These lines rotate the array formed in Block 4 into data acceptable by the printer and pack the data back into a string (S$) to be returned by DUMP.

To use this program, simply MERGE it into your program and add the statement "GOSUB 32000" when you want a screen dump. The problem is that the program will take about 20 minutes to run so be prepared to wait.

Disk 3. Contents of File READ-THIS

The Assembly language DUMP routine

By J. Peter Hoddie for the BCS TIUG

This routine is based on a routine by John Clulow from his New Horizons User Group.

This routine will produce a full graphics dump to an Epson printer in under one minute. The dump is done in double density graphics mode and so produces a very clear image. The routine is set to run in Extended BASIC but can be easily modified to run in other environments.

To load the program type "CALL INIT :: CALL LOAD("DSK1.DUMPJPH").The program is relocatable so if you have another assembly language program in memory you won't lose it if you load the DUMP routine.

The routine comes set up to dump to the "PIO" device but if you have your printer hooked up to a different port this device name may be modified as follows. Type "CALL LINK("INIT","RS232.BA=4800.CR")" or whatever your device name may be. However the device name must end in ".CR" or the DUMP routine will lock up.

To dump a screen you must execute the statement "CALL LINK("DUMP")" and when the routine is done executing it will return you to your program.

There is a second way to obtain a screen dump but it is rather messy in that after the screen dump is complete you must turn your computer off and on again to continue. It has one major advantage: It will dump the screen at the push of button. This is accomplished by using the user directed ISR. To initialized this mode type "CALL LINK("START")".To dump a screen now hold down **FCTN 5** and the screen will be dumped unfortunately once the routine is done your computer will lock. I'm still working on this problem but for now this is the best I can do.

Finally: If you go over the code to this program you will find that it is rather inefficient and could probably be speed up quite a bit more, I leave that for you to work on and would be interested to see the results. And as usual this DUMP routine doesn't do sprites . . . has anyone seen one that does . . . it is possible.

Disk 3. Contents of File SECTOR-ASM

Information on accessing the DISKETTE by SECTOR

By J. Peter Hoddie for the BCS TIUG

This information was supplied to me by Mark Hoogendoorn over the phone after he copied it off of CompuServe. It has worked for both of us after some effort. Good luck

When you wish to access a sector of the disk as opposed to files you must set up a PAB but it is different from the regular PAB. It is simply the word >0110 in VDP RAM with the word at >8356 in CPU RAM pointing to it.

At location >8350 you put one full word indicating the number of the sector you wish to read (from 0 to 359 for a single sided, single density disk). At >834C you put the drive number to read/write the sector from/to and at the location >834E you must put a pointer to a 256 byte buffer in VDP RAM for the sector to b read into or written from. Finally at >834D you must put a >01 if you wish to read the sector or a >00 if you wish to write to it. Once all of this is done you simple execute the following code:

```
BLWP @DSRLNK
DATA >000A
```

This will do it. I have no idea how or why, or what the data >000A instead of the usual >0008 does but it does seem to work. Have fun.

Disk 3. Contents of File TYPE

```
.TL 91:27,14; [ ENLARGED ON
.TL 93:27,18; ] enlarged off
.TL 123:27,15;  CONDENSED ON
.TL 125:27,18; } condensed off
.TL 60:27,77; < ELITE ON
.TL 62:27,80;  > elite off
.TL 126:27,69; ~ EMPHASIZED ON
.TL 124:27,70; | emphasized off
.TL 92:27;52;  \ ITALIC ON
.TL 96:27;53;  ` italic off
.TL 43:27;71;  + DOUBLE STRIKE ON
.TL 61:27;72;  = double strike off
```

Disk 3. Contents of File TYPE2

```
.PL 80
.TL 91:27,14                [ ENLARGED ON
.TL 93:27,18                ] enlarged off
.TL 123:27,15               { CONDENSED ON
.TL 125:27,18               } condensed off
.TL 60:27,77;               < ELITE ON
.TL 62:27,80;               > elite off
.TL 126:27,69               ~ EMPHASIZED ON
.TL 124:27,70               | emphasized off
.TL 92:27;52                \ ITALIC ON
.TL 96:27;53                ` italic off
.TL 43:27;71                + DOUBLE STRIKE ON
.TL 61:27;72                = double strike off

[Print Enlarged]

[{Print Enlarged CONDENSED}]

[<Print Enlarged ELITE>]

[~Print Enlarged EMPHASIZED|]

[\Print enlarged ITALIC`]

[+Print Enlarged DOUBLE STRIKE =]

{Print Condensed }

<{Print Condensed ELITE}>

~{PRINT Condensed EMPHASIZED|}

{\Print Condensed ITALIC`}

{+Print Condensed DOUBLE STRIKE=}

{[Print Condensed ENLARGED]}

<Print ELITE>

[<Print Elite ENLARGED]>

{<Print Elite CONDENSED}>

~<Print Elite EMPHASIZED|>

<\Print Elite ITALIC`>
```

```
<+Print Elite DOUBLE STRIKE=>
~Print Emphasized|
[~Print Emphasized ENLARGED]|
~{Print Emphasized CONDENSED|}
~<Print Emphasized ELITE>|
~\Print Emphasized ITALIC`|
~+Print Emphasized DOUBLE STRIKE=|
\Print Italic`
\[Print Italic ENLARGED]`
\{Print Italic CONDENSED}`
\<Print Italic ELITE>`
\~Print Italic EMPHASIZED|`
\+Print Italic DOUBLE STRIKE=`
```

Disk 4. Super-Bug Debugger

Version:
Requires: EA or XB
Author: Navarone
Language: AL
Updated: 08/31/86

Utility developed for TI by Navarone. Both compressed and uncompressed formats of this debugger are included. Complete documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-4
Sectors total           = 360
Sectors used            = 260
Sectors available       = 98
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >004 HELP         59 DIS/VAR 80   >06e 058
002 >003 ORIGHELP     59 DIS/VAR 80   >034 058
003 >002 SBUG         97 DIS/FIX 80   >022 018 >0a8 078
004 >006 SBUGC        45 DIS/FIX 80   >108 044
```

Disk 4. Contents of File HELP

SUPER BUGGER VERSION 3.1 19-FEB-1983

SUPER-BUGGER is a stand-alone program that may be loaded by the Editor/Assembler Load and Run option, or either TI-BASIC or EXT-BASIC CALL LOAD Options. No special hardware is required, but this program will operate only on the TI-99/4A with Memory Expansion, and a Disk Controller. The RS232 may optionally be used to get a hard copy printout on some operations.

SUPER-BUGGER is a very sophisticated and powerful debug tool which can provide functions usually only available on very expensive development systems requiring special hardware. SUPER-BUGGER allows you to actually step through your machine language program, executing each machine instruction one at a time. It enables you to examine the logic of your program as it is being run. As each instruction is executed, the SYMBOLIC interpretation is displayed on the screen in the same format as it occurs in your assembly source listing, providing a trace of instruction execution.

SUPER-BUGGER has a built in DIS-ASSEMBLER which you can use to decipher machine code to its symbolic assembly language representation. It will interpret any instruction and show all types of operand uses, even displaying the JMP address of jump instructions.

Operation of the SUPER-BUGGER is syntactically identical to the TI-DEBUGGER program, however there are features provided by the TI program that are not supported by SUPER-BUGGER due to memory size limitations. It is recommended you become familiar with the TI-DEBUGGER and its documentation before using SUPER-BUGGER.

The following is a summary comparison of the two debuggers. The TI-DEBUGGER and SUPER-BUGGER can be used to complement each other to provide the best development tool in the industry.

COMMAND	*TI-DEBUG*	*SUPER-BUGGER*
A	Load Memory with ASCII	DIS-ASSEMBLE machine code to Mnemonic
B	Breakpoint Set/Clear	Same except always two-word Breakpoints
C	CRU Inspect/Change	** Not Supported
D	** Not Supported	Dump memory to HARD COPY DEVICE
E	Execute	Same
F	Find Word or Byte	** Not Supported
G	GROM Base change	** Not Supported
H	Hex Arithmetic	** Not Supported
I	Inspect Screen Location	** Not Supported
K	Find Data Not Equal	** Not Supported
L	**Not Supported	Hard Copy List device Toggle on/off
M	Memory Inspect/Change	Same
N	Move Block	** Not Supported
P	Compare Memory Block	** Not Supported
Q	QUIT Debugger	Same
R	Inspect/Change WP,PC,& SR	Same
S	Step with special hardware	Single Step on any TI99/4A
T	Trade Screen	Trades user screen with SBUG screen
U	Toggle Basic offset on/off	Same
V	VDP Base change	Run till VALUE = entered number
W	Inspect/Change Register	Same
X,Y,or Z	Change BIAS	Same
>	Hex to Decimal convert	** Not Supported
.	Decimal to Hex convert	** Not Supported

SUPER-BUGGER OPERATING PROCEDURES

The syntax for SUPER-BUGGER is the same as the TI-DEBUGGER. The following conventions are used. Items surrounded by <angle brackets> represent mandatory data to be provided. Items surrounded by {braces} indicate you must choose between the two or more items included. Items surrounded by [brackets] indicate optional data. The ellipsis (. . .) indicates the previous item may be repeated.

The SUPER-BUGGER is located on the diskette in two versions. The version named "SBUG" is stored in DISPLAY/FIXED object code format. The Version named "SBUGC" is stored in condensed format which can be loaded by the Editor/Assembler loader. The condensed format cannot be loaded by TI-BASIC or EXTENDED BASIC loaders.

STEP 1. Load SUPER-BUGGER

The "SBUG[C]" program is relocatable and can be loaded the same way the TI "DEBUG" program is loaded, i.e.; the LOAD AND RUN Option on the Editor/Assembler or with CALL LOAD From TI-BASIC and EXT. BASIC. The name of the file to Load is "DSK1.SBUG", (if the diskette is in disk drive 1). Enter the SUPER-BUGGER exactly as you would enter TI-DEBUGGER.

If entered by the Editor/Assembler Load and Run option, then Select LOAD AND RUN from the menu and at the prompt:

```
FILE NAME?
```

Enter the file name of your DEVELOPMENT object code program in the format (DSKn.Filename).

After the file is loaded, the filename is erased from the screen and you may load other program modules. The SUPER-BUGGER module should be the last program module loaded. Enter the following:

```
DSK1.SBUG    or whatever disk drive has SUPER-BUG
```

After SBUG is loaded, you may proceed by pressing **ENTER** without entering a filename.

The prompt:

```
PROGRAM NAME?
```

appears next. you must enter the SUPER-BUGGER Program first in order to gain control over your development program. So, enter:

```
SBUG
```

When initially entered, the ID is displayed

```
*** SUPER-BUG  VER 3.1 ***
```

STEP 2. Select User screen type.

The message will ask you to select the type of USER screen you are using.

```
*  BIT MAP SCREEN (Y OR N)
```

Press **N** if using the TI graphic screens. When using this option SBUG will share the graphic screen with the user's program screen.

Press **Y** to set up screen address which allows the user screen to be automatically restored to BIT MAP Mode upon exit from SBUG into the user program. This is extremely helpful when developing software that uses different screen modes than the standard TI system screens. You may use the **T** option to flip to the user screen when in SBUG.

IMPORTANT: YOU CANNOT USE BITMAP SCREEN IF RUNNING UNDER BASIC.

VDP Screen Memory Map for BIT MAP MODE under SBUG

SUPER-BUG	*USER*	*Contents*
>3800	>0000	PATTERN DESCRIPTOR TABLE
>3C00	>1C00	SCREEN IMAGE TABLE
>3B00	>1F00	SPRITE ATTRIBUTE TABLE
>Not used	>1800	SPRITE DESCRIPTION TABLE
>3B80	>2000	COLOR TABLE
>3F00	????	PABs
>3F20	????	PAB BUFFER TO >3FFF

The eight bytes of VDP WRITE ONLY Register values are located at >0132 (relative to loading address of SBUG).

Use the M option to alter these locations and provide your own VDP screen locations if you want to change screen addresses in your program. (The entry point of SBUG is located at >0000 relative to load address.)

STEP 3. Enter the list device.

The message will instruct you to enter the list device.

```
ENTER LIST DEVICE
```

Press **ENTER** to use the default LIST device of:

```
RS232.BA=4800
```

Or enter any output device such as a DSKn.Filename or an RS232 device. (When using BIT MAP Screen you cannot use a DISK device for output as the VDP addresses used for SBUGGER will be used by the disk routines).

This device is then used for output on the D(ump) and A(ssemble) commands only. With the use of the L(ist) command you can select to print a hard copy of memory dumps and dis-assembly listings. If no hard copy is desired, you may skip this prompt by pressing return without entering a device name, but you must also turn the List device OFF By using the L(ist) command.

If BIT MAP screen is not selected, then SUPER-BUGGER will not save the screen upon entry. Therefore, the user screen is used by the SBUG program to display its messages. However, the screen offset will be automatically set for BASIC when "SBUG" is loaded by the BASIC loaders.

STEP 4. Find the entry point of your program.

To find the entry point of your program in memory, you can inspect the REF/DEF table beginning at >3FB0. Each 8-Byte table entry has the name of your program and its DEF entry address. The first 6 bytes of each table entry contain the REF/DEF Name, and the WORD following is the entry point for that name.

Use the <M> option to scan through the REF/DEF table and locate the address where your program is entered.

You must also determine where your program has been loaded. If you loaded your program with the LOAD and RUN Option from the Editor/Assembler, and it was the first program, it will be placed at address >A000. Therefore, it will be handy to enter an offset or relocation BIAS into one of the X, Y, or Z BIAS locations.

Before you can actually ENTER your development program, you must first set up the WP, PC and SR must be set for your program. Locate these values from your listings, then add your BIAS to these addresses and enter with the <R> option.

At this point you may enter your program by the <S> option or the <E> option to single step through your program.

IMPORTANT

Prior to using the <E> Option you must place a Breakpoint in your program at a place where you expect to stop. otherwise, there is no way to re-enter the SBUG program.

Breakpoints are not necessary when using the <S>ingle step or <V>alue option to step through your program.

COMMANDS

A — DIS-ASSEMBLE MACHINE CODE

* To dis-assemble code at current User PC Location

ENTER:

```
 A<return>
[<number>]{<return> or <space>}
```

Dis-assembles the machine instructions beginning at the user PC Address. If a single parameter is entered it is treated as the number of instructions to display or print.

* To Dis-assemble code between locations specified

ENTER:

```
A[<Start Address>{<space> or <,>}<stop address>]
```

Dis-assembles the machine instructions beginning with the start address entered and continuing to the stop address entered.

Note: Output from this command will be directed the HARD COPY DEVICE if the L(ist) has been turned ON. Otherwise the listing will be displayed on the user's screen.

D — DUMP MEMORY

* To Dump code at current User PC Location

ENTER:

```
D<return>
D[<number>]{<return> or <space>}
```

Dumps the memory Data in HEX and ASCII beginning at the user PC Address. If a single parameter is entered it is treated as the number of locations to display or print.

* To Dump code between locations specified

ENTER:

```
D[<Start Address>{<space> or <,>}<stop address>]
```

Dumps the memory Data in HEX and ASCII beginning with the start address entered and continuing to the stop address entered.

Note: Output from this command will be directed to the HARD COPY DEVICE if the L(ist) has been turned ON. Otherwise the listing will be displayed on the user's screen.

L — LIST DEVICE TOGGLE

* To Toggle the List device between the Hard copy device entered, and the user screen display.

ENTER:

```
L
```

The current disposition is displayed on the screen. To change conditions, simply re-enter this command.

S — SINGLE STEP

* To Step through machine instructions one at a time.

ENTER:

```
S
```

The instruction located at the user's PC is executed and displayed on the screen next to its memory address. The effective jump address is also shown for all JUMP instructions.

Note: Care must be taken when stepping through VDP RAM accessing. You should avoid this by placing a breakpoint following the VDP access instructions. You should also avoid stepping through GROM code.

T — TRADE SCREEN (BIT MAP MODE ONLY)

* To toggle or Trade the user screen for the SUPER-BUGGER Screen.

ENTER:

```
T
```

The SBUG screen will be swapped with the user screen ONLY if BIT MAP mode was selected upon entry to SBUG. Each time the **T** is pressed, the screen will toggle from user screen to SBUG, and reverse.

V — VALUE EQUAL

* To execute in slow speed until the value at address entered is equal to the value entered

ENTER:

```
V<Address><space> or <,><Value>
```

The User program will be executed in interpretive mode (slow-speed) until the value contained at the address specified is equal to the value entered on the command line. When the value is equal, the SBUG screen is restored and the program will be halted with the message "NO BREAKPOINTS SET".

IMPORTANT: DO NOT EXECUTE TI OPERATING SYSTEM CODE UNDER THIS OPTION.

If User program makes any branches into SYSTEM utilities such as SCAN or GPLLNK routines, then results are unpredictable and your computer may go into a catatonic state requiring you to power OFF then back on to recover.

This option is useful only for executing code which will not leave your program area.

THIS PRODUCT WAS DEVELOPED BY NAVARONE INDUSTRIES. IF YOU ARE INTERESTED IN ANY OF THEIR OTHER HARDWARE OR SOFTWARE DEVELOPMENT TOOLS, WRITE TO:

NAVARONE INDUSTRIES, INC.
510 LAWRENCE EXPRESSWAY, #800
SUNNYVALE, CA 94086

OR CALL (408) 866-8579.

[The file HELP was probably the result of minor editing changes by J.P. Hoddie of the BCS, and is almost identical to the HELP file written by Navarone. The original help file is called ORIGHELP and is on the disk. The contents of ORIGHELP are not reproduced here. —Ed.]

Disk 5. TI-Writer and Multiplan Updates

Version: *Author*: TI, Tom Knight
Requires: TIW, EA, Multiplan *Language*: *Updated*:

Update files for TI-Writer and Multiplan. Also TO-Writer which allows use of TI-Writer without the cartridge.

```
dskdir. v2.0. 12-dec-96
Disk name               = TIMP&TIWRT
Sectors total           = 360
Sectors used            = 316
Sectors available       = 42
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHARA1        9 PROGRAM      >022 008
002 >003 EDITA1       33 PROGRAM      >02a 032
003 >004 EDITA2        6 PROGRAM      >04a 005
004 >005 FORMA1       33 PROGRAM      >04f 032
005 >006 FORMA2       15 PROGRAM      >06f 014
006 >007 FORMA4800A   33 PROGRAM      >07d 032
007 >008 FORMA4800B   15 PROGRAM      >09d 014
008 >009 MPBASE       41 PROGRAM      >0ab 040
009 >00a MPCHAR        9 PROGRAM      >0d3 008
010 >00b MPDATA       33 PROGRAM      >0db 032
011 >00c MPINTR       33 PROGRAM      >0fb 032
012 >00d OVERLAY      31 INT/FIX128   >11b 030
013 >00e TKWRITER      3 PROGRAM      >139 002
014 >00f WRITER1      22 DIS/FIX 80   >13b 021
```

Disk 6. Miscellaneous Routines

Version: *Author*: Lawless, TI, others
Requires: XB, EA *Language*: *Updated*:

Includes Masscopy — a 3-pass disk copier, a disk sector editor from TI, an interrupt driven clock for XB with source code, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-6
Sectors total           = 360
Sectors used            = 133
Sectors available       = 225
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 D-PATCH      49 DIS/FIX 80 Y >022 048
002 >003 DUMP         11 DIS/FIX 80   >052 010
003 >004 MASS-COPY    17 DIS/FIX 80   >05c 016
004 >005 TI-DIS-ASM   14 DIS/FIX 80   >06c 013
005 >006 TIME          7 DIS/FIX 80   >079 006
006 >007 TIME/S       35 DIS/VAR 80   >07f 034
```

Disk 6. Contents of File TIME/S

```
*****************************************************************
* A RELOCATABLE ASSEMBLY LANGUAGE INTERRUPT DRIVEN CLOCK/TIMER   *
* DISASSEMBLED FROM D. L. FITCHHORN'S (305 NAVAJO, KELLER, TX    *
* 76248) AORG PROGRAM WITH MODIFICATIONS BY JAMES S. MCCULLOCH,  *
* 9505 DRAKE AVE., EVANSTON, IL 60203 VERSION 28 AUG 1984        *
*                                                               *
* AFTER CALL LOAD(ING THE OBJECT FILE GENERATED FROM ASSEMBLING  *
* THIS SOURCE CODE, YOU CAN USE IT AS AN ELAPSED TIME COUNTER    *
* BY SIMPLY DOING A CALL LINK("STARTT").  IF YOU WANT TO USE IT  *
* AS A CLOCK, BEFORE DOING THE ABOVE CALL LINK, YOU SHOULD FIRST *
* LOAD SOME DATA AT >2038, PERHAPS BY USING THE FOLLOWING:       *
*      CALL LOAD(8248,A,B,C,D,E,F,G,H,I,J)  WHERE-               *
* A=250 (ALWAYS)   these load the word at >2038->2039 with the   *
* B=206 (ALWAYS)   value >FACE, which is used to flag the fact   *
*                  that there is user inserted data to read.     *
* C=0   (ALWAYS)   these load the next word with the type of     *
* D=13 OR 24       clock you want-either 12 or 24 hour time      *
*                  respectively (yes, that's a 13)               *
* E=0   (ALWAYS)                                                *
* F=NUMBER REPRESENTING THE HOUR SET (WHAT HOUR DO YOU WANT TO   *
*   START COUNTING FROM? (from 1 to 12 {or 23})                 *
* G=0   (ALWAYS)                                                *
* H='MINUTES' TO BE SET  (from 0 to 59)                         *
* I=0   (ALWAYS)                                                *
* J='SECONDS' TO BE SET  (from 0 to 59)                         *
*                                                               *
* IF YOU GET TIRED OF WATCHING IT RELENTLESSLY COUNT, YOU CAN    *
* GET RID OF IT WITH   CALL LINK("STOP")                        *
*****************************************************************
       DEF  STARTT,STOP  DEFINES LABELS TO BE PLACED IN REF/DEF TABLE
STATUS EQU  >837C        GPL STATUS REGISTER
VDPWA  EQU  >8C02        VDP WRITE ADDRESS REGISTER-SELECTS PLACE TO
*                        WRITE DATA ON SCREEN. FIRST 2 BITS MUST BE 01.
VDPWD  EQU  >8C00        VDP WRITE DATA REGISTER-DATA PLACED HERE WILL
*                        BE WRITTEN TO SUCCESSIVE BYTES BEGINNING AT
*                        ADDRESS IN VDPWA.
LC     DATA >003C        LOOP COUNTER=60
SL     DATA >0017        SCREEN LOCATION=23
UL     DATA >000D        UPPER LIMITS PERMITTED FOR HOURS
MSB01  DATA >4000        WILL BE USED TO SET FIRST 2 BITS TO 01
* ARRAY USED TO GENERATE DIGITS *
ARR    DATA >0001,>0203,>0405,>0607
       DATA >0809,>1011,>1213,>1415
       DATA >1617,>1819,>2021,>2223
       DATA >2425,>2627,>2829,>3031
       DATA >3233,>3435,>3637,>3839
       DATA >4041,>4243,>4445,>4647
       DATA >4849,>5051,>5253,>5455
```

```
       DATA >5657,>5859
SAVRTN DATA 0            AREA TO SAVE RETURN ADDRESS IN
WS     BSS  32           PROGRAM WORKSPACE
*********************************************
* WORKSPACE REGISTERS USED IN THIS PROGRAM    *
* R0=SCREEN LOCATION POINTER                  *
* R1=GENERAL SCRATCHPAD FOR DATA MANIPULATION *
* R2=VDP INTERRUPT COUNTER-DECREMENTED 60X/SEC*
* R3=UPPER LIMIT ALLOWED FOR HOURS            *
* R4=HOURS COUNTER                            *
* R5=MINUTES COUNTER                          *
* R6=SECONDS COUNTER                          *
*********************************************
STARTT MOV  R11,@SAVRTN  SAVE GPL RETURN ADDRESS
       LWPI WS           SWITCH TO PROGRAMS WORKSPACE
       MOV  @LC,R2       LOAD LOOP COUNTER
       MOV  @SL,R0       LOAD SCREEN LOCATION POINTER
       MOV  @>2038,R1    SEE WHAT'S AT >2038
       CI   R1,>FACE     IS IT=>FACE?
       JEQ  LOAD         IF IT IS, THERE IS DATA TO LOAD INTO PROGRAM
       MOV  @UL,R3       IF NOT, LOAD DEFAULT VALUES-UPPER HOURS LIMITS
       CLR  R4           ZERO HOURS COUNTER
       CLR  R5           ZERO MINUTES
       CLR  R6           ZERO SECONDS
       JMP  GOON         'GO TO' REST OF PROGRAM SEGMENT
LOAD   MOV  @>203A,R3    LOAD UPPER HOURS LIMITS-SHOULD BE EITHER 13 OR 24
       MOV  @>203C,R4    LOAD HOUR
       MOV  @>203E,R5    LOAD MINUTES
       MOV  @>2040,R6    LOAD SECONDS
GOON   LI   R1,LOOP      TAKE ADDRESS OF MAIN PROGRAM SEGMENT AND
       MOV  R1,@>83C4    LOAD IT INTO USER DEFINABLE INTERRUPT SERVICE
*                        ROUTINE. ON EACH VDP INTERRUPT (60/SEC), WILL
*                        BRANCH TO ROUTINE WHOSE ADDRESS IS HERE
       CLR  @STATUS      GET READY TO
       LWPI >83E0        RETURN TO GPL TERRITORY
       MOV  @SAVRTN,R11  AND
       B    *R11         GO ON
STOP   CLR @>83C4        CLEAR THE ISR HOOK (NO ROUTINE TO BRANCH TO)
       B    *R11         RETURN
LOOP   MOV  R11,@SAVRTN  SAVE GPL RETURN ADDRESS
       LIMI >0000        DISABLE VDP INTERRUPTS-DON'T BOTHER ME, I'M COUNTING
       LWPI WS           SWITCH TO THIS PROGRAM'S WORKSPACE
       DEC  R2           HAS ONE SECOND (60 VDP INTERRUPTS) GONE BY?
       JNE  EXIT         IF NOT, GO BACK TO GPL
       MOV  @LC,R2       OTHERWISE GO ON, RELOADING LOOP COUNTER
       INC  R6           ONE MORE SECOND
       CI   R6,>003C     HAVE 60 SECONDS GONE BY?
       JNE  WRITE        IF NOT, WRITE ON SCREEN
       CLR  R6           OTHERWISE GO ON, ZEROING SECONDS
       INC  R5           AND ADDING ONE MORE MINUTE.
       CI   R5,>003C     HAVE 60 MINUTES GONE BY?
```

```
       JNE  WRITE         IF NOT, WRITE ON SCREEN
       CLR  R5            OTHERWISE GO ON, ZEROING MINUTES
       INC  R4            AND ADDING ONE MORE HOUR.
       C    R4,R3         IS HOURS COUNTER AT UPPER PERMISSIBLE LIMIT?
       JNE  WRITE         IF NOT, WRITE TO SCREEN
       CLR  R4            IF IT IS, GO ON, ZEROING HOURS
       CI   R3,>0018      IS UPPER HOURS LIMIT SET AT 24?
       JEQ  WRITE         IF SO, WRITE TO SCREEN
       INC  R4            OTHERWISE, WRITE TO SCREEN, BUT MAKE HRS=1
*                         IF NOT 24, THEN UPPER LIMITS=13. THE NEXT HOUR
*                         AFTER 12 ON A CLOCK=1
WRITE  SWPB R0            GET LSB OF R0
       MOVB R0,@VDPWA     AND LOAD IT INTO VDPWA
       SWPB R0            RESTORE R0
       SOC  @MSB01,R0     SET MOST SIGNIFICANT BITS OF R0 TO 01-TELLS VDPWA
*                         TO GET READY TO WRITE DATA.
       MOVB R0,@VDPWA     LOAD THE PREPARED MSB TO VDPWA
       SZC  @MSB01,R0     RESTORE R0
       MOVB @ARR(4),R1    GET HOURS DIGITS FROM ARRAY
       BL   @DIGITS       PRINT HOURS
       BL   @COLON        PRINT COLON
       MOVB @ARR(5),R1    GET MINUTES DIGITS FROM ARRAY
       BL   @DIGITS       PRINT MINUTES
       BL   @COLON        PRINT ANOTHER COLON
       MOVB @ARR(6),R1    GET SECONDS DIGITS FROM ARRAY
       BL   @DIGITS       PRINT SECONDS
       SETO @>83D6       DON'T LET THE SCREEN TIME OUT COUNTER BLANK THE SCREEN
EXIT   LWPI >83E0         RETURN TO GPL TERRITORY
       MOV  @SAVRTN,R11   AND
       B    *R11          GO ON.
* SUBROUTINES *
DIGITS SRL  R1,4          MOVE R1 OVER I NYBBLE
       AI   R1,>9000      ADD APPROPRIATE MASK
       MOVB R1,@VDPWD     WRITE OUT FIRST DIGIT TO SCREEN
       SLA  R1,4          RESTORE R1
       ANDI R1,>0F00      NOW WE ONLY WANT THE SECOND DIGIT
       AI   R1,>9000      ADD MASK VALUE
       MOVB R1,@VDPWD     AND PRINT IT TO SCREEN
       B    *R11          RETURN TO CALLING PROGRAM
COLON  LI   R1,>9A00      MASKED COLON
       MOVB R1,@VDPWD     PRINT IT
       B    *R11          AND RETURN TO CALLING PROGRAM
       END
```

Disk 7. Terminal Emulator Programs

Version:
Author: TI, others
Requires: EA
Language: AL
Updated:

Collection of various terminal emulators. Some support up to a 24K incoming text buffer and/or 1200 baud operation. Source code to Terminal Emulator 99.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-7
Sectors total           = 360
Sectors used            = 230
Sectors available       = 128
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >003 T99          73 DIS/VAR 80   >036 072
002 >004 TE12/DOC     20 DIS/VAR 80   >07e 019
003 >005 TE1200       82 DIS/FIX 80   >091 081
004 >002 TERMEX       32 DIS/FIX 80   >022 020 >0e2 011
005 >007 TERMEX/DOC   23 DIS/VAR 80   >0f2 022
```

Disk 7. Contents of File T99

```
*
*    TERMINAL/99 - (C) 1984 Randy Holcomb
*    For non-commercial use only
*    Port 1 - 300 bps, 1 stop bit, even parity
*    Port 2 - 9600 bps, 1 stop bit, odd parity (Printer)
*
*    See the September and October 1984 Issues of Computer Shopper
*    For details about this program and the TMS 9902 ACC.
*
       UNL
       EVEN
       DEF  SFIRST,SLOAD,SLAST
SLOAD  EQU  $
WS     BSS  32           ;WORKSPACE
WS1    BSS  32           ;REGISTER
WS2    BSS  32           ;STORAGE
********************************
*      DATACOMM CONTROL CODES    *
********************************
NUL    BYTE >00
SOH    BYTE >01
STX    BYTE >02
ETX    BYTE >03
EOT    BYTE >04
ENQ    BYTE >05
ACK    BYTE >06
BEL    BYTE >07
BS     BYTE >08
HT     BYTE >09
LF     BYTE >0A
VT     BYTE >0B
FF     BYTE >0C
CR     BYTE >0D
SO     BYTE >0E
SI     BYTE >0F
DLE    BYTE >10
DC1    BYTE >11
DC2    BYTE >12
DC3    BYTE >13
DC4    BYTE >14
NAK    BYTE >15
SYN    BYTE >16
ETB    BYTE >17
CAN    BYTE >18
EM     BYTE >19
SUB    BYTE >1A
ESC    BYTE >1B
FS     BYTE >1C
```

```
GS     BYTE >1D
RS     BYTE >1E
US     BYTE >1F
BYTE80 BYTE >80
       LIST
*******************************
*      SCREEN WORK AREA       *
*******************************
VDPTMP BSS  920          ;STORAGE FOR 23 LINES
SPACES TEXT '                                        '
POSN   DATA >0           ;CURRENT SCREEN POSITION
XPOS   DATA >0           ;POINTER HOLD POS X
YPOS   DATA >0           ;POINTER HOLD POS Y
ALTFLG DATA >0           ;USE ALTERNATE CHARACTER SET
NOKEY  BYTE >FF
SPACE  EQU  >2020
PRNFLG DATA >0
QUIT   BYTE >05
CTLCOD BYTE >80
SCRLEN EQU  23
CHRSLN DATA 40
SCRBUF BSS  920
VDPR1C EQU  >83D4
KBDSEL EQU  >8374
KBD    EQU  >8375
KBDCHR BYTE >0
PREVCH BYTE >0
HAVKEY DATA >2000
       UNL
*****************************************************************
*      TERMINAL/99 RESIDENT CHARACTER SET (WITH DESCENDERS)       *
*****************************************************************
CHRSET DATA >0000,>0000,>0000,>0000          ; (SPACE)
       DATA >1010,>1010,>1000,>1000          ; !
       DATA >2828,>2800,>0000,>0000          ; "
       DATA >2828,>7C28,>7C28,>2800          ; #
       DATA >103C,>5038,>1478,>1000          ; $
       DATA >6464,>0810,>204E,>4E00          ; %
       DATA >0020,>5020,>5448,>3400          ; &
       DATA >0004,>0810,>0000,>0000          ; '
       DATA >0810,>2020,>2010,>0800          ; (
       DATA >1008,>0404,>0408,>1000          ; )
       DATA >0010,>5438,>5410,>0000          ; *
       DATA >0010,>107C,>1010,>0000          ; +
       DATA >0000,>0000,>0010,>1020          ; ,
       DATA >0000,>007C,>0000,>0000          ; -
       DATA >0000,>0000,>0010,>1000          ; .
       DATA >0404,>0810,>2040,>4000          ; /
       DATA >0038,>4C54,>6444,>3800          ; 0
       DATA >0010,>3010,>1010,>3800          ; 1
       DATA >0038,>440C,>3040,>7C00          ; 2
```

```
DATA >0038,>0418,>0444,>3800          ; 3
DATA >0018,>2848,>7C08,>0800          ; 4
DATA >007C,>4078,>0444,>3800          ; 5
DATA >001C,>2078,>4444,>3800          ; 6
DATA >007C,>4408,>1020,>2000          ; 7
DATA >0038,>4438,>4444,>3800          ; 8
DATA >0038,>4444,>3808,>7000          ; 9
DATA >0000,>1000,>0010,>0000          ; :
DATA >0000,>1000,>1010,>2000          ; ; (SEMICOLON)
DATA >0810,>2040,>2010,>0800          ; <
DATA >0000,>7C00,>7C00,>0000          ; =
DATA >2010,>0804,>0810,>2000          ; >
DATA >3844,>0408,>1000,>1000          ; ?
DATA >0038,>4454,>5C40,>3C00          ; @
DATA >0038,>447C,>4444,>4400          ; A
DATA >0078,>4478,>4444,>7800          ; B
DATA >0038,>4440,>4044,>3800          ; C
DATA >0070,>4844,>4448,>7000          ; D
DATA >007C,>4078,>4040,>7C00          ; E
DATA >007C,>4078,>4040,>4000          ; F
DATA >0038,>4440,>4C44,>3C00          ; G
DATA >0044,>447C,>4444,>4400          ; H
DATA >0038,>1010,>1010,>3800          ; I
DATA >001C,>0808,>0848,>3000          ; J
DATA >0048,>5060,>5048,>4400          ; K
DATA >0040,>4040,>4040,>7C00          ; L
DATA >0044,>6C54,>5444,>4400          ; M
DATA >0044,>6464,>544C,>4400          ; N
DATA >0038,>4444,>4444,>3800          ; O
DATA >0078,>4444,>7840,>4000          ; P
DATA >0038,>4444,>5448,>3400          ; Q
DATA >0078,>4444,>7848,>4400          ; R
DATA >003C,>4038,>0404,>7800          ; S
DATA >007C,>1010,>1010,>1000          ; T
DATA >0044,>4444,>4444,>3800          ; U
DATA >0044,>4428,>2810,>1000          ; V
DATA >0044,>4454,>5454,>2800          ; W
DATA >0044,>2810,>2844,>4400          ; X
DATA >0044,>2810,>1010,>1000          ; Y
DATA >007C,>0408,>1020,>7C00          ; Z
DATA >3820,>2020,>2020,>3800          ; [
DATA >4040,>2010,>0804,>0400          ; \
DATA >3808,>0808,>0808,>3800          ; ]
DATA >1028,>4410,>1010,>1000          ; ^
DATA >0000,>0000,>0000,>7C00          ; _
DATA >0020,>1008,>0000,>0000          ; `
DATA >0000,>3008,>3848,>3400          ; a
DATA >0040,>4078,>4444,>7800          ; b
DATA >0000,>3844,>4044,>3800          ; c
DATA >0004,>043C,>4444,>3800          ; d
```

```
       DATA >0000,>3844,>7C40,>3C00          ; e
       DATA >0008,>1010,>3810,>1000          ; f
       DATA >0000,>3844,>443C,>0438          ; g
       DATA >0040,>4058,>6444,>4400          ; h
       DATA >0010,>0030,>1010,>3800          ; i
       DATA >0008,>0018,>0808,>4830          ; j
       DATA >0040,>4050,>6050,>4800          ; k
       DATA >0010,>1010,>1010,>1800          ; l
       DATA >0000,>2854,>5454,>4400          ; m
       DATA >0000,>5864,>4444,>4400          ; n
       DATA >0000,>3844,>4444,>3800          ; o
       DATA >0000,>7844,>4478,>4040          ; p
       DATA >0000,>3844,>443C,>0404          ; q
       DATA >0000,>5864,>4040,>4000          ; r
       DATA >0000,>3840,>3804,>7800          ; s
       DATA >0010,>3810,>1010,>0800          ; t
       DATA >0000,>4848,>4848,>3400          ; u
       DATA >0000,>4444,>4428,>1000          ; v
       DATA >0000,>4454,>5454,>2800          ; w
       DATA >0000,>4428,>1028,>4400          ; x
       DATA >0000,>4444,>241C,>0438          ; y
       DATA >0000,>7C08,>1020,>7C00          ; z
       DATA >0810,>1020,>1010,>0800          ; {
       DATA >1010,>1000,>1010,>1000          ; |
       DATA >2010,>1008,>1010,>2000          ; }
       DATA >0000,>2054,>0800,>0000          ; ~
       DATA >003C,>3C3C,>3C3C,>3C00          ; (RUBOUT)
       LIST
************************************
*      PABs FOR DISK FILES & PIO    *
************************************
PAB1   EQU  $
OPCODE BYTE >0
DSRST  BYTE >0
BUFADR DATA >0
RECLEN BYTE >0
CHRCNT BYTE >0
RECNBR DATA >0
FLR001 BYTE >0
NAMLEN BYTE 24
FILNAM TEXT 'RS232/2.BA=9600.CR.LF.EC'
DSR1   DATA >0           ;ADDRESS OF 1ST PAB IN VDP RAM
DSR2   DATA >0           ;AND THE SECOND ONE....
DSR3   DATA >0           ;ONE MORE (WHY NOT?)
PRNSIZ BYTE >01
PRNBAD DATA >1900
WRITE  BYTE >03
****************************************
*      SYSTEM EQUATES AND ADDRESSES     *
****************************************
       REF  DSRLNK,XMLLNK,VMBW,VMBR,VSBW,KSCAN,GPLLNK,VWTR
```

```
       REF  VSBR
       DEF  T99
GPLWS  EQU  >83E0
STATUS EQU  >837C
KBDATA EQU  >8375
GPLSTS EQU  >83D4        ;GPL STATUS REGISTER
FAC    EQU  >834A        ;FLOATING POINT ACCUMULATOR
VDPWA  EQU  >8C02        ;VDP WRITE ADDRESS
VDPRD  EQU  >8800        ;VDP READ DATA
VDPWD  EQU  >8C00        ;VDP WRITE DATA
VDPSTA EQU  >8802        ;VDP STATUS REGISTER
****************************************
*      9902 ACC PROGRAMMING DATA       *
****************************************
ACCBYT BYTE >0           ;CONTROL BYTE
ACCSPD DATA >0           ;SPEED REGISTER
ACUNIT DATA >1           ;UNIT ADDRESS FOR 9902 (CRU)
DUPLEX BYTE >0           ;HALF/FULL DUPLEX FLAG
BD110  DATA 110          ;110 BPS
BD300  DATA 300          ;300 BPS
BD1200 DATA 1200         ;1200 BPS
****************************************
*      MISCEALLANEOUS FLAGS/TEMP AREAS  *
****************************************
BREAK  BYTE 187          ;BREAK KEY VALUE
BUFON  BYTE >0           ;IN-MEMORY CAPTURE BUFFER
BUFFER EQU  >C000
BUFST  DATA BUFFER       ;START OF IN-MEMORY BUFFER
BUFEND DATA BUFFER+16384 ;16K BUFFER!
BUFPTR DATA >0           ;CURRENT BUFFER LOCATION POINTER
DSKBUF BSS  256          ;FILE BUFFER FOR UPLOAD/DOWNLOAD
       TITL 'Terminal/99 Routine Vectors'
       PAGE
*****************************************************************************
*
*
*      ROUTINE VECTORS FOR TERMINAL/99
*
*
*
*****************************************************************************
TSTKEY DATA WS1
       DATA TSTKBD
*
TSTACC DATA WS1
       DATA TST902
*
XMTACC DATA WS1
       DATA XMT902
*****************************************************************************
```

```
*                                                                               *
*       INITIALIZATION CODE                                                     *
*                                                                               *
*********************************************************************************
*                                                                               *
*       FUNCTIONS:                                                              *
*       Clear out VDP memory                                                    *
*       Load Small Caps/lowercase character sets                                *
*       Get terminal configuration                                              *
*       Configure RS232/PIOs                                                    *
*       go to main routine                                                      *
*                                                                               *
*********************************************************************************
T99     EQU  $
SFIRST EQU  $
        LWPI WS
        LIMI 0
        LI   R0,>4000
        SWPB R0
        MOVB R0,@VDPWA
        SWPB R0
        MOVB R0,@VDPWA
        LI   R1,>1FFF
        LI   R2,>0000
ZEROVM NOP
        MOVB R2,@VDPWD
        DEC  R1
        JGT  ZEROVM
*
*
        LI   R0,>01F0
        SWPB R0
        MOVB R0,@VDPR1C
        SWPB R0
        BLWP @VWTR
        LI   R0,>0200
        BLWP @VWTR
        LI   R0,>0401
        BLWP @VWTR
        LI   R0,>07F5
        BLWP @VWTR
        LI   R0,>0900      ;SET VDP BASE
        LI   R1,CHRSET     ;POINT TO RESIDENT CHAR SET
        LI   R2,768        ;NBR BYTES IN CHARACTER SET
        BLWP @VMBW         ;MOVE THE CHARACTER SET
*
*       SET UP INVERSE CHARACTER SET...
*
        CLR  R0
        LI   R1,>4C00
        LI   R2,>FF00
```

```
        SWPB R1
        MOVB R1,@VDPWA
        SWPB R1
        MOVB R1,@VDPWA
INV1    NOP
        MOVB R2,@VDPWD
        INC  R0
        CI   R0,>0100
        JLE  INV1
        CLR  R0
        LI   R1,>4D00
        SWPB R1
        MOVB R1,@VDPWA
        SWPB R1
        MOVB R1,@VDPWA
        LI   R2,CHRSET
INV2    NOP
        MOVB *R2+,R3
        INV  R3
        MOVB R3,@VDPWD
        INC  R0
        CI   R0,768
        JLE  INV2
*       ACC CONFIG TEST (REPLACE WITH FULL CODE LATER)
        LI   R12,>1300
        SBO  0
        BL   @SETACC
        LI   R10,>10
FOO     DEC  R10
        JNE  FOO
        SBO  31
        SBZ  13
        LI   R1,>8300
        LDCR R1,8
        LI   R1,>04D0
        LDCR R1,12
        LI   R12,>1380
        SBO  31
        SBZ  13
        LI   R1,>B200
        LDCR R1,8
        LI   R1,52
        LDCR R1,12
        LI   R12,>1300
        SBZ  0
*       OPEN PRINTER
        TITL 'Terminal/99 Main Routine'
        PAGE
*************************************
*       M A I N   R O U T I N E     *
```

```
************************************
MAIN   BLWP @TSTKEY
       C    R0,@HAVKEY
       JEQ  PUTKEY
       BLWP @TSTACC
       MOV  R0,R0
       JNE  GETACC
       JMP  MAIN
CURSOR CB   @KBDCHR,@CR
       JEQ  MAIN
       CB   @KBDCHR,@BYTE80
       JH   MAIN
       CB   @KBDCHR,@LF
       JEQ  CSRDSP
       CB   @KBDCHR,@BS
       JEQ  CSRDSP
       CB   @KBDCHR,@US
       JLE  MAIN
CSRDSP MOV  @POSN,R0
       CLR  R1
       LI   R1,>8000
       BLWP @VSBW
       JMP  MAIN
*
*
PUTKEY CLR  R0
       MOVB @KBDCHR,@PREVCH
       BL   @GETKBD
       MOVB @KBDCHR,R0
       BLWP @XMTACC
       JMP  CURSOR
*
*
GETACC BL   @GET902
       MOVB R0,@KBDCHR
       BL   @DSPLY
       CB   @KBDCHR,@CR
       JEQ  MAIN
       JMP  CURSOR
*
*
TSTKBD LIMI 2
       LIMI 0
       CLR  *R13
       BLWP @KSCAN
       MOVB @>837C,*R13
       RTWP
*
*
GETKBD MOVB @KBD,@KBDCHR
       CB   @KBDCHR,@QUIT
```

```
       JEQ  BYE
       RT
BYE    CLR  R0
       MOVB R0,@STATUS
       LWPI GPLWS
       B    @>0070
*
*
TST902 LI   R12,>1300
       SBO  0
       BL   @SETACC
       CLR  *R13
       TB   21
       JNE  RCV902
       INC  *R13
RCV902 LI   R12,>1300
       SBZ  0
       RTWP
*
*
GET902 MOV  R11,R10
       LI   R12,>1300
       SBO  0
       BL   @SETACC
       CLR  R0
       STCR R0,8
       SBZ  18
       MOV  R10,R11
       LI   R12,>1300
       SBZ  0
       RT
*
*
XMT902 LI   R12,>1300
       SBO  0
       BL   @SETACC
       SBO  16
WTXBUF TB   22
       JNE  WTXBUF
       LDCR *R13,8
       SBZ  16
       LI   R12,>1300
       SBZ  0
       RTWP
*
SETACC LI   R4,>40
       MPY  @ACUNIT,R4
       AI   R5,>1300
       MOV  R5,R12
       RT
```

```
       TITL 'Terminal/99 Screen Handler'
       PAGE
****************************************
*      S C R E E N  H A N D L E R      *
****************************************
*      CHECK FOR STANDARD CONTROL CODES...
*
DSPLY  EQU  $
       MOV  R11,R8
       MOVB @KBDCHR,R0
       ANDI R0,>7FFF
       MOVB R0,@KBDCHR
       CB   @KBDCHR,@DC2
       JEQ  SETPRN
       CB   @KBDCHR,@DC4
       JNE  TSTPRN
       CLR  @PRNFLG
       JMP  CONT
SETPRN INC  @PRNFLG
TSTPRN MOV  @PRNFLG,@PRNFLG
       JEQ  CONT
*      PRINT CHARACTER ON PRINTER
       LI   R12,>1300
       SBO  0
       AI   R12,>0080
       SBO  16
PRNXMT TB   22
       JNE  PRNXMT
       MOVB @KBDCHR,R1
       LDCR R1,8
       SBZ  16
       LI   R12,>1300
       SBZ  0
*
CONT   EQU  $
       CB   @KBDCHR,@BS
       JEQ  BKSPAC
       CB   @KBDCHR,@BEL
       JEQ  BELL
       CB   @KBDCHR,@CR
       JEQ  CARRET
       CB   @KBDCHR,@LF
       JEQ  LNFD
       CB   @KBDCHR,@FF
       JEQ  CLR
*      CB   @KBDCHR,@SI
*      JEQ  SETALT
*      CB   @KBDCHR,@SO
*      JEQ  CLRALT
       CB   @KBDCHR,@ESC
       JEQ  ESCSEQ
```

```
*
*       ISNT A SPECIAL CONTROL CODE...JUST DISPLAY IT...
*
DSPLAY EQU  $
       CB   @KBDCHR,@US
       JLE  ENDDSP
       MOV  @POSN,R6
       AI   R6,>4000
       SWPB R6
       MOVB R6,@VDPWA
       SWPB R6
       MOVB R6,@VDPWA
       NOP
       MOV  @ALTFLG,@ALTFLG
       JNE  ALTDSP
DSP1   MOVB @KBDCHR,@VDPWD  ;AND PLACE IT ON SCREEN...
       MOV  @POSN,R6
       CI   R6,959
       JNE  BUMPX
       BL   @SCROLL
BUMPX  INC  @POSN
       INC  @XPOS
       MOV  @XPOS,R10
       CI   R10,40
       JEQ  FIXPOS
ENDDSP MOV  R8,R11       ;GO AND LOOK FOR MORE...
       RT
FIXPOS INC  @YPOS
       CLR  @XPOS
       JMP  ENDDSP
*
CLR    B    @CLRSCR
ESCSEQ B    @ESCAPE

ALTDSP MOVB @KBDCHR,R0
       ORI  R0,>8000
       MOVB R0,@KBDCHR
       JMP  DSP1
SETALT INC  @ALTFLG
       JMP  ENDDSP
CLRALT DEC  @ALTFLG
       JMP  ENDDSP
*
LNFD   BL   @LNFEED
       JMP  ENDDSP
*
*     SPECIAL ROUTINES...
*
BKSPAC MOV  @POSN,@POSN
       JEQ  BELL
```

```
        MOV  @POSN,R0
        LI   R1,>2000
        BLWP @VSBW
        JEQ  BELL
        DEC  @POSN
        MOV  @XPOS,@XPOS
        JNE  BK1
        LI   R7,40
        DEC  @YPOS
BK1     DEC  @XPOS
        JMP  ENDDSP
*
BELL    CLR  @STATUS
        BLWP @GPLLNK
        DATA >0034
        JMP  ENDDSP
*
*
CARRET EQU  $
        MOV  R8,R10
        CLR  R7
        MOV  @POSN,R8
        MOV  R8,R0
        CLR  R1
        BLWP @VSBR
        SLA  R1,1
        JNE  CR1
        LI   R1,SPACE
        MOV  R8,R0
        BLWP @VSBW
CR1     LI   R9,40
        DIV  R9,R7
        MPY  R9,R7
        MOV  R8,@POSN
        MOV  R10,R8
        JMP  ENDDSP
*
*
LNFEED EQU  $
        CLR  R1
        MOV  @POSN,R0
        BLWP @VSBR
        SLA  R1,1
        JNE  LF1
        MOV  @POSN,R0
        LI   R1,SPACE
        BLWP @VSBW
LF1     A    @CHRSLN,@POSN
        MOV  @POSN,R2
        CI   R2,959
        JLE  BUMPOS
```

```
       MOV  R11,R3
       BL   @SCROLL
       MOV  R3,R11
BUMPOS RT
*
*
SCROLL LI   R0,40          ;LOSE TOP LINE
       LI   R1,SCRBUF      ;POINT TO TEMP RAM
       LI   R2,920         ;FOR LINES 2-24
       BLWP @VMBR          ;MOVE 'EM
       LI   R0,920         ;CLEAR OUT LINE 24
       LI   R1,SPACES      ;SPACE LOCATION
       LI   R2,40          ;# CHARS/LINE
       BLWP @VMBW          ;BLANK THE LINE
       LI   R0,0
       LI   R1,SCRBUF
       LI   R2,920
       BLWP @VMBW          ;MOVE IT BACK...
       S    @CHRSLN,@POSN
       RT
*
*
CLRSCR LI   R0,>4000
       LI   R1,SPACE
       SWPB R0
       MOVB R0,@VDPWA
       SWPB R0
       MOVB R0,@VDPWA
       LI   R2,960
CLRVDP MOVB R1,@VDPWD
       DEC  R2
       JGT  CLRVDP
       CLR  @POSN
       B    @ENDDSP
*
ESCAPE B    @ENDDSP
SLAST  END
```

Disk 7. Contents of File TE12/DOC

TE3/DOC

The so-called TE3 emulator is a half-finished product from TI. It is a long way from being finished and the user should allow for this. The program supports no screen or cursor protocols and can be flaky at 1200 baud. Run the program from the LOAD AND RUN option and the file name is TE3 or TE3C. It will automatically run.

This version of TE3 has been modified by Joe Freeman CIS 75156,55 to make it a little more usable. A RAM trap, del key, and improved backspace have been added. The backspace modification is the reason for the flicker in the cursor. The cursor is alternating with the character that should be in the space. Other functions added include a 20K RAM trap, upload support, pulse dialer support for my acoustic coupled modem (write for details) and an online list of functions by the **AID** key.

Several function keys have useful functions:

FCTN 1	Backspace
FCTN 2	Escape character
FCTN 3	Toggle between 40 and 80 column mode
FCTN 4	Trade between left and right half of screen in 80 column mode
FCTN 5	Clear RAM buffer (reset buffer pointers)
FCTN 6	Output RAM buffer contents, you will be prompted for device name!
FCTN 7	List all of the important function and control keys.
FCTN 8	Send disk file to host (carriage returns can be added by program)
FCTN 9	Pulse dial by toggling bit 5 of RS232/1 to flip relay (special hardware)

Several control keys are useful also:

CTRL H	Backspace
CTRL .	Escape character
CTRL V	Del char for DEC systems
CTRL S	No scroll(no translation needed in emulator
CTRL Q	Start scroll (no translation needed in emulator

All of the above functions are self explanatory except for the RAM buff.

The RAM buffer traps all incoming characters up to 24K chars. The buffer is circular so that the last 20K is held. I don't recommend you rely on this feature when using the RAM buffer since the circular queue has not been fully tested. **FCTN 5** will clear the buffer and I recommend you use it just before listing anything important.

USING THE BUFFER

1. Sign on to the remote site
2. Do whatever you want until you find something you wish to save
3. Clear the RAM buffer
4. Have whatever text you wish to save listed to you
5. Log off the remote system
6. Dump the buffer using **FCTN 6**

This can save expensive connect time on CIS and other utilities, if you only stay on long enough to list and then log off while dumping the buffer to your printer or disk drive.

I recommend only using the RAM buffer for saving ASCII text files!

USING UPLOAD

The upload function will send VAR/80 TEXT files from the disk to the host site. You can choose to have carriage returns added at the end of each line or your file can have them in it. Answer **Y** at prompt to have carriage returns added to the sent file. Files created by the E/A editor need to have carriage returns added to the end. Most hosts must have a CR at the end of each line so make sure you have them one way or the other. You will not want to send files that contain carriage returns and linefeeds on each line. Hosts such as CompuServe will freak out. Also blank lines with no characters in them could cause problems.

METHOD OF UPLOAD

1. Sign on to host
2. Put host in input or insert mode designed for the user to just keep typing. End of line is signaled by CR.
3. Push **FCTN 8** to upload and type file name.
4. Pick the appropriate control method. VAXs and most newer equipment support XON/XOFF protocol which is faster while IBM supports nothing. They need the delay option which should be long enough even for them.
5. Read the paragraph above and decide if you wish to add CRs.
6. You have control again when the ". . .working" prompt disappears.

CONFIGURE FILE

A configure file will set up this program automatically. Create the file with the E/A editor and save it. Type in the file name at the "Configure File:" prompt and it will load. You will not have to answer the usual boot up questions. The data is all on one line and in the same order as the normal powerup. Here is an example.

```
1 N E 2 1 7       ---   for 300 baud, no echo, even parity, port 2,
                        1 stop bit and 7 data bits.
```

I hope this is of some help to those hardy souls who use the TI orphan.

Disk 7. Contents of File TERMEX/DOC

Terminal Emulator X for 99/4A

by Steven R. Sostrom.

This version is for use with the Editor/Assembler module. Last revision date: 06/08/83

The purpose of this program is to convert the 99/4A computer to a smart terminal. You can use ports 1-4 and they can by configured to your need. To load the program, go to the load and run menu and type in DSK1.TERMEX (you may use any drive, DSK1-DSK3). When the program is loaded, press **ENTER**. When the prompt for program name appears, enter START. The top level menu should appear. The menu will be as follows:

```
0  SELECT LINE LENGTH
1  SELECT RS232 PORT
2  SELECT BAUD RATE
3  SET ECHO ON/OFF
4  SET PARITY
5  SET DATA LENGTH
6  AUTO LINE FEED ON/OFF
7  SET DUPLEX TO FULL/HALF
8  START TERMINAL EMULATOR
9  EXIT
```

Each item of the top level menu has its own menu except for selection 9 which exits the program. You may return to a previous level of the program at any time by pressing **BACK (FCTN 9)** or **Escape (CTRL .)**.

All of the sub-menus should be self-explanatory, but I will discuss some of their features.

Line length in this version can be set to 32 or 40 columns. Do not use the 64 or 80 column selection because these selections have not been fully implemented and the result will be strange. The default is 40 columns.

The default RS232 port is port 1

Baud rate can be set in 9 increments from 110 baud to 19200 baud. This version cannot handle 19200 baud without some delay between characters. I have had excellent success with 19200 baud using TIBUG with the ASR flag set (TF in MAPBUG). Also, 9600 baud will miss certain characters occasionally, but still is usable for TIBUG at this baud rate. The default baud rate is 9600 baud.

If echo is set to on, all characters received will be echoed back to the sender on the same port. The default is echo off.

The default parity is odd.

The default data length is 7.

It should be noted that if the control keys that have an ASCII code of >80 or greater will not have the most significant bit transmitted if the data length is set to 7. Setting the data length to 8 will allow these codes to be transmitted correctly.

If auto line feed is set, a line feed is performed any time that a carriage return (enter) is encountered. The default for this is auto line feed off.

In full duplex mode, no entries made from the keyboard will be displayed except for the roll up and roll down keys. The default is half duplex, where all keyboard entries will be displayed by the program.

When you press **8** to start the terminal emulator, the prompt "Clear screen buffer? (Y/N) " will be displayed. When this program is first loaded, it is advisable to clear the buffer. The option is given so that the data in the buffer can be accessed even if the program is exited and re-entered. It is also advisable to clear the screen buffer when the line length has been changed. This version's buffer will hold 10 pages in the 40 column line mode.

The control characters are as follows:

KEY	*FUNCTION*	*CODE*	*COMMENTS*
CTRL G	BELL	>07	Bell will sound
CTRL H	BACKSPACE	>08	
CTRL J	LINE FEED	>0A	
CTRL L	FORM FEED	>0C	Will clear the screen
CTRL M	CARRIAGE RETURN	>0D	
CTRL .	ESCAPE	>1B	Return to the previous level
FCTN 4	ROLL UP		Page forward. No output code
FCTN =	QUIT	>85	Return to color bar screen
FCTN S	LEFT ARROW	>08	Backspace
FCTN D	RIGHT ARROW	>89	Forward space cursor
FCTN X	DOWN ARROW	>0A	Line feed
FCTN E	UP ARROW	>8B	Move cursor up 1 row
FCTN 6	ROLL DOWN		Page backward. No output code
ENTER	CARRIAGE RETURN	>0D	
FCTN 9	BACK	>8F	Perform an escape

When in the terminal emulator part of the program, you may output the entire contents of the buffer to any legal device with the possible exception of CS1. To do this, press **CTRL O**. The prompt "Save to file:" will appear. You may enter a device name of up to 2 lines long. When the output is completed, the previous screen will appear.

Note that when the screen buffer is full, no error message is produced. At this time, the first line entered will be written over and will not be blanked first.

Disk 8. TI Basic Games

Version: *Author*: Others, and Hoddie.
Requires: XB optional *Language*: TI Basic, XB *Updated*:

Includes 3D-Maze, checkers, animal, and color-guess. Also 3 games in assembly that load from XB: Bomb, Mouse, and Maze.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-8
Sectors total           = 360
Sectors used            = 183
Sectors available       = 175
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 3D-MAZE      39 PROGRAM      >022 038
002 >003 ANIMAL        8 PROGRAM      >048 007
003 >004 BLOCKADE     15 PROGRAM      >04f 014
004 >005 BOMB         14 PROGRAM    Y >05d 013
005 >006 CHECKERS     35 PROGRAM      >06a 034
006 >007 CLR-GUESS    15 PROGRAM      >08c 014
007 >008 LOAD          6 PROGRAM      >09a 005
008 >009 LOADBOMB      9 DIS/FIX 80 Y >09f 008
009 >00a LOADMAZE      9 DIS/FIX 80 Y >0a7 008
010 >00b LOADMOUS      9 DIS/FIX 80 Y >0af 008
011 >00c MAZE         11 PROGRAM    Y >0b7 010
012 >00d MOUS         13 PROGRAM    Y >0c1 012
```

Disk 9. TI-Forth Source Code — Disk 1

Version: Author: TI
Requires: EA Language: AL Updated:

Source code to TI-Forth. Very useful for "borrowing* routines from for your own assembly programs.

```
dskdir. v2.0. 12-dec-96
Disk name               = 03NOV82
Sectors total           = 360
Sectors used            = 250
Sectors available       = 108
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 ASMSRC        2 DIS/VAR 80   >022 001
002 >003 ASMSRC1      58 DIS/VAR 80   >023 057
003 >004 ASMSRC2      73 DIS/VAR 80   >05c 072
004 >005 ASMSRC3      64 DIS/VAR 80   >0a4 063
005 >006 FORTH2       53 DIS/FIX 80   >0e3 052
```

Disk 9. Contents of File ASMSRC

```
COPY "DSK2.ASMSRC1"
COPY "DSK2.ASMSRC2"
COPY "DSK2.ASMSRC3"
END
```

Disk 9. Contents of File ASMSRC1

```
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*
********************************************************************
DODOES EQU  >832E
DOCOL  EQU  >8334
$NEXT  EQU  >833A
DOEXEC EQU  >833C
$SEMIS EQU  >8340
********************************************************************
*
FF9900 LI   IP,COLD+2
       MOV  @$U0(TEMP1),U
       MOV  TEMP1,@$UCONS(U)
       MOV  @$UCONS(U),TEMP1
       MOV  @$S0(TEMP1),SP
       MOV  @$R0(TEMP1),R
       MOV  @$U0(TEMP1),U
       MOV  U,@$U0(U)
       LI   NEXT,$NEXT
       B    *NEXT
*
*
*** EXECUTE ***
       DATA >0
L1000  DATA >8745,>5845,>4355,>54C5
EXECUT DATA $+2
       MOV  *SP+,W
       B    @DOEXEC
*
*
*** LIT ***
       DATA L1000
L1001  DATA >834C,>49D4
```

```
LIT    DATA $+2
       DECT SP
       MOV  *IP+,*SP
       B    *NEXT
*
*
*** BRANCH ***
       DATA L1001
L1002  DATA >8642,>5241,>4E43,>48A0
BRANCH DATA $+2
BRAN2  A    *IP,IP
       B    *NEXT
*
*
*** 0BRANCH ***
       DATA L1002
L1003  DATA >8730,>4252,>414E,>43C8
ZBRAN  DATA $+2
       MOV  *SP+,TEMP1
       JEQ  ZBRAN1
       INCT IP
       B    *NEXT
ZBRAN1 A    *IP,IP
       B    *NEXT
*
*
*** (OF) ***
       DATA L1003
L1004  DATA >8428,>4F46,>29A0
POF    DATA $+2
       C    *SP+,*SP
       JNE  POF1
       INCT SP
       INCT IP
       B    *NEXT
POF1   A    *IP,IP
       B    *NEXT
*
*
*** (LOOP) ***
       DATA L1004
L1005  DATA >8628,>4C4F,>4F50,>29A0
PLOOP  DATA $+2
       INC  *R
       C    *R,@2(R)
       JLT  PLOOPA
       AI   R,4
       INCT IP
       B    *NEXT
PLOOPA A    *IP,IP
```

```
       B    *NEXT
*
*
*** (+LOOP) ***
       DATA L1005
L1006  DATA >8728,>2B4C,>4F4F,>50A9
PPLOOP DATA $+2
       MOV  *SP+,TEMP1
       A    TEMP1,*R
       MOV  TEMP1,TEMP1
       JLT  PLOOP2
PLOOP1 C    *R,@2(R)
       JLT  PLOOP3
       AI   R,4
       INCT IP
       B    *NEXT
PLOOP2 C    *R,@2(R)
       JGT  PLOOP3
       AI   R,4
       INCT IP
       B    *NEXT
PLOOP3 A    *IP,IP
       B    *NEXT
*
*
*** (DO) ***
       DATA L1006
L1007  DATA >8428,>444F,>29A0
PDO    DATA $+2
       AI   R,-4
       MOV  *SP+,*R
       MOV  *SP+,@2(R)
       B    *NEXT
*
*
*** I ***
       DATA L1007
L1008  DATA >81C9
I      DATA $+2
       DECT SP
       MOV  *R,*SP
       B    *NEXT
*
*
*** J ***
       DATA L1008
J1008  DATA >81CA
J      DATA $+2
       DECT SP
       MOV  @4(R),*SP
       B    *NEXT
```

```
*
*
*** DIGIT ***
       DATA J1008
L1009  DATA >8544,>4947,>49D4
DIGIT  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP,TEMP2
       AI   TEMP2,->0030
       CI   TEMP2,10
       JL   DIGIT1
       AI   TEMP2,-7
       CI   TEMP2,10
       JHE  DIGIT1
DIGIT2 CLR  *SP
       B    *NEXT
DIGIT1 C    TEMP2,TEMP1
       JHE  DIGIT2
       MOV  TEMP2,*SP
       DECT  SP
       SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** (FIND) ***
       DATA L1009
L100A  DATA >8628,>4649,>4E44,>29A0
PFIND  DATA $+2
       MOV  *SP,TEMP1
       JEQ  PFIND4
PFIND1 MOV  TEMP1,TEMP2
       MOV  @2(SP),TEMP3
       MOVB *TEMP2+,W
       ANDI W,>3F00
       CB   W,*TEMP3+
       JNE  PFIND3
PFIND2 MOVB *TEMP2+,W
       JLT  PFIND5
       CB   W,*TEMP3+
       JEQ  PFIND2
PFIND3 MOV  @-2(TEMP1),TEMP1
       JNE  PFIND1
PFIND4 INCT SP
       CLR  *SP
       B    *NEXT
PFIND5 ANDI W,>7F00
       CB   W,*TEMP3
       JNE  PFIND3
       INCT TEMP2
```

```
       MOV  TEMP2,@2(SP)
       CLR  *SP
       MOVB *TEMP1,@1(SP)
       DECT SP
       SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** ENCLOSE ***
       DATA L100A
L100B  DATA >8745,>4E43,>4C4F,>53C5
ENCLOS DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP,TEMP2
       SWPB TEMP1
       SETO TEMP3
ENCL1  INC  TEMP3
       CB   TEMP1,*TEMP2+
       JEQ  ENCL1
       DEC  TEMP2
       AI   SP,-6
       MOV  TEMP3,@4(SP)
       MOV  TEMP3,*SP
       INC  TEMP3
       MOV  TEMP3,@2(SP)
       MOVB *TEMP2,W
       JNE  ENCL4
       B    *NEXT
ENCL4  INC  TEMP2
ENCL2  MOV  TEMP3,@2(SP)
       MOVB *TEMP2,W
       JEQ  ENCL3
       INC  TEMP3
       CB   TEMP1,*TEMP2+
       JNE  ENCL2
ENCL3  MOV  TEMP3,*SP
       B    *NEXT
*
*
*** kEY ***
       DATA L100B
L100C  DATA >836B,>45D9
KE     DATA $+2
       LI   TEMP1,-2
       MOV  @$SYS(U),LINK
       BL   *LINK
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
```

```
*
*** KEY ***
       DATA L100C
L100CX DATA >834B,>45D9
KEY    DATA DOCOL,KE,LIT,>7F,AND,SEMIS
*
*
*** KEY8 ***
       DATA L100CX
L100CY DATA >844B,>4559,>38A0
KEY8   DATA DOCOL,KE,SEMIS
*
*
*** EMIT ***
       DATA L100CY
L100D  DATA >8445,>4D49,>54A0
EMIT   DATA $+2
       MOV  *SP+,TEMP2
       ANDI TEMP2,>007F
       LI   TEMP1,-4
       MOV  @$SYS(U),LINK
       BL   *LINK
       INC  @$OUT(U)
       B    *NEXT
*
*
*** EMIT8 ***
       DATA L100D
L100DX DATA >8545,>4D49,>54B8
EMIT8  DATA $+2
       MOV  *SP+,TEMP2
       ANDI TEMP2,>00FF
       LI   TEMP1,-4
       MOV  @$SYS(U),LINK
       BL   *LINK
       INC  @$OUT(U)
       B    *NEXT
*
*
*** CR ***
       DATA L100DX
L100E  DATA >8243,>52A0
CR     DATA $+2
       LI   TEMP1,-6
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
*
*
*** ?TERMINAL ***
```

```
       DATA L100E
L100F  DATA >893F,>5445,>524D,>494E,>41CC
QTERM  DATA $+2
       LI   TEMP1,-8
       MOV  @$SYS(U),LINK
       BL   *LINK
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** ?KEY ***
       DATA L100F
L1010  DATA >843F,>4B45,>59A0
QKEY   DATA $+2
       LI   TEMP1,-10
       MOV  @$SYS(U),LINK
       BL   *LINK
       ANDI TEMP0,>007F
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** ?KEY8 ***
       DATA L1010
L1010X DATA >853F,>4B45,>59B8
QKEY8  DATA $+2
       LI   TEMP1,-10
       MOV  @$SYS(U),LINK
       BL   *LINK
       ANDI TEMP0,>00FF
       DECT SP
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** GOTOXY ***
       DATA L1010X
L1011  DATA >8647,>4F54,>4F58,>59A0
GOTOXY DATA $+2
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       LI   TEMP1,-12
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
*
*
*** WDISK ***
       DATA L1011
```

```
L1012  DATA >8557,>4449,>53CB
WDISK  DATA $+2
       LI   TEMP1,-14
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP4
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** RDISK ***
       DATA L1012
L1013  DATA >8552,>4449,>53CB
RDISK  DATA $+2
       LI   TEMP1,-16
       MOV  *SP+,TEMP3
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP4
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,*SP
       B    *NEXT
*
*
*** DRIVE ***
       DATA L1013
L1014  DATA >8544,>5249,>56C5
DRIVE  DATA $+2
       MOV  *SP+,TEMP2
       LI   TEMP1,-18
       MOV  @$SYS(U),LINK
       BL   *LINK
       MOV  TEMP0,@$OFFST(U)
       B    *NEXT
*
*
*** CMOVE ***
       DATA L1014
L1015  DATA >8543,>4D4F,>56C5
CMOVE  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOV  *SP+,TEMP3
       MOV  TEMP1,TEMP1
       JEQ  CMOVE2
CMOVE1 MOVB *TEMP3+,*TEMP2+
       DEC  TEMP1
       JNE  CMOVE1
```

```
CMOVE2 B    *NEXT
*
*
*** MOVE ***
       DATA L1015
A1000  DATA >844D,>4F56,>45A0
MOVE   DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOV  *SP+,TEMP3
       MOV  TEMP1,TEMP1
       JEQ  MOVE2
MOVE1  MOV  *TEMP3+,*TEMP2+
       DEC  TEMP1
       JNE  MOVE1
MOVE2  B    *NEXT
*
*
*** SWPB ***
       DATA A1000
A1001  DATA >8453,>5750,>42A0
SWPB   DATA $+2
       SWPB *SP
       B    *NEXT
*
*
*** SRL ***
       DATA A1001
A1002  DATA >8353,>52CC
SRL    DATA $+2
       MOV  *SP+,TEMP0
       MOV  *SP,TEMP1
       SRL  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SLA ***
       DATA A1002
A1003  DATA >8353,>4CC1
SLA    DATA $+2
       MOV  *SP+,TEMP0
       MOV  *SP,TEMP1
       SLA  TEMP1,0
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SRA ***
       DATA A1003
A1004  DATA >8353,>52C1
```

```
SRA     DATA $+2
        MOV  *SP+,TEMP0
        MOV  *SP,TEMP1
        SRA  TEMP1,0
        MOV  TEMP1,*SP
        B    *NEXT
*
*
*** SRC ***
        DATA A1004
A1005   DATA >8353,>52C3
SRC     DATA $+2
        MOV  *SP+,TEMP0
        MOV  *SP,TEMP1
        SRC  TEMP1,0
        MOV  TEMP1,*SP
        B    *NEXT
*
*
*** U* ***
        DATA A1005
L1016   DATA >8255,>2AA0
MULT    DATA $+2
        MOV  *SP+,TEMP2
        MPY  *SP,TEMP2
        MOV  TEMP3,*SP
        DECT SP
        MOV  TEMP2,*SP
        B    *NEXT
*
*
*** U/ ***
        DATA L1016
L1017   DATA >8255,>2FA0
DIV     DATA $+2
        MOV  @2(SP),TEMP2
        MOV  @4(SP),TEMP3
        DIV  *SP+,TEMP2
        MOV  TEMP2,*SP
        MOV  TEMP3,@2(SP)
        B    *NEXT
*
*
*** AND ***
        DATA L1017
L1018   DATA >8341,>4EC4
AND     DATA $+2
        INV  *SP
        SZC  *SP+,*SP
        B    *NEXT
```

```
*
*
*** OR ***
       DATA L1018
L1019  DATA >824F,>52A0
OR     DATA $+2
       SOC  *SP+,*SP
       B    *NEXT
*
*
*** XOR ***
       DATA L1019
L101A  DATA >8358,>4FD2
XOR    DATA $+2
       MOV  *SP+,TEMP1
       XOR  *SP,TEMP1
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** SP@ ***
       DATA L101A
L101B  DATA >8353,>50C0
SPAT   DATA $+2
       DECT SP
       MOV  SP,*SP
       INCT *SP
       B    *NEXT
*
*
*** SP! ***
       DATA L101B
L101C  DATA >8353,>50A1
SPSTOR DATA $+2
       MOV  @$S0(U),SP
       B    *NEXT
*
*
*** RP! ***
       DATA L101C
L101D  DATA >8352,>50A1
RSTOR  DATA $+2
       MOV  @$R0(U),R
       B    *NEXT
*
*
*** ;S ***
       DATA L101D
L101E  DATA >823B,>53A0
SEMIS  DATA $SEMIS
*
```

```
*
*** LEAVE ***
       DATA L101E
L101F  DATA >854C,>4541,>56C5
LEAVE  DATA $+2
       MOV  *R,@2(R)
       B    *NEXT
*
*
*** >R ***
       DATA L101F
L1020  DATA >823E,>52A0
TOR    DATA $+2
       DECT R
       MOV  *SP+,*R
       B    *NEXT
*
*
*** R> ***
       DATA L1020
L1021  DATA >8252,>3EA0
FROMR  DATA $+2
       DECT SP
       MOV  *R+,*SP
       B    *NEXT
*
*
*** R ***
       DATA L1021
L1022  DATA >81D2
RR     DATA $+2
       DECT SP
       MOV  *R,*SP
       B    *NEXT
*
*
*** U ***
       DATA L1022
L1023  DATA >81D5
UU     DATA $+2
       DECT SP
       MOV  U,*SP
       B    *NEXT
*
*
*** 0= ***
       DATA L1023
L1024  DATA >8230,>3DA0
ZEQU   DATA $+2
       MOV  *SP,TEMP1
```

```
       JEQ  ZEQUTR
       CLR  *SP
       B    *NEXT
ZEQUTR SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** 0< ***
       DATA L1024
L1025  DATA >8230,>3CA0
ZLESS  DATA $+2
       MOV  *SP,TEMP1
       JLT  PUSHTR
PUSHFL CLR  *SP
       B    *NEXT
PUSHTR SETO *SP
       NEG  *SP
       B    *NEXT
*
*
*** + ***
       DATA L1025
L1026  DATA >81AB
PLUS   DATA $+2
       A    *SP+,*SP
       B    *NEXT
*
*
*** D+ ***
       DATA L1026
L1027  DATA >8244,>2BA0
DPLUS  DATA $+2
       A    *SP+,@2(SP)
       A    *SP+,@2(SP)
       JNC  DPLUS1
       INC  *SP
DPLUS1 B    *NEXT
*
*
*** MINUS ***
       DATA L1027
L1028  DATA >854D,>494E,>55D3
MINUS  DATA $+2
       NEG  *SP
       B    *NEXT
*
*
*** DMINUS ***
       DATA L1028
L1029  DATA >8644,>4D49,>4E55,>53A0
```

```
DMINUS DATA $+2
       INV  @2(SP)
       INV  *SP
       INC  @2(SP)
       JNC  DM1
       INC  *SP
DM1    B    *NEXT
*
*
*** OVER ***
       DATA L1029
L102A  DATA >844F,>5645,>52A0
OVER   DATA $+2
       DECT SP
       MOV  @4(SP),*SP
       B    *NEXT
*
*
*** DROP ***
       DATA L102A
L102B  DATA >8444,>524F,>50A0
DROP   DATA $+2
       INCT SP
       B    *NEXT
*
*
*** SWAP ***
       DATA L102B
L102C  DATA >8453,>5741,>50A0
SWAP   DATA $+2
       MOV  *SP,TEMP1
       MOV  @2(SP),*SP
       MOV  TEMP1,@2(SP)
       B    *NEXT
*
*
*** DUP ***
       DATA L102C
L102D  DATA >8344,>55D0
DUP    DATA $+2
       DECT SP
       MOV  @2(SP),*SP
       B    *NEXT
*
*
*** +! ***
       DATA L102D
L102E  DATA >822B,>21A0
PSTORE DATA $+2
       MOV  *SP+,TEMP1
```

```
       A    *SP+,*TEMP1
       B    *NEXT
*
*
*** TOGGLE ***
       DATA L102E
L102F  DATA >8654,>4F47,>474C,>45A0
TOGGLE DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,TEMP2
       MOVB *TEMP2,TEMP3
       SWPB TEMP1
       XOR  TEMP1,TEMP3
       MOVB TEMP3,*TEMP2
       B    *NEXT
*
*
*** @ ***
       DATA L102F
L1030  DATA >81C0
AT     DATA $+2
       MOV  *SP,TEMP1
       MOV  *TEMP1,*SP
       B    *NEXT
*
*
*** C@ ***
       DATA L1030
L1031  DATA >8243,>40A0
CAT    DATA $+2
       MOV  *SP,TEMP1
       MOVB *TEMP1,TEMP1
       SRL  TEMP1,8
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** ! ***
       DATA L1031
L1032  DATA >81A1
STORE  DATA $+2
       MOV  *SP+,TEMP1
       MOV  *SP+,*TEMP1
       B    *NEXT
*
*
*** C! ***
       DATA L1032
L1033  DATA >8243,>21A0
CSTORE DATA $+2
       MOV  *SP+,TEMP1
```

```
       MOVB @1(SP),*TEMP1
       INCT SP
       B    *NEXT
*
*
*** 1+ ***
       DATA L1033
L1034  DATA >8231,>2BA0
ONEP   DATA $+2
       INC  *SP
       B    *NEXT
*
*
*** 2+ ***
       DATA L1034
L1035  DATA >8232,>2BA0
TWOP   DATA $+2
       INCT *SP
       B    *NEXT
*
*
*** 1- ***
       DATA L1035
L1035A DATA >8231,>2DA0
ONEM   DATA $+2
       DEC  *SP
       B    *NEXT
*
*
*** 2- ***
       DATA L1035A
L1035B DATA >8232,>2DA0
TWOM   DATA $+2
       DECT *SP
       B    *NEXT
*
*
*** - ***
       DATA L1035B
L1036  DATA >81AD
SUB    DATA $+2
       S    *SP+,*SP
       B    *NEXT
*
*
*** =CELLS ***
       DATA L1036
L1037  DATA >863D,>4345,>4C4C,>53A0
ECELLS DATA $+2
       MOV  *SP,TEMP1
```

```
       INC  TEMP1
       ANDI TEMP1,>FFFE
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** S->D ***
       DATA L1037
L1038  DATA >8453,>2D3E,>44A0
STOD   DATA $+2
       SETO TEMP1
       MOV  *SP,TEMP2
       JLT STOD1
       CLR  TEMP1
STOD1  DECT SP
       MOV  TEMP1,*SP
       B    *NEXT
*
*
*** ABS ***
       DATA L1038
L1039  DATA >8341,>42D3
ABS    DATA $+2
       ABS  *SP
       B    *NEXT
*
*
*** MIN ***
       DATA L1039
L103A  DATA >834D,>49CE
MIN    DATA $+2
       C    @2(SP),*SP
       JLT  MIN1
       MOV  *SP,@2(SP)
MIN1   INCT SP
       B    *NEXT
*
*
*** MAX ***
       DATA L103A
L103B  DATA >834D,>41D8
MAX    DATA $+2
       C    *SP,@2(SP)
       JLT  MAX1
       MOV  *SP,@2(SP)
MAX1   INCT SP
```

Disk 9. Contents of File ASMSRC2

```
       B    *NEXT
*
*
*** U< ***
       DATA L103B
L103C  DATA >8255,>3CA0
ULESS  DATA $+2
       MOV  *SP+,TEMP2
       MOV  *SP,TEMP1
       CLR  *SP
       C    TEMP1,TEMP2
       JHE  ULESS1
       INC  *SP
ULESS1 B    *NEXT
*
*
*** 0 ***
       DATA L103C
L103F  DATA >81B0
ZERO   DATA DOCON,>0
*
*** 1 ***
       DATA L103F
L1040  DATA >81B1
ONE    DATA DOCON,>1
*
*** 2 ***
       DATA L1040
L1041  DATA >81B2
TWO    DATA DOCON,>2
*
*** 3 ***
       DATA L1041
L1042  DATA >81B3
THREE  DATA DOCON,>3
*
*** BL ***
       DATA L1042
L1043  DATA >8242,>4CA0
BL     DATA DOCON,>20
*
*** UCONS$ ***
       DATA L1043
L1044  DATA >8655,>434F,>4E53,>24A0
UCONS$ DATA DOUSER,>6
*
*** S0 ***
       DATA L1044
```

```
L1045  DATA >8253,>30A0
S0     DATA DOUSER,>8
*
*** R0 ***
       DATA L1045
L1046  DATA >8252,>30A0
RR0    DATA DOUSER,>A
*
*** U0 ***
       DATA L1046
L1047  DATA >8255,>30A0
U0     DATA DOUSER,>C
*
*** TIB ***
       DATA L1047
L1048  DATA >8354,>49C2
TIB    DATA DOUSER,>E
*
*** WIDTH ***
       DATA L1048
L1049  DATA >8557,>4944,>54C8
WIDTH  DATA DOUSER,>10
*
*** DP ***
       DATA L1049
L104A  DATA >8244,>50A0
DP     DATA DOUSER,>12
*
*** SYS$ ***
       DATA L104A
L104B  DATA >8453,>5953,>24A0
SYS$   DATA DOUSER,>14
*
*** CURPOS ***
       DATA L104B
L104C  DATA >8643,>5552,>504F,>53A0
TERM$  DATA DOUSER,>16
*
*** INTLNK ***
       DATA L104C
L104D  DATA >8649,>4E54,>4C4E,>4BA0
DISK$  DATA DOUSER,>18
*
*** WARNING ***
       DATA L104D
L104E  DATA >8757,>4152,>4E49,>4EC7
WARNIN DATA DOUSER,>1A
*
*** C/L$ ***
       DATA L104E
L104F  DATA >8443,>2F4C,>24A0
```

```
CL$    DATA DOUSER,>1C
*
*** FIRST$ ***
       DATA L104F
L1050  DATA >8646,>4952,>5354,>24A0
FIRST$ DATA DOUSER,>1E
*
*** LIMIT$ ***
       DATA L1050
L1051  DATA >864C,>494D,>4954,>24A0
LIMIT$ DATA DOUSER,>20
*
*** B/BUF$ ***
       DATA L1051
L1052  DATA >8642,>2F42,>5546,>24A0
BBUF$  DATA DOUSER,>22
*
*** B/SCR$ ***
       DATA L1052
L1053  DATA >8642,>2F53,>4352,>24A0
BSCR$  DATA DOUSER,>24
*
*** DISK_LO ***
       DATA L1053
X0001  DATA >8744,>4953,>4B5F,>4CCF
       DATA DOUSER,>26
*
*** DISK_HI ***
       DATA X0001
X0002  DATA >8744,>4953,>4B5F,>48C9
       DATA DOUSER,>28
*
*** DISK_SIZE ***
       DATA X0002
X0003  DATA >8944,>4953,>4B5F,>5349,>5AC5
       DATA DOUSER,>2A
*
*** DISK_BUF ***
       DATA X0003
X0004  DATA >8844,>4953,>4B5F,>4255,>46A0
       DATA DOUSER,>2C
*
*** PABS ***
       DATA X0004
X0005  DATA >8450,>4142,>53A0
       DATA DOUSER,>2E
*
*** SCRN_WIDTH ***
       DATA X0005
X0006  DATA >8A53,>4352,>4E5F,>5749,>4454,>48A0
```

```
       DATA DOUSER,>30
*
*** SCRN_START ***
       DATA X0006
X0007  DATA >8A53,>4352,>4E5F,>5354,>4152,>54A0
       DATA DOUSER,>32
*
*** SCRN_END ***
       DATA X0007
X0008  DATA >8853,>4352,>4E5F,>454E,>44A0
       DATA DOUSER,>34
*
*** ISR ***
       DATA X0008
X0009  DATA >8349,>53D2
       DATA DOUSER,>36
*
*** ALTIN ***
       DATA X0009
X000A  DATA >8541,>4C54,>49CE
       DATA DOUSER,>38
*
*** ALTOUT ***
       DATA X000A
X000B  DATA >8641,>4C54,>4F55,>54A0
       DATA DOUSER,>3A
*
*** FENCE ***
       DATA X000B
L1054  DATA >8546,>454E,>43C5
FENCE  DATA DOUSER,>3C
*
*** BLK ***
       DATA L1054
L1055  DATA >8342,>4CCB
BLK    DATA DOUSER,>3E
*
*** IN ***
       DATA L1055
L1056  DATA >8249,>4EA0
IN     DATA DOUSER,>40
*
*** OUT ***
       DATA L1056
L1057  DATA >834F,>55D4
OUT    DATA DOUSER,>42
*
*** SCR ***
       DATA L1057
L1058  DATA >8353,>43D2
SCR    DATA DOUSER,>44
```

```
*
*** OFFSET ***
       DATA L1058
L1059  DATA >864F,>4646,>5345,>54A0
OFFSET DATA DOUSER,>46
*
*** CONTEXT ***
       DATA L1059
L105A  DATA >8743,>4F4E,>5445,>58D4
CONTEX DATA DOUSER,>48
*
*** CURRENT ***
       DATA L105A
L105B  DATA >8743,>5552,>5245,>4ED4
CURREN DATA DOUSER,>4A
*
*** STATE ***
       DATA L105B
L105C  DATA >8553,>5441,>54C5
STATE  DATA DOUSER,>4C
*
*** BASE ***
       DATA L105C
L105D  DATA >8442,>4153,>45A0
BASE   DATA DOUSER,>4E
*
*** DPL ***
       DATA L105D
L105E  DATA >8344,>50CC
DPL    DATA DOUSER,>50
*
*** FLD ***
       DATA L105E
L105F  DATA >8346,>4CC4
FLD    DATA DOUSER,>52
*
*** CSP ***
       DATA L105F
L1060  DATA >8343,>53D0
CSP    DATA DOUSER,>54
*
*** R# ***
       DATA L1060
L1061  DATA >8252,>23A0
RNUM   DATA DOUSER,>56
*
*** HLD ***
       DATA L1061
L1062  DATA >8348,>4CC4
HLD    DATA DOUSER,>58
```

```
*
*** USE ***
       DATA L1062
L1063  DATA >8355,>53C5
USE    DATA DOUSER,>5A
*
*** PREV ***
       DATA L1063
L1064  DATA >8450,>5245,>56A0
PREV   DATA DOUSER,>5C
*
*** FORTH_LINK ***
       DATA L1064
L1065  DATA >8A46,>4F52,>5448,>5F4C,>494E,>4BA0
FORTHL DATA DOUSER,>62
*
*** ECOUNT ***
       DATA L1065
L1066  DATA >8645,>434F,>554E,>54A0
ECOUNT DATA DOUSER,>64
*
*** VOC-LINK ***
       DATA L1066
L1066X DATA >8856,>4F43,>2D4C,>494E,>4BA0
VLINK  DATA DOUSER,>66
*
*
       DORG 0
UBASE  BSS  6             BASE OF USER VARIABLES
$UCONS BSS  2             06 USER UCONS
$S0    BSS  2             08 USER S0
$R0    BSS  2             0A USER R0 { R0$
$U0    BSS  2             0C USER U0
       BSS  2             0E USER TIB
       BSS  2             10 USER WIDTH
       BSS  2             12 USER DP
$SYS   BSS  2             14 USER SYS$
CURPO$ BSS  2             16 USER CURPOS
$INTLK BSS  2             18 USER INTLNK
       BSS  2             1A USER WARNING
       BSS  2             1C USER C/L$ { CL$
       BSS  2             1E USER FIRST$
       BSS  2             20 USER LIMIT$
       BSS  2             22 USER B/BUF$ { BBUF$
       BSS  2             24 USER B/SCR$ { BSCR$
       BSS  2             26 USER DISK_LO
       BSS  2             28 USER DISK_HI
       BSS  2             2A USER DISK_SIZE
       BSS  2             2C USER DISK_BUF
       BSS  2             2E USER PABS
       BSS  2             30 USER SCRN_WIDTH
```

```
       BSS  2             32 USER SCRN_START
       BSS  2             34 USER SCRN_END
       BSS  2             36 USER ISR
       BSS  2             38 USER ALTIN
       BSS  2             3A USER ALTOUT
ULNGTH EQU  $
       BSS  2             3C USER FENCE
       BSS  2             3E USER BLK
       BSS  2             40 USER IN
$OUT   BSS  2             42 USER OUT
       BSS  2             44 USER SCR
$OFFST BSS  2             46 USER OFFSET
       BSS  2             48 USER CONTEXT
       BSS  2             4A USER CURRENT
       BSS  2             4C USER STATE
       BSS  2             4E USER BASE
       BSS  2             50 USER DPL
       BSS  2             52 USER FLD
       BSS  2             54 USER CSP
       BSS  2             56 USER R# { RNUM
       BSS  2             58 USER HLD
       BSS  2             5A USER USE
       BSS  2             5C USER PREV
       BSS  2             5E
       BSS  2             60
       BSS  2             62 USER FORTH_LINK
       BSS  2             64 USER ECOUNT
UMAX   BSS  0
       RORG
*
*** C/L ***
       DATA L1066X
L1067  DATA >8343,>2FCC
CSL    DATA DOCOL,CL$,AT,SEMIS
*
*** B/BUF ***
       DATA L1067
L1068  DATA >8542,>2F42,>55C6
BSLBUF DATA DOCOL,BBUF$,AT,SEMIS
*
*** B/SCR ***
       DATA L1068
L1069  DATA >8542,>2F53,>43D2
BSLSCR DATA DOCOL,BSCR$,AT,SEMIS
*
*** FIRST ***
       DATA L1069
L106A  DATA >8546,>4952,>53D4
FIRST  DATA DOCOL,FIRST$,AT,SEMIS
*
```

```
*** LIMIT ***
       DATA L106A
L106B  DATA >854C,>494D,>49D4
LIMIT  DATA DOCOL,LIMIT$,AT,SEMIS
*
*** DR0 ***
       DATA L106B
L106C  DATA >8344,>52B0
DR0    DATA DOCOL,ZERO,DRIVE,SEMIS
*
*** DR1 ***
       DATA L106C
L106X  DATA >8344,>52B1
DR1    DATA DOCOL,ONE,DRIVE,SEMIS
*
*** DR2 ***
       DATA L106X
L106Y  DATA >8344,>52B2
DR2    DATA DOCOL,TWO,DRIVE,SEMIS
*
*** HERE ***
       DATA L106Y
L106D  DATA >8448,>4552,>45A0
HERE   DATA DOCOL,DP,AT,SEMIS
*
*** ALLOT ***
       DATA L106D
L106E  DATA >8541,>4C4C,>4FD4
ALLOT  DATA DOCOL,SPAT,OVER,HERE,PLUS,LIT,>80
       DATA PLUS,ULESS,TWO,QERROR,DP,PSTORE
       DATA SEMIS
*
***  ,  ***
       DATA L106E
L106F  DATA >81AC
COMMA  DATA DOCOL,HERE,STORE,TWO,ALLOT,SEMIS
*
*** C, ***
       DATA L106F
L1070  DATA >8243,>2CA0
CCOMMA DATA DOCOL,HERE,CSTORE,ONE,ALLOT,SEMIS
*
*** = ***
       DATA L1070
L1071  DATA >81BD
EQUAL  DATA DOCOL,SUB,ZEQU,SEMIS
*
*** < ***
       DATA L1071
L1072  DATA >81BC
LESS   DATA $+2
```

```
       CLR  TEMP1
       C    *SP+,*SP
       JLT  LESS1
       JEQ  LESS1
       INC  TEMP1
LESS1  MOV  TEMP1,*SP
       B    *NEXT
*
*** > ***
       DATA L1072
L1073  DATA >81BE
GREAT  DATA DOCOL,SWAP,LESS,SEMIS
*
*** ROT ***
       DATA L1073
L1074  DATA >8352,>4FD4
ROT    DATA DOCOL,TOR,SWAP,FROMR,SWAP,SEMIS
*
*** SPACE ***
       DATA L1074
L1075  DATA >8553,>5041,>43C5
SPACE  DATA DOCOL,BL,EMIT,SEMIS
*
*** -DUP ***
       DATA L1075
L1076  DATA >842D,>4455,>50A0
DDUP   DATA DOCOL,DUP,ZBRAN,L1077-$,DUP
L1077  DATA SEMIS
*
*** TRAVERSE ***
       DATA L1076
L1078  DATA >8854,>5241,>5645,>5253,>45A0
TRAVER DATA DOCOL,SWAP
L1079  DATA OVER,PLUS,LIT,>7F,OVER,CAT,LESS,ZBRAN
       DATA L1079-$,SWAP,DROP,SEMIS
*
*** CFA ***
       DATA L1078
L107A  DATA >8343,>46C1
CFA    DATA DOCOL,TWOM,SEMIS
*
*** NFA ***
       DATA L107A
L107B  DATA >834E,>46C1
NFA    DATA DOCOL,THREE,SUB,LIT,>FFFF,TRAVER,SEMIS
*
*** PFA ***
       DATA L107B
L107C  DATA >8350,>46C1
PFA    DATA DOCOL,ONE,TRAVER,THREE,PLUS,SEMIS
```

```
*
*** LFA ***
       DATA L107C
L107D  DATA >834C,>46C1
LFA    DATA DOCOL,NFA,TWOM,SEMIS
*
*** LATEST ***
       DATA L107D
L107E  DATA >864C,>4154,>4553,>54A0
LATEST DATA DOCOL,CURREN,AT,AT,SEMIS
*
*** !CSP ***
       DATA L107E
L107F  DATA >8421,>4353,>50A0
STRCSP DATA DOCOL,SPAT,CSP,STORE,SEMIS
*
*** ?ERROR ***
       DATA L107F
L1080  DATA >863F,>4552,>524F,>52A0
QERROR DATA DOCOL,SWAP,ZBRAN,L1081-$,ERROR,BRANCH
       DATA L1082-$
L1081  DATA DROP
L1082  DATA SEMIS
*
*** ?COMP ***
       DATA L1080
L1083  DATA >853F,>434F,>4DD0
QCOMP  DATA DOCOL,STATE,AT,ZEQU,LIT,>11,QERROR
       DATA SEMIS
*
*** ?EXEC ***
       DATA L1083
L1084  DATA >853F,>4558,>45C3
QEXEC  DATA DOCOL,STATE,AT,LIT,>12,QERROR,SEMIS
*
*** ?PAIRS ***
       DATA L1084
L1085  DATA >863F,>5041,>4952,>53A0
QPAIRS DATA DOCOL,SUB,LIT,>13,QERROR,SEMIS
*
*** ?CSP ***
       DATA L1085
L1086  DATA >843F,>4353,>50A0
QCSP   DATA DOCOL,SPAT,CSP,AT,SUB,LIT,>14,QERROR
       DATA SEMIS
*
*** ?LOADING ***
       DATA L1086
L1087  DATA >883F,>4C4F,>4144,>494E,>47A0
QLOADI DATA DOCOL,BLK,AT,ZEQU,LIT,>16,QERROR,SEMIS
*
```

```
*** COMPILE ***
       DATA L1087
L1088  DATA >8743,>4F4D,>5049,>4CC5
COMPIL DATA DOCOL,QCOMP,FROMR,DUP,TWOP,TOR,AT,COMMA
       DATA SEMIS
*
*** [ ***
       DATA L1088
L1089  DATA >C1DB
LBRCKT DATA DOCOL,ZERO,STATE,STORE,SEMIS
*
*** ] ***
       DATA L1089
L108A  DATA >81DD
RBRCKT DATA DOCOL,LIT,>C0,STATE,STORE,SEMIS
*
*** SMUDGE ***
       DATA L108A
L108B  DATA >8653,>4D55,>4447,>45A0
SMUDGE DATA DOCOL,LATEST,LIT,>20,TOGGLE,SEMIS
*
*** HEX ***
       DATA L108B
L108C  DATA >8348,>45D8
HEX    DATA DOCOL,LIT,>10,BASE,STORE,SEMIS
*
*** DECIMAL ***
       DATA L108C
L108D  DATA >8744,>4543,>494D,>41CC
DECIMA DATA DOCOL,LIT,>A,BASE,STORE,SEMIS
*
*** COUNT ***
       DATA L108D
L108E  DATA >8543,>4F55,>4ED4
COUNT  DATA DOCOL,DUP,ONEP,SWAP,CAT,SEMIS
*
*** TYPE ***
       DATA L108E
L108F  DATA >8454,>5950,>45A0
TYPE   DATA DOCOL,DDUP,ZBRAN,L1090-$,ZERO,PDO
L1091  DATA DUP,CAT,EMIT,ONEP,PLOOP,L1091-$
L1090  DATA DROP,SEMIS
*
*** -TRAILING ***
       DATA L108F
L1092  DATA >892D,>5452,>4149,>4C49,>4EC7
DTRAIL DATA DOCOL,DUP,ZERO,PDO
L1093  DATA OVER,OVER,PLUS,ONEM,CAT,BL,SUB,ZBRAN
       DATA L1094-$,LEAVE,BRANCH,L1095-$
L1094  DATA ONEM
```

```
L1095  DATA PLOOP,L1093-$,SEMIS
*
*** ?STACK ***
       DATA L1092
L1096  DATA >863F,>5354,>4143,>4BA0
QSTACK DATA DOCOL,SPAT,S0,AT,SWAP,ULESS,ONE,QERROR
       DATA SPAT,HERE,LIT,>80,PLUS,ULESS
       DATA LIT,>7
       DATA QERROR,SEMIS
*
*** EXPECT ***
       DATA L1096
L1097  DATA >8645,>5850,>4543,>54A0
EXPECT DATA DOCOL,ZERO,PDO
L1098  DATA KEY,DUP,LIT,>D,EQUAL,ZBRAN,L1099-$
       DATA DROP,SPACE,LEAVE,ZERO,BRANCH,L109A-$
L1099  DATA DUP,LIT,>8,EQUAL,ZBRAN,L109B-$,DROP
       DATA I,ZEQU,ZBRAN,L109C-$,LIT,>7,EMIT,ZERO
       DATA BRANCH,L109D-$
L109C  DATA LIT,>8,EMIT,FROMR,ONEM,TOR,ONEM
       DATA ZERO
L109D  DATA BRANCH,L109E-$
L109B  DATA DUP,EMIT,OVER,CSTORE,ONEP,ONE
L109E
L109A  DATA PPLOOP,L1098-$,ZERO,SWAP,OVER,OVER
       DATA CSTORE,ONEP,CSTORE,SEMIS
*
*** QUERY ***
       DATA L1097
L109F  DATA >8551,>5545,>52D9
QUERY  DATA DOCOL,TIB,AT,LIT,>50,EXPECT,ZERO,IN
       DATA STORE,SEMIS
*
*** FILL ***
       DATA L109F
L10A0  DATA >8446,>494C,>4CA0
FILL   DATA DOCOL,SWAP,TOR,OVER,CSTORE,DUP,ONEP
       DATA FROMR,ONEM,CMOVE,SEMIS
*
*** ERASE ***
       DATA L10A0
L10A1  DATA >8545,>5241,>53C5
ERASE  DATA DOCOL,ZERO,FILL,SEMIS
*
*** BLANKS ***
       DATA L10A1
L10A2  DATA >8642,>4C41,>4E4B,>53A0
BLANKS DATA DOCOL,BL,FILL,SEMIS
*
*** HOLD ***
       DATA L10A2
```

```
L10A3  DATA >8448,>4F4C,>44A0
HOLD   DATA DOCOL,LIT,>FFFF,HLD,PSTORE,HLD,AT,CSTORE
       DATA SEMIS
*
*** PAD ***
       DATA L10A3
L10A4  DATA >8350,>41C4
PAD    DATA DOCOL,HERE,LIT,>44,PLUS,SEMIS
*
*** WORD ***
       DATA L10A4
L10A5  DATA >8457,>4F52,>44A0
WORD   DATA DOCOL,BLK,AT,ZBRAN,L10A6-$,BLK,AT,BLOCK
       DATA BRANCH,L10A7-$
L10A6  DATA TIB,AT
L10A7  DATA IN,AT,PLUS,SWAP,ENCLOS,HERE,LIT,>22
       DATA BLANKS,IN,PSTORE,OVER,SUB,DUP,TOR,HERE
       DATA CSTORE,PLUS,HERE,ONEP,FROMR,CMOVE,SEMIS
*
*** (.") ***
       DATA L10A5
L10A8  DATA >8428,>2E22,>29A0
PTYPE  DATA DOCOL,RR,COUNT,DUP,ONEP,ECELLS,FROMR
       DATA PLUS,TOR,TYPE,SEMIS
*
*** ." ***
       DATA L10A8
L10A9  DATA >C22E,>22A0
STRNG  DATA DOCOL,LIT,>22,STATE,AT,ZBRAN,L10AA-$
       DATA COMPIL,PTYPE,WORD,HERE,CAT,ONEP,ECELLS
       DATA ALLOT,BRANCH,L10AB-$
L10AA  DATA WORD,HERE,COUNT,TYPE
L10AB  DATA SEMIS
*
*** (NUMBER) ***
       DATA L10A9
L10AC  DATA >8828,>4E55,>4D42,>4552,>29A0
PNUMBR DATA DOCOL
L10AD  DATA ONEP,DUP,TOR,CAT,BASE,AT,DIGIT,ZBRAN
       DATA L10AE-$,SWAP,BASE,AT,MULT,DROP,ROT
       DATA BASE,AT,MULT,DPLUS,DPL,AT,ONEP,ZBRAN
       DATA L10AF-$,ONE,DPL,PSTORE
L10AF  DATA FROMR,BRANCH,L10AD-$
L10AE  DATA FROMR,SEMIS
*
*** NUMBER ***
       DATA L10AC
L10B0  DATA >864E,>554D,>4245,>52A0
NUMBER DATA DOCOL,ZERO,ZERO,ROT,DUP,ONEP,CAT,LIT
       DATA >2D,EQUAL,DUP,TOR,PLUS,LIT,>FFFF
```

```
L10B1  DATA DPL,STORE,PNUMBR,DUP,CAT,BL,SUB,ZBRAN
       DATA L10B2-$,DUP,CAT,LIT,>2E,SUB,ZERO,QERROR
       DATA ZERO,BRANCH,L10B1-$
L10B2  DATA DROP,FROMR,ZBRAN,L10B3-$,DMINUS
L10B3  DATA SEMIS
*
*** -FIND ***
       DATA L10B0
L10B4  DATA >852D,>4649,>4EC4
DFIND  DATA DOCOL,BL,WORD,HERE,CONTEX,AT,AT,PFIND
       DATA DUP,ZEQU,ZBRAN,L10B5-$,DROP,HERE,LATEST
       DATA PFIND
L10B5  DATA SEMIS
*
*** (ABORT) ***
       DATA L10B4
L10B6  DATA >8728,>4142,>4F52,>54A9
PABORT DATA DOCOL,ABORT,SEMIS
*
*** ERROR ***
       DATA L10B6
L10B7  DATA >8545,>5252,>4FD2
ERROR  DATA DOCOL,WARNIN,AT,ZLESS,ZBRAN,L10B8-$
       DATA PABORT,BRANCH,L10B9-$
L10B8  DATA ECOUNT,AT,ZEQU,ZBRAN,L10BA-$,ONE,ECOUNT
       DATA STORE,HERE,COUNT,TYPE,PTYPE,>420,>203F
       DATA >2020,MESSAG
L10BA
L10B9  DATA ZERO,ECOUNT,STORE,SPSTOR,IN,AT,BLK
       DATA AT,QUIT,SEMIS
*
*** ID. ***
       DATA L10B7
L10BB  DATA >8349,>44AE
IDDOT  DATA DOCOL,PAD,LIT,>20,LIT,>5F,FILL,DUP
       DATA ONE,TRAVER,OVER,SUB,DUP,TOR,ONEP,PAD
       DATA SWAP,CMOVE,PAD,FROMR,PLUS,LIT,>80,TOGGLE
       DATA PAD,COUNT,LIT,>1F,AND,TYPE,SPACE,SEMIS
*
*** CREATE ***
       DATA L10BB
L10BC  DATA >8643,>5245,>4154,>45A0
CREATE DATA DOCOL,HERE,ECELLS,DP,STORE
       DATA LATEST,COMMA,DFIND,ZBRAN,L10BD-$
       DATA DROP,NFA,IDDOT,LIT,>4,MESSAG,SPACE
L10BD  DATA HERE,DUP,CAT,WIDTH,AT,MIN,ONEP,ECELLS
       DATA ALLOT,DUP,LIT,>A0,TOGGLE,HERE,ONEM
       DATA LIT,>80,TOGGLE,CURREN,AT,STORE,HERE
       DATA TWOP,COMMA,SEMIS
*
*** [COMPILE] ***
```

```
       DATA L10BC
L10BE  DATA >C95B,>434F,>4D50,>494C,>45DD
BCOMPI DATA DOCOL,DFIND,ZEQU,ZERO,QERROR,DROP,CFA
       DATA COMMA,SEMIS
*
*** LITERAL ***
       DATA L10BE
L10BF  DATA >C74C,>4954,>4552,>41CC
LITERA DATA DOCOL,STATE,AT,ZBRAN,L10C0-$,COMPIL
       DATA LIT,COMMA
L10C0  DATA SEMIS
*
*** DLITERAL ***
       DATA L10BF
L10C1  DATA >C844,>4C49,>5445,>5241,>4CA0
DLITER DATA DOCOL,STATE,AT,ZBRAN,L10C2-$,SWAP,LITERA
       DATA LITERA
L10C2  DATA SEMIS
*
*** INTERPRET ***
       DATA L10C1
L10C3  DATA >8949,>4E54,>4552,>5052,>45D4
INTERP DATA DOCOL
L10C4  DATA DFIND,ZBRAN,L10C5-$,STATE,AT,LESS,ZBRAN
       DATA L10C6-$,CFA,COMMA,BRANCH,L10C7-$
L10C6  DATA CFA,EXECUT
L10C7  DATA QSTACK,BRANCH,L10C8-$
L10C5  DATA HERE,NUMBER,DPL,AT,ONEP,ZBRAN,L10C9-$
       DATA DLITER,BRANCH,L10CA-$
L10C9  DATA DROP,LITERA
L10CA  DATA QSTACK
L10C8  DATA BRANCH,L10C4-$,SEMIS
*
*** IMMEDIATE ***
       DATA L10C3
L10CB  DATA >8949,>4D4D,>4544,>4941,>54C5
IMMEDI DATA DOCOL,LATEST,LIT,>40,TOGGLE,SEMIS
*
*** ( ***
       DATA L10CB
L10CC  DATA >C1A8
PAREN  DATA DOCOL,LIT,>29,WORD,SEMIS
*
*** FORTH ***
       DATA L10CC
L10CD  DATA >C546,>4F52,>54C8
FORTH  DATA DOCOL,FORTHL,LIT,>4,SUB,CONTEX,STORE
       DATA SEMIS
*
*** DEFINITIONS ***
```

Disk 9. Contents of File ASMSRC3

```
       DATA L10CD
L10CE  DATA >8B44,>4546,>494E,>4954,>494F,>4ED3
DEFINI DATA DOCOL,CONTEX,AT,CURREN,STORE,SEMIS
*
*** QUIT ***
       DATA L10CE
L10CF  DATA >8451,>5549,>54A0
QUIT   DATA DOCOL,ZERO,BLK,STORE,LBRCKT
L10D0  DATA RSTOR,CR,QUERY,INTERP,STATE,AT,ZEQU
       DATA ZBRAN,L10D1-$,PTYPE,>320,>6F6B
L10D1  DATA BRANCH,L10D0-$,SEMIS
*
*** ABORT ***
       DATA L10CF
L10D2  DATA >8541,>424F,>52D4
ABORT  DATA DOCOL,SPSTOR,DECIMA,ZERO,ECOUNT,STORE
       DATA CR,PTYPE,>854,>4920
       DATA >464F,>5254,>4820,FORTH,DEFINI,QUIT
       DATA SEMIS
*
*** +- ***
       DATA L10D2
L10D3  DATA >822B,>2DA0
PM     DATA DOCOL,ZLESS,ZBRAN,L10D4-$,MINUS
L10D4  DATA SEMIS
*
*** D+- ***
       DATA L10D3
L10D5  DATA >8344,>2BAD
DPM    DATA DOCOL,ZLESS,ZBRAN,L10D6-$,DMINUS
L10D6  DATA SEMIS
*
*** DABS ***
       DATA L10D5
L10D7  DATA >8444,>4142,>53A0
DABS   DATA DOCOL,DUP,DPM,SEMIS
*
*** M* ***
       DATA L10D7
L10D8  DATA >824D,>2AA0
MSTAR  DATA DOCOL,OVER,OVER,XOR,TOR,ABS,SWAP,ABS
       DATA MULT,FROMR,DPM,SEMIS
*
*** M/ ***
       DATA L10D8
L10D9  DATA >824D,>2FA0
MSLASH DATA DOCOL,OVER,TOR,TOR,DABS,RR,ABS,DIV
       DATA FROMR,RR,XOR,PM,SWAP,FROMR,PM,SWAP
```

```
       DATA SEMIS
*
*** * ***
       DATA L10D9
L10DA  DATA >81AA
TIMES  DATA DOCOL,MULT,DROP,SEMIS
*
*** /MOD ***
       DATA L10DA
L10DB  DATA >842F,>4D4F,>44A0
DMOD   DATA DOCOL,TOR,STOD,FROMR,MSLASH,SEMIS
*
*** / ***
       DATA L10DB
L10DC  DATA >81AF
DDIV   DATA DOCOL,DMOD,SWAP,DROP,SEMIS
*
*** MOD ***
       DATA L10DC
L10DD  DATA >834D,>4FC4
MOD    DATA DOCOL,DMOD,DROP,SEMIS
*
*** */MOD ***
       DATA L10DD
L10DE  DATA >852A,>2F4D,>4FC4
MDMOD  DATA DOCOL,TOR,MSTAR,FROMR,MSLASH,SEMIS
*
*** */ ***
       DATA L10DE
L10DF  DATA >822A,>2FA0
MD     DATA DOCOL,MDMOD,SWAP,DROP,SEMIS
*
*** M/MOD ***
       DATA L10DF
L10E0  DATA >854D,>2F4D,>4FC4
MSLMOD DATA DOCOL,TOR,ZERO,RR,DIV,FROMR,SWAP,TOR
       DATA DIV,FROMR,SEMIS
*
*** SPACES ***
       DATA L10E0
L10E1  DATA >8653,>5041,>4345,>53A0
SPACES DATA DOCOL,ZERO,MAX,DDUP,ZBRAN,L10E2-$,ZERO
       DATA PDO
L10E3  DATA SPACE,PLOOP,L10E3-$
L10E2  DATA SEMIS
*
*** <# ***
       DATA L10E1
L10E4  DATA >823C,>23A0
STRTCN DATA DOCOL,PAD,HLD,STORE,SEMIS
```

```
*
*** #> ***
       DATA L10E4
L10E5  DATA >8223,>3EA0
STOPCN DATA DOCOL,DROP,DROP,HLD,AT,PAD,OVER,SUB
       DATA SEMIS
*
*** SIGN ***
       DATA L10E5
L10E6  DATA >8453,>4947,>4EA0
SIGN   DATA DOCOL,ROT,ZLESS,ZBRAN,L10E7-$,LIT,>2D
       DATA HOLD
L10E7  DATA SEMIS
*
*** # ***
       DATA L10E6
L10E8  DATA >81A3
NUMSGN DATA DOCOL,PAD,HLD,AT,SUB,DPL,AT,EQUAL,ZBRAN
       DATA L10E9-$,LIT,>2E,HOLD
L10E9  DATA BASE,AT,MSLMOD,ROT,LIT,>9,OVER,LESS
       DATA ZBRAN,L10EA-$,LIT,>7,PLUS
L10EA  DATA LIT,>30,PLUS,HOLD,SEMIS
*
*** #S ***
       DATA L10E8
L10EB  DATA >8223,>53A0
NUMS   DATA DOCOL
L10EC  DATA NUMSGN,OVER,OVER,OR,ZEQU,ZBRAN,L10EC-$
       DATA SEMIS
*
*** D.R ***
       DATA L10EB
L10ED  DATA >8344,>2ED2
DDOTR  DATA DOCOL,TOR,SWAP,OVER,DABS,STRTCN,NUMS
       DATA SIGN,STOPCN,FROMR,OVER,SUB,SPACES,TYPE
       DATA SEMIS
*
*** D. ***
       DATA L10ED
L10EE  DATA >8244,>2EA0
DDOT   DATA DOCOL,ZERO,DDOTR,SPACE,SEMIS
*
*** .R ***
       DATA L10EE
L10EF  DATA >822E,>52A0
DOTR   DATA DOCOL,TOR,STOD,FROMR,DDOTR,SEMIS
*
*** . ***
       DATA L10EF
L10F0  DATA >81AE
DOT    DATA DOCOL,STOD,DDOT,SEMIS
```

```
*
*** ? ***
       DATA L10F0
L10F1  DATA >81BF
QMARK  DATA DOCOL,AT,DOT,SEMIS
*
*** UD.R ***
       DATA L10F1
L10F2  DATA >8455,>442E,>52A0
UDDOTR DATA DOCOL,TOR,STRTCN,NUMS,STOPCN,FROMR
       DATA OVER,SUB,SPACES,TYPE,SEMIS
*
*** UD. ***
       DATA L10F2
L10F3  DATA >8355,>44AE
UDDOT  DATA DOCOL,ZERO,UDDOTR,SPACE,SEMIS
*
*** U.R ***
       DATA L10F3
L10F4  DATA >8355,>2ED2
UDOTR  DATA DOCOL,TOR,ZERO,FROMR,UDDOTR,SEMIS
*
*** U. ***
       DATA L10F4
L10F5  DATA >8255,>2EA0
UDOT   DATA DOCOL,ZERO,UDDOT,SEMIS
*
*** +BUF ***
       DATA L10F5
L10F6  DATA >842B,>4255,>46A0
PLSBUF DATA DOCOL,BSLBUF,LIT,>4,PLUS,PLUS,DUP,LIMIT
       DATA EQUAL,ZBRAN,L10F7-$,DROP,FIRST
L10F7  DATA DUP,PREV,AT,SUB,SEMIS
*
*** BUFFER ***
       DATA L10F6
L10F8  DATA >8642,>5546,>4645,>52A0
BUFFER DATA DOCOL,USE,AT,DUP,TOR
L10F9  DATA PLSBUF,ZBRAN,L10F9-$,USE,STORE,RR,AT
       DATA ZLESS,ZBRAN,L10FA-$,RR,TWOP,RR,AT,LIT
       DATA >7FFF,AND,ZERO,RSLW
L10FA  DATA RR,STORE,RR,PREV,STORE,FROMR,TWOP,SEMIS
*
*** UPDATE ***
       DATA L10F8
L10FB  DATA >8655,>5044,>4154,>45A0
UPDATE DATA DOCOL,PREV,AT,AT,LIT,>8000,OR,PREV
       DATA AT,STORE,SEMIS
*
*** FLUSH ***
```

```
       DATA L10FB
L10FC  DATA >8546,>4C55,>53C8
FLUSH  DATA DOCOL,LIMIT,FIRST,SUB,BSLBUF,LIT,>4
       DATA PLUS,DDIV,ONEP,ZERO,PDO
L10FD  DATA LIT,>7FFF,BUFFER,DROP,PLOOP,L10FD-$
       DATA SEMIS
*
*** EMPTY-BUFFERS ***
       DATA L10FC
L10FE  DATA >8D45,>4D50,>5459,>2D42,>5546,>4645
       DATA >52D3
EMPTYB DATA DOCOL,FIRST,LIMIT,OVER,SUB,ERASE,FLUSH
       DATA FIRST,USE,STORE,FIRST,PREV,STORE,SEMIS
*
*** CLEAR ***
       DATA L10FE
L10FF  DATA >8543,>4C45,>41D2
CLEAR  DATA DOCOL,DUP,SCR,STORE,BSLSCR,TIMES,OFFSET
       DATA AT,PLUS,FLUSH,BSLSCR,ZERO,PDO
L1100  DATA I,OVER,PLUS,BUFFER,BSLBUF,BLANKS,UPDATE
       DATA PLOOP,L1100-$,DROP,SEMIS
*
*** BLOCK ***
       DATA L10FF
L1101  DATA >8542,>4C4F,>43CB
BLOCK  DATA DOCOL,OFFSET,AT,PLUS,TOR,PREV,AT,DUP
       DATA AT,RR,SUB,DUP,PLUS,ZBRAN,L1102-$
L1103  DATA PLSBUF,ZEQU,ZBRAN,L1104-$,DROP,RR,BUFFER
       DATA DUP,RR,ONE,RSLW,TWOM
L1104  DATA DUP,AT,RR,SUB,DUP,PLUS,ZEQU,ZBRAN
       DATA L1103-$,DUP,PREV,STORE
L1102  DATA FROMR,DROP,TWOP,SEMIS
*
*** (LINE) ***
       DATA L1101
L1105  DATA >8628,>4C49,>4E45,>29A0
PLINE  DATA DOCOL,TOR,CSL,BSLBUF,MDMOD,FROMR,BSLSCR
       DATA TIMES,PLUS,BLOCK,PLUS,CSL,SEMIS
*
*** .LINE ***
       DATA L1105
L1106  DATA >852E,>4C49,>4EC5
DOTLN  DATA DOCOL,PLINE,DTRAIL,TYPE,SEMIS
*
*** MESSAGE ***
       DATA L1106
L1107  DATA >874D,>4553,>5341,>47C5
MESSAG DATA DOCOL,WARNIN,AT,ZBRAN,L1108-$,DDUP
       DATA ZBRAN,L1109-$,LIT,>4,OFFSET,AT,BSLSCR
       DATA DDIV,SUB,DOTLN
L1109  DATA BRANCH,L110A-$
```

```
L1108  DATA PTYPE,>64D,>5347,>2023,>2020,DOT
L110A  DATA SEMIS
*
*** LOAD ***
       DATA L1107
L110B  DATA >844C,>4F41,>44A0
LOAD   DATA DOCOL,DDUP,ZEQU,LIT,>C,QERROR,BLK,AT
       DATA TOR,IN,AT,TOR,ZERO,IN
       DATA STORE,BSLSCR,TIMES,BLK,STORE,INTERP
       DATA FROMR,IN,STORE,FROMR
       DATA BLK,STORE,SEMIS
*
*** --> ***
       DATA L110B
L110C  DATA >C32D,>2DBE
ARROW  DATA DOCOL,QLOADI,ZERO,IN,STORE,BSLSCR
       DATA BLK,AT,OVER,MOD,SUB
       DATA BLK,PSTORE,SEMIS
*
*** R/W ***
       DATA L110C
L110D  DATA >8352,>2FD7
RSLW   DATA DOCOL,BSLBUF,SWAP,ZBRAN,L110E-$,RDISK
       DATA BRANCH,L110F-$
L110E  DATA WDISK
L110F  DATA DUP,QERROR,SEMIS
*
*** ' ***
       DATA L110D
L1110  DATA >C1A7
TICK   DATA DOCOL,DFIND,ZEQU,ZERO,QERROR,DROP,LITERA
       DATA SEMIS
*
*** UNFORGETABLE ***
       DATA L1110
L1110X DATA >8C55,>4E46,>4F52,>4745,>5441,>424C,>45A0
UNFORG DATA DOCOL,DUP,FENCE,AT,ULESS,OVER,LIT,$TASK1
       DATA ULESS,OR,HERE,ROT,ULESS,OR,SEMIS
*
*** FORGET ***
       DATA L1110X
L1111  DATA >8646,>4F52,>4745,>54A0
FORGET DATA DOCOL,TICK,LFA,DUP,UNFORG,LIT,>15,QERROR
       DATA TOR,VLINK,AT
FORGE1 DATA RR,OVER,ULESS,OVER,UNFORG,ZEQU,AND
       DATA ZBRAN,FORGE2-$,FORTH,DEFINI,AT
       DATA BRANCH,FORGE1-$
FORGE2 DATA DUP,VLINK,STORE
FORGE3 DATA DUP,TWOM
FORGE4 DATA PFA,LFA,AT,DUP,PFA,LFA,RR,ULESS,OVER
```

```
       DATA UNFORG,OR,ZBRAN,FORGE4-$
       DATA OVER,LIT,>4,SUB,STORE,AT,DDUP,ZEQU
       DATA ZBRAN,FORGE3-$,FROMR,DP,STORE,SEMIS
*
*** : ***
       DATA L1111
L1112  DATA >C1BA
COLON  DATA DOCOL,QEXEC,STRCSP,CURREN,AT,CONTEX
       DATA STORE,CREATE,RBRCKT,LIT,DOCOL
       DATA HERE,TWOM,STORE,SEMIS
*
*** ; ***
       DATA L1112
L1113  DATA >C1BB
SEMIC  DATA DOCOL,QCSP,COMPIL,SEMIS,SMUDGE,LBRCKT
       DATA SEMIS
*
*** BACK ***
       DATA L1113
L1114  DATA >8442,>4143,>4BA0
BACK   DATA DOCOL,HERE,SUB,COMMA,SEMIS
*
*** BEGIN ***
       DATA L1114
L1115  DATA >C542,>4547,>49CE
BEGIN  DATA DOCOL,QCOMP,HERE,ONE,SEMIS
*
*** ENDIF ***
       DATA L1115
L1116  DATA >C545,>4E44,>49C6
ENDIF  DATA DOCOL,QCOMP,TWO,QPAIRS,HERE,OVER,SUB
       DATA SWAP,STORE,SEMIS
*
*** THEN ***
       DATA L1116
L1117  DATA >C454,>4845,>4EA0
THEN   DATA DOCOL,ENDIF,SEMIS
*
*** DO ***
       DATA L1117
L1118  DATA >C244,>4FA0
DO     DATA DOCOL,QCOMP,COMPIL,PDO,HERE,THREE,SEMIS
*
*** LOOP ***
       DATA L1118
L1119  DATA >C44C,>4F4F,>50A0
LOOP   DATA DOCOL,QCOMP,THREE,QPAIRS,COMPIL,PLOOP
       DATA BACK,SEMIS
*
*** +LOOP ***
       DATA L1119
```

```
L111A  DATA >C52B,>4C4F,>4FD0
PLLOOP DATA DOCOL,QCOMP,THREE,QPAIRS,COMPIL,PPLOOP
       DATA BACK,SEMIS
*
*** UNTIL ***
       DATA L111A
L111B  DATA >C555,>4E54,>49CC
UNTIL  DATA DOCOL,QCOMP,ONE,QPAIRS,COMPIL,ZBRAN
       DATA BACK,SEMIS
*
*** END ***
       DATA L111B
L111C  DATA >C345,>4EC4
END    DATA DOCOL,UNTIL,SEMIS
*
*** AGAIN ***
       DATA L111C
L111D  DATA >C541,>4741,>49CE
AGAIN  DATA DOCOL,QCOMP,ONE,QPAIRS,COMPIL,BRANCH
       DATA BACK,SEMIS
*
*** REPEAT ***
       DATA L111D
L111E  DATA >C652,>4550,>4541,>54A0
REPEAT DATA DOCOL,QCOMP,TOR,TOR,AGAIN,FROMR,FROMR
       DATA TWOM,ENDIF,SEMIS
*
*** IF ***
       DATA L111E
L111F  DATA >C249,>46A0
IF     DATA DOCOL,QCOMP,COMPIL,ZBRAN,HERE,ZERO
       DATA COMMA,TWO,SEMIS
*
*** ELSE ***
       DATA L111F
L1120  DATA >C445,>4C53,>45A0
ELSE   DATA DOCOL,QCOMP,TWO,QPAIRS,COMPIL,BRANCH
       DATA HERE,ZERO,COMMA,SWAP,TWO,ENDIF,TWO
       DATA SEMIS
*
*** WHILE ***
       DATA L1120
L1121  DATA >C557,>4849,>4CC5
WHILE  DATA DOCOL,IF,TWOP,SEMIS
*
*** CASE ***
       DATA L1121
L1122  DATA >C443,>4153,>45A0
CASE   DATA DOCOL,QCOMP,CSP,AT,STRCSP,LIT,>4,SEMIS
*
```

```
*** OF ***
       DATA L1122
L1123  DATA >C24F,>46A0
OF     DATA DOCOL,LIT,>4,QPAIRS,COMPIL,POF,HERE
       DATA ZERO,COMMA,LIT,>5,SEMIS
*
*** ENDOF ***
       DATA L1123
L1124  DATA >C545,>4E44,>4FC6
ENDOF  DATA DOCOL,LIT,>5,QPAIRS,COMPIL,BRANCH,HERE
       DATA ZERO,COMMA,SWAP,TWO,ENDIF,LIT,>4,SEMIS
*
*** ENDCASE ***
       DATA L1124
L1125  DATA >C745,>4E44,>4341,>53C5
ENDCAS DATA DOCOL,LIT,>4,QPAIRS,COMPIL,DROP
L1126  DATA SPAT,CSP,AT,EQUAL,ZEQU,ZBRAN,L1127-$
       DATA TWO,ENDIF,BRANCH,L1126-$
L1127  DATA CSP,STORE,SEMIS
*
*** BASE->R ***
       DATA L1125
L1128  DATA >8742,>4153,>452D,>3ED2
BASTOR DATA DOCOL,FROMR,BASE,AT,TOR,TOR,SEMIS
*
*** R->BASE ***
       DATA L1128
L1129  DATA >8752,>2D3E,>4241,>53C5
RTOBAS DATA DOCOL,FROMR,FROMR,BASE,STORE,TOR,SEMIS
*
*** L/SCR ***
       DATA L1129
L112A  DATA >854C,>2F53,>43D2
LPSCR  DATA DOCOL,BSLSCR,BSLBUF,TIMES,CSL,DDIV
       DATA SEMIS
*
*** PAUSE ***
       DATA L112A
L112AX DATA >8550,>4155,>53C5
PAUSE  DATA DOCOL,QKEY,DUP,TWO,EQUAL
       DATA ZBRAN,PAUSE1-$,DROP,ONE,BRANCH,PAUSE2-$
PAUSE1 DATA ZBRAN,PAUSE3-$
PAUSE4 DATA QKEY,ZEQU,ZBRAN,PAUSE4-$
PAUSE5 DATA QKEY,DDUP,ZBRAN,PAUSE5-$
       DATA TWO,EQUAL,ZBRAN,PAUSE6-$
       DATA ONE,BRANCH,PAUSE7-$
PAUSE6 DATA QKEY,ZEQU,ZBRAN,PAUSE6-$,ZERO
PAUSE7 DATA BRANCH,PAUSE2-$
PAUSE3 DATA ZERO
PAUSE2 DATA SEMIS
*
```

```
*** LIST ***
       DATA L112AX
L112B  DATA >844C,>4953,>54A0
LIST   DATA DOCOL,BASTOR,DECIMA,CR,DUP,SCR,STORE
       DATA PTYPE,>553,>4352,>2023,DOT,LPSCR,ZERO
       DATA PDO
L112C  DATA CR,I,THREE,DOTR,SPACE,I,SCR,AT,DOTLN
       DATA PAUSE,ZBRAN,L112CX-$,LEAVE
L112CX DATA PLOOP,L112C-$,CR,RTOBAS,SEMIS
*
*** <BUILDS ***
       DATA L112B
L1139  DATA >873C,>4255,>494C,>44D3
BUILDS DATA DOCOL,CREATE,SMUDGE,SEMIS
*
*** (DOES>) ***
       DATA L1139
L113A  DATA >8728,>444F,>4553,>3EA9
PDOES  DATA DOCOL,FROMR,LATEST,PFA,CFA,STORE,SEMIS
*
*** DOES> ***
       DATA L113A
L113B  DATA >C544,>4F45,>53BE
DOES   DATA DOCOL,LIT,PDOES,COMMA,LIT,>6A0,COMMA
       DATA LIT,DODOES,COMMA,SEMIS
*
*** CONSTANT ***
       DATA L113B
L113C  DATA >8843,>4F4E,>5354,>414E,>54A0
CONSTA DATA DOCOL,BUILDS,COMMA
DOCON  EQU  $+2
       DATA PDOES,>6A0,DODOES,AT,SEMIS
*
*** USER ***
       DATA L113C
L113D  DATA >8455,>5345,>52A0
USER   DATA DOCOL,BUILDS,COMMA
DOUSER EQU  $+2
       DATA PDOES,>6A0,DODOES,AT,UU,PLUS,SEMIS
*
*** VARIABLE ***
       DATA L113D
L113E  DATA >8856,>4152,>4941,>424C,>45A0
VARIAB DATA DOCOL,BUILDS,COMMA
DOVAR  EQU  $+2
       DATA PDOES,>6A0,DODOES,SEMIS
*
*** VOCABULARY ***
       DATA L113E
L113F  DATA >8A56,>4F43,>4142,>554C,>4152,>59A0
```

```
VOCABU DATA DOCOL,BUILDS,CURREN,AT,TWOP,COMMA,LIT
       DATA >81A0,COMMA,HERE,VLINK,AT,COMMA
       DATA VLINK,STORE
DOVOC  EQU  $+2
       DATA PDOES,>6A0,DODOES,CONTEX,STORE,SEMIS
*
*** (;CODE) ***
       DATA L113F
L1140  DATA >8728,>3B43,>4F44,>45A9
PSCODE DATA DOCOL,FROMR,LATEST,PFA,CFA,STORE,SEMIS
*
*** MYSELF ***
       DATA L1140
L1144  DATA >C64D,>5953,>454C,>46A0
MYSELF DATA DOCOL,LATEST,PFA,CFA,COMMA,SEMIS
*
*** ~ ***
       DATA L1144
L1145  DATA >C180
NULL   DATA DOCOL,BLK,AT,ZBRAN,L1146-$,ONE,BLK
       DATA PSTORE,ZERO,IN,STORE,BLK,AT,BSLSCR
       DATA MOD,ZEQU,ZBRAN,L1147-$,QEXEC,FROMR
       DATA DROP
L1147  DATA BRANCH,L1148-$
L1146  DATA FROMR,DROP
L1148  DATA SEMIS
*
*** NOP ***
       DATA L1145
L1166  DATA >834E,>4FD0
NOP    DATA DOCOL,SEMIS
*
*** BLOAD ***
       DATA L1166
L1166X DATA >8542,>4C4F,>41C4
BLOAD  DATA DOCOL
BLOAD1 DATA DUP,ONEP,SWAP,BLOCK
       DATA DUP,LIT,14,PLUS,AT,LIT,29801,EQUAL
       DATA ZBRAN,BLOAD2-$,DUP,AT,TOR
       DATA TWOP,DUP,AT,DUP,TOR,DP,STORE
       DATA TWOP,DUP,AT,CURREN,STORE
       DATA TWOP,DUP,AT,CURREN,AT,STORE
       DATA TWOP,DUP,AT,CONTEX,STORE
       DATA TWOP,DUP,AT,CONTEX,AT,STORE
       DATA TWOP,DUP,AT,VLINK,STORE
       DATA LIT,12,PLUS,FROMR,FROMR,SWAP
       DATA OVER,SUB,DUP,TOR,LIT,1000,MIN
       DATA CMOVE,FROMR,LIT,1001,LESS,BRANCH,BLOAD3-$
BLOAD2 DATA DROP,DROP,ZERO,ONE
BLOAD3 DATA ZBRAN,BLOAD1-$,ZEQU,SEMIS
*
```

```
*** COLD ***
       DATA L1166X
L1167  DATA >8443,>4F4C,>44A0
COLD   DATA DOCOL,UCONS$,AT,U0,AT,LIT,ULNGTH,CMOVE
       DATA LIT,$TASK0,LIT,$TASK1,OVER,SUB,TOR
       DATA HERE,RR,CMOVE,HERE,TWOP,DUP,FORTHL
       DATA LIT,>4,SUB,STORE,FENCE,STORE,LIT,>81A0
       DATA FORTHL,TWOM,STORE,ZERO,FORTHL,STORE
       DATA FORTHL,VLINK,STORE
       DATA FIRST,USE,STORE,FIRST,PREV,STORE,FROMR
       DATA ALLOT,DR0,EMPTYB,LIT,>FFFF,DPL,STORE
       DATA BOOT,ABORT,SEMIS
*
*** BOOT ***
       DATA L1167
BOOTN  DATA >8442,>4F4F,>54A0
BOOT   DATA DOCOL,SPSTOR,DECIMA,ZERO,ECOUNT
       DATA STORE,FORTH,DEFINI,ZERO,BLK,STORE,LBRCKT
       DATA ZERO,THREE,BLOCK,DUP,LIT,>400,PLUS
       DATA SWAP,PDO
BOOT1  DATA I,CAT,DUP,LIT,>20,LESS,SWAP
       DATA LIT,>7F,GREAT,OR,ZBRAN,BOOT2-$
       DATA ONEP,LEAVE
BOOT2  DATA PLOOP,BOOT1-$,ZEQU,ZBRAN,BOOT3-$
       DATA THREE,LOAD
BOOT3  DATA SEMIS
*
*** SYSTEM ***
       DATA BOOTN
L1168  DATA >8653,>5953,>5445,>4DA0
SYST$  DATA $+2
       MOV  *SP+,TEMP1
       MOV  @$SYS(U),LINK
       BL   *LINK
       B    *NEXT
$TASK0 EQU  $
*
*** TASK ***
       DATA L1168
L1169  DATA >8454,>4153,>4BA0
TASK   DATA DOCOL,SEMIS

$TASK1 EQU  $
*
```

Disk 10. TI Forth Source Code — Disk 2

Version: *Author*:
Requires: *Language*: *Updated*:

See disk 9

```
dskdir. v2.0. 12-dec-96
Disk name               = 08DEC82
Sectors total           = 360
Sectors used            = 180
Sectors available       = 178
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BOOT         21 DIS/VAR 80   >022 020
002 >003 BOOTOBJ       6 DIS/FIX 80   >036 005
003 >004 DRIVER       98 DIS/VAR 80   >03b 097
004 >005 UTILEQU       3 DIS/VAR 80   >09c 002
005 >006 UTILRAM       4 DIS/VAR 80   >09e 003
006 >007 UTILROM      48 DIS/VAR 80   >0a1 047
```

Disk 10. Contents of File BOOT

```
       TITL 'FORTH BOOT PROGRAM'
       IDT  'BOOT'
*************************************************************
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*************************************************************
MAINWS EQU  >8300        IN CONSOLE CPU RAM
SUBPTR EQU  >8356        POINTS TO SUBROUTINE NAMES IN VDP
DSKERR EQU  >8350        DISK DSR ERROR CODES HERE
*************************************************************
       DEF  BOOT
       REF  VWTR,VDPWA,VDPWD
       REF  VMBW,DSRLNK,VMBR
*
FF9900 EQU  >A000
VSPTR  EQU  >836E
KYSTAT EQU  >837C
FAC    EQU  >834A
GRMWA  EQU  >9C02
GRMRA  EQU  >9802
GRMRD  EQU  >9800
SVGPRT EQU  >3C70
GRMSAV EQU  >3CD6
RTFGPL EQU  >3AB0
UBASE0 EQU  >3944
ENTLNK EQU  >3A2A
GPLLNK EQU  >3A06
XMLTAB EQU  >0CFA
CIFENT EQU  >3B32
*************************************************************
PAB    EQU  >F80
PDATA  DATA >0500,>1000,0,>8B6+>1C72,>000E
       TEXT 'DSK1.FORTHSAVE'
       EVEN
```

```
************************************************************
       DORG >832E
DODOES DECT SP           DUMMY COPY TO GET ADDRESSES
       MOV  W,*SP
       MOV  LINK,W
DOCOL  DECT R
       MOV  IP,*R
       MOV  W,IP
$NEXT  MOV  *IP+,W
DOEXEC MOV  *W+,TEMP1
       B    *TEMP1
$SEMIS MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
*
       AORG >C000
FMOVE  DECT SP           COPY TO MOVE TO CONSOLE RAM
       MOV  W,*SP
       MOV  LINK,W
       DECT R
       MOV  IP,*R
       MOV  W,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
       MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
BOOT   LWPI MAINWS
       LIMI 0
       LI   TEMP0,PAB
       LI   TEMP1,PDATA
       LI   TEMP2,>20
       BLWP @VMBW
       LI   TEMP6,PAB+9
       MOV  TEMP6,@SUBPTR
       BLWP @DSRLNK
       DATA 8
       LI   TEMP0,>1000
       LI   TEMP1,>3424
       LI   TEMP2,>8B6
       BLWP @VMBR
       LI   TEMP0,>18B6
       LI   TEMP1,>A000
       LI   TEMP2,>1C72
       BLWP @VMBR
       MOVB @GRMRA,TEMP1
```

```
       SWPB TEMP1
       MOVB @GRMRA,TEMP1
       SWPB TEMP1
       AI   TEMP1,-3
       MOV  TEMP1,@GRMSAV
       INC  TEMP1
       MOVB TEMP1,@GRMWA
       SWPB TEMP1
       MOVB TEMP1,@GRMWA
       NOP
       MOVB @GRMRD,TEMP1
       MOV  TEMP1,TEMP2
       SRL  TEMP1,12
       SLA  TEMP1,1
       SLA  TEMP2,4
       SRL  TEMP2,11
       A    @XMLTAB(TEMP1),TEMP2
       MOV  @>2030,@SVGPRT       >2030 IS SVGPRT USED BY E/A LOADER
       LI   TEMP1,RTFGPL
       MOV  TEMP1,*TEMP2
       LI   TEMP1,BOOT-FMOVE
       LI   TEMP2,FMOVE
       LI   TEMP3,DODOES
MLOOP  MOV  *TEMP2+,*TEMP3+
       DECT TEMP1
       JNE  MLOOP
*
*** INITIALIZE VDP STUFF
*
       LI   TEMP0,>01B0  BLANK SCREEN
       BLWP @VWTR
       LI   TEMP0,>030E  SET COLOR TABLE AT >0380
       BLWP @VWTR
       LI   TEMP0,>0401  SET PATTERN DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>0506  SET SPRITE ATTRIBUTE TABLE >0300
       BLWP @VWTR
       LI   TEMP0,>0601  SET SPRITE DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>07F4  SET TEXTMODE COLORS
       BLWP @VWTR
       LI   TEMP0,>2000  BLANK
       LI   TEMP1,>960   TEXT-MODE SCREEN SIZE
       LI   TEMP2,>0     SCREEN STARTS AT 0
       BL   @FILLER      CLEAR SCREEN
       LI   TEMP0,>FF00  CHAR FF
       LI   TEMP1,>2048  BLOCK SIZE
       LI   TEMP2,>800   STARTING LOCATION IN VDP
       BL   @FILLER      FILL AREA WITH FF'S
       LI   TEMP0,>81F0
```

```
       SWPB TEMP0
       MOVB TEMP0,@>83D4 USED TO UPDATE VDP REG EACH KEYSTROKE
       MOVB TEMP0,@VDPWA FORCE TEXT MODE
       SWPB TEMP0
       MOVB TEMP0,@VDPWA
       LI   TEMP0,>900   VDP LOCATION
       MOV  TEMP0,@FAC
       CLR  TEMP1        CLEAR GPL STATUS
       MOVB TEMP1,@KYSTAT
       LI   TEMP7,>3E0
       MOV  TEMP7,@VSPTR
       BLWP @GPLLNK      LOAD CAPITAL LETTER SHAPES
       DATA >0018
       LI   TEMP2,CIFENT
       MOV  TEMP2,@>2006
       LI   TEMP2,>1200
       CB   TEMP2,@3     IF BYTE @3 IN THE CONSOLE IS >12 IT'S A 99/4
       JEQ  FOUR         DON'T LOAD LOWER CASE IN A 99/4
       LI   TEMP0,>0B00
       MOV  TEMP0,@FAC
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD LOWER CASE IN 99/4A
       DATA >004A
*
FOUR   LI   TEMP1,UBASE0
       B    @FF9900      BRANCH TO ACMSRC
*
FILLER ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLLOOP MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLLOOP       NOT DONE, FILL ANOTHER
       B    *LINK
*
       END  BOOT
```

Disk 10. Contents of File DRIVER

```
       TITL 'FORTH DRIVER WITH UTIL'
       IDT  'FORTH'
*************************************************************
TEMP0  EQU  0
TEMP1  EQU  1
TEMP2  EQU  2
TEMP3  EQU  3
TEMP4  EQU  4
TEMP5  EQU  5
TEMP6  EQU  6
TEMP7  EQU  7
U      EQU  8
SP     EQU  9
W      EQU  10
LINK   EQU  11
CRU    EQU  12
IP     EQU  13
R      EQU  14
NEXT   EQU  15
*************************************************************
MAINWS EQU  >8300        IN CONSOLE CPU RAM
KYSTAT EQU  >837C
KYCHAR EQU  >8375
FAC    EQU  >834A        FLOATING POINT ACCUMULATOR
SUBPTR EQU  >8356        POINTS TO SUBROUTINE NAMES IN VDP
VSPTR  EQU  >836E
DSKERR EQU  >8350        DISK DSR ERROR CODES HERE
*************************************************************
       DEF  FORTH
       REF  KEY,SEMIS
*
FF9900 EQU  >A000
VDPSTA EQU  >8802
GRMWA  EQU  >9C02
GRMRA  EQU  >9802
*************************************************************
       DORG >832E
DODOES DECT SP           DUMMY COPY TO GET ADRESSES
       MOV  W,*SP
       MOV  LINK,W
DOCOL  DECT R
       MOV  IP,*R
       MOV  W,IP
$NEXT  MOV  *IP+,W
DOEXEC MOV  *W+,TEMP1
       B    *TEMP1
$SEMIS MOV  *R+,IP
       MOV  *IP+,W
```

```
       MOV  *W+,TEMP1
       B    *TEMP1
       DORG 0
UBASE  BSS  6            BASE OF USER VARIABLES
$UCONS BSS  2            06 USER UCONS
$S0    BSS  2            08 USER S0
$R0    BSS  2            0A USER R0 { R0$
$U0    BSS  2            0C USER U0
       BSS  2            0E USER TIB
       BSS  2            10 USER WIDTH
       BSS  2            12 USER DP
$SYS   BSS  2            14 USER SYS$
CURPO$ BSS  2            16 USER CURPOS
       BSS  2            18 USER INTLNK
       BSS  2            1A USER WARNING
       BSS  2            1C USER C/L$ { CL$
       BSS  2            1E USER FIRST$
       BSS  2            20 USER LIMIT$
       BSS  2            22 USER B/BUF$ { BBUF$
       BSS  2            24 USER B/SCR$ { BSCR$
$DKFLO BSS  2            26 USER DISK_LO  (LOW DISK FENCE)
$DKFHI BSS  2            28 USER DISK_HI  (HIGH DISK FENCE)
$DKSIZ BSS  2            2A USER DISK_SIZE (IN SCREENS)
$DKBUF BSS  2            2C USER DISK_BUF  (BUFFER LOC IN VDP. SIZE=1K)  1K)
$PABS  BSS  2            2E USER PABS  (AREA FOR PABS ETC.)
$SWDTH BSS  2            30 USER SCRN_WIDTH
$SSTRT BSS  2            32 USER SCRN_START
$SEND  BSS  2            34 USER SCRN_END
$ISR   BSS  2            36 USER ISR
$ALTI  BSS  2            38 USER ALTIN
$ALTO  BSS  2            3A USER ALTOUT
       BSS  2            3C USER FENCE
       BSS  2            3E USER BLK
       BSS  2            40 USER IN
$OUT   BSS  2            42 USER OUT
       BSS  2            44 USER SCR
$OFFST BSS  2            46 USER OFFSET
       BSS  2            48 USER CONTEXT
       BSS  2            4A USER CURRENT
       BSS  2            4C USER STATE
       BSS  2            4E USER BASE
       BSS  2            50 USER DPL
       BSS  2            52 USER FLD
       BSS  2            54 USER CSP
       BSS  2            56 USER R# { RNUM
       BSS  2            58 USER HLD
       BSS  2            5A USER USE
       BSS  2            5C USER PREV
       BSS  2            5E
FORLNK BSS  2            60 USER FORTH_LINK
       BSS  2            62
```

```
       BSS  2             64 USER ECOUNT
UMAX   BSS  0
*
*
       DORG >BC80
DPBASE BSS  >4320         START OF RAM DICTIONARY
SPBASE BSS  0             BASE OF PARAMETER STACK
       BSS  82            TEXT INPUT BUFFER
RBASE  EQU  >3FFE         BASE OF RETURN STACK
*
*** UTILITY EQUATES ******
       COPY "DSK2.UTILEQU"
*
       AORG >2002
       DATA ENTLNK       PATCH TO MY ENTRY
*
       AORG >C000
FMOVE  DECT SP           COPY TO MOVE TO CONSOLE RAM
       MOV  W,*SP
       MOV  LINK,W
       DECT R
       MOV  IP,*R
       MOV  W,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
       MOV  *R+,IP
       MOV  *IP+,W
       MOV  *W+,TEMP1
       B    *TEMP1
*
FORTH  LIMI 0
       LWPI MAINWS
       LI   TEMP1,FORTH-FMOVE
       LI   TEMP2,FMOVE
       LI   TEMP3,DODOES
MLOOP  MOV  *TEMP2+,*TEMP3+
       DECT TEMP1
       JNE  MLOOP
*
*** INITIALIZE VDP STUFF
*
       LI   TEMP0,>01B0  BLANK SCREEN
       BLWP @VWTR
       LI   TEMP0,>030E  SET COLOR TABLE AT >0380
       BLWP @VWTR
       LI   TEMP0,>0401  SET PATTERN DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>0506  SET SPRITE ATTRIBUTE TABLE >0300
       BLWP @VWTR
```

```
       LI   TEMP0,>0601  SET SPRITE DESCRIPTOR TABLE >0800
       BLWP @VWTR
       LI   TEMP0,>07F4  SET TEXTMODE COLORS
       BLWP @VWTR
       LI   TEMP0,>2000  BLANK
       LI   TEMP1,>960   TEXT-MODE SCREEN SIZE
       LI   TEMP2,>0     SCREEN STARTS AT 0
       BL   @FILLER      CLEAR SCREEN
       LI   TEMP0,>FF00  CHAR FF
       LI   TEMP1,>2048  BLOCK SIZE
       LI   TEMP2,>800   STARTING LOCATION IN VDP
       BL   @FILLER      FILL AREA WITH FF'S
       LI   TEMP0,>81F0
       SWPB TEMP0
       MOVB TEMP0,@>83D4 USED TO UPDATE VDP REG EACH KEYSTROKE
       MOVB TEMP0,@VDPWA FORCE TEXT MODE
       SWPB TEMP0
       MOVB TEMP0,@VDPWA
       LI   TEMP0,>900   VDP LOCATION
       MOV  TEMP0,@FAC
       CLR  TEMP1        CLEAR GPL STATUS
       MOVB TEMP1,@KYSTAT
       LI   TEMP7,>3E0
       MOV  TEMP7,@VSPTR
       BLWP @GPLLNK      LOAD CAPITAL LETTER SHAPES
       DATA >0018
       LI   TEMP2,>1200
       CB   TEMP2,@3     IF BYTE @3 IN THE CONSOLE IS >12 IT'S A 99/4
       JEQ  FOUR         DON'T LOAD LOWER CASE IN A 99/4
       LI   TEMP0,>0B00
       MOV  TEMP0,@FAC
       MOVB TEMP1,@KYSTAT
       BLWP @GPLLNK      LOAD LOWER CASE IN 99/4A
       DATA >004A
*
FOUR   LI   TEMP1,UBASE0
       B    @FF9900      BRANCH TO ACMSRC
*
FILLER ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLLOOP MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLLOOP       NOT DONE, FILL ANOTHER
       B    *LINK
*
       DORG >2010
*
```

```
$BUFF  BSS  5*>404        I/O BUFFERS
$LO    BSS  0
*
       AORG $LO
*
*
*** INTERRUPT SERVICE
*
INT1   LI   TEMP1,INT2   FIX 'NEXT' SO THAT INTERRUPT IS
       MOV  TEMP1,@2*NEXT+MAINWS  PROCESSED AT END OF
       LWPI >83C0        NEXT 'CODE' WORD
       RTWP
*
INT2   LIMI 0
       MOVB @>83D4,TEMP0
       SRL  TEMP0,8
       ORI  TEMP0,>100
       ANDI TEMP0,>FFDF
       BLWP @VWTR        TURN OFF VDP INTERRUPTS
       LI   NEXT,$NEXT   RESTORE 'NEXT'
       SETO @INTACT
       DECT R            SET UP RETURN LINKAGE
       MOV  IP,*R
       LI   IP,INT3
       MOV  @$ISR(U),W   DO THE FORTH ROUTINE
       B    @DOEXEC
INT3   DATA $+2
       DATA $+2
       MOV  *R+,IP
       CLR  @INTACT
       MOVB @>83D4,TEMP0
       SRL  TEMP0,8
       AI   TEMP0,>100
       MOVB @VDPSTA,TEMP1 REMOVE PENDING INTERRUPT
       BLWP @VWTR
       LIMI 2
       B    *NEXT        CONTINUE NORMAL TASK
*==============================================================
BKLINK MOV  @INTACT,TEMP7
       JNE  BKLIN1
       LIMI 2
BKLIN1 B    *LINK
*==============================================================
*
$SYS$  LIMI 0
       MOV  @SYSTAB(TEMP1),TEMP0
       B    *TEMP0
       DATA DSK          CODE=-18 DRIVE SELECTION
       DATA DSK          CODE=-16 READ DISK
       DATA DSK          CODE=-14 WRITE DISK
```

```
       DATA GXY          CODE=-12 GOTOXY
       DATA QKY          CODE=-10 ?KEY
       DATA QTM          CODE=-8  ?TERMINAL
       DATA CLF          CODE=-6  CRLF
       DATA EMT          CODE=-4  EMIT
       DATA KY           CODE=-2  KEY
SYSTAB DATA SBW          CODE=0   VSBW
       DATA MBW          CODE=2   VMBW
       DATA SBR          CODE=4   VSBR
       DATA MBR          CODE=6   VMBR
       DATA WTR          CODE=8   VWTR
       DATA GPL          CODE=10  GPLLNK
       DATA XML          CODE=12  XMLLNK
       DATA DSR          CODE=14  DSRLNK
       DATA CLS          CODE=16  CLS
       DATA FMT          CODE=18  FORMAT-DISK
       DATA FILL         CODE=20  VFILL
       DATA AOX          CODE=22  VAND
       DATA AOX          CODE=24  VOR
       DATA AOX          CODE=26  VXOR
*
*== THIS IS A VDP SINGLE BYTE WRITE.  CODE=0  ==============
*
SBW    MOV  *SP+,TEMP0   VDP ADDRESS (DESTINATION)
       MOV  *SP+,TEMP1   CHARACTER TO WRITE
       SWPB TEMP1        GET IN LEFT BYTE
       BLWP @VSBW
       B    @BKLINK
*
*== THIS IS A VDP MULTI BYTE WRITE.  CODE=2   ==============
*
MBW    MOV  *SP+,TEMP2   NUMBER OF BYTE TO MOVE
       MOV  *SP+,TEMP0   VDP ADDRESS (DESTINATION)
       MOV  *SP+,TEMP1   RAM ADDRESS (SOURCE)
       BLWP @VMBW
       B    @BKLINK
*
*== THIS IS A VDP SINGLE BYTE READ.  CODE=4   ==============
*
SBR    MOV  *SP,TEMP0    VDP ADDRESS (SOURCE)
       BLWP @VSBR
       SRL  TEMP1,8      CHARACTER TO RIGHT HALF FOR FORTH
       MOV  TEMP1,*SP    STACK IT
       B    @BKLINK
*
*== THIS IS A VDP MULTI BYTE READ.  CODE=6    ==============
*
MBR    MOV  *SP+,TEMP2   NUMBER OF BYTES TO READ
       MOV  *SP+,TEMP1   RAM ADDRESS (DESTINATION)
       MOV  *SP+,TEMP0   VDP ADDRESS (SOURCE)
       BLWP @VMBR
```

```
       B    @BKLINK
*
*== VDP REGISTER WRITE.  CODE=8               ==============
*
WTR    MOV  *SP+,TEMP1   VDP REGISTER NUMBER
       MOV  *SP+,TEMP0   DATA FOR REGISTER
       SWPB TEMP1        GET REGISTER TO LEFT BYTE
       MOVB TEMP1,TEMP0  PLACE WITH DATA
       BLWP @VWTR
       B    @BKLINK
*
*== THIS IS THE GPL LINK UTILITY.  CODE=10    ==============
*
GPL    CLR  TEMP0
       MOVB TEMP0,@KYSTAT
       LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,GPLLNK      TO THE GPLLNK UTILITY
       MOV  *SP+,TEMP2        WITH THIS DATA IDENTIFYING THE ROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE XML LINK UTILITY.  CODE=12    ==============
*
XML    LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,XMLLNK      TO THE XMLLNK UTILITY
       MOV  *SP+,TEMP2        WITH THIS DATA IDENTIFYING THE ROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE DSR LINK UTILITY.  CODE=14    ==============
*
DSR    LI   TEMP0,>0420       CONSTRUCT THE BLWP INSTRUCTION
       LI   TEMP1,DSRLNK      TO THE DSRLNK UTILITY
       MOV  *SP+,TEMP2        THIS DATUM SELECTS DSR OR SUBROUTINE
       LI   TEMP3,>045B       CONSTRUCT THE B *LINK INSTRUCTION
       MOV  LINK,TEMP4        SAVE LINK ADDRESS
       BL   @2*TEMP0+MAINWS   EXECUTE THE ABOVE INSTRUCTIONS
       MOV  TEMP4,LINK        AND RECONSTRUCT LINK
       B    @BKLINK
*
*== THIS IS THE SCREEN CLEARING UTILITY.  CODE=16 ==========
*
CLS    MOV  @$SSTRT(U),TEMP2     BEGINNING OF SCREEN IN VDP
       MOV  @$SEND(U),TEMP1      END OF SCREEN IN VDP
```

```
       S    TEMP2,TEMP1       SCREEN SIZE
       LI   TEMP0,>2000       BLANK CHARACTER
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
*
*== THIS IS THE DISK FORMATTER.  CODE=18   =================
*
FMT    MOV  @$DKBUF(U),TEMP0 VDP BUFFER START ADDRESS (3300 BYTES)
       MOV  TEMP0,@FAC+4
       MOV  *SP+,TEMP0
       SLA  TEMP0,8      DRIVE # TO LEFT BYTE
       AI   TEMP0,40     40 TRACKS
       MOV  TEMP0,@FAC+2
       MOV  @$PABS(U),TEMP0 SETUP SUBR NAME (>11)
       MOV  TEMP0,@SUBPTR
       LI   TEMP1,DFMT
       LI   TEMP2,2
       BLWP @VMBW
       BLWP @DSRLNK
       DATA >A
       B    @BKLINK
DFMT   DATA >0111        USED BY VMBW ABOVE
*
*== THIS IS THE VDP FILL ROUTINE.  CODE=20
*
FILL   MOV  *SP+,TEMP0   FILL CHARACTER
       SWPB TEMP0        TO LEFT BYTE
       MOV  *SP+,TEMP1   FILL COUNT
       MOV  *SP+,TEMP2   ADDRESS TO START VDP FILL
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
*=========================================================
FILL1  ORI  TEMP2,>4000  SET BIT FOR VDP WRITE
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
FLOOP  MOVB TEMP0,@VDPWD WRITE A BYTE
       DEC  TEMP1
       JNE  FLOOP        NOT DONE, FILL ANOTHER
       B    *LINK
*=========================================================
*
*== VDP BYTE 'AND' 'OR' 'XOR' ROUTINES.  CODE=22,24,26   ===
*
AOX    MOV  *SP+,TEMP2   VDP ADDRESS
```

```
       SWPB TEMP2
       MOVB TEMP2,@VDPWA LS BYTE FIRST
       SWPB TEMP2
       MOVB TEMP2,@VDPWA THEN MS BYTE
       NOP               KILL TIME
       MOVB @VDPRD,TEMP3 READ BYTE
       MOV  *SP+,TEMP0   GET DATA TO OPERATE WITH
       SWPB TEMP0        TO LEFT BYTE
*** NOW DO REQUESTED OPERATION *****************
       CI   TEMP1,24
       JEQ  DOOR
       JGT  DOXOR
       INV  TEMP3        THESE TWO INSTRUCTIONS
       SZC  TEMP3,TEMP0  PERFORM AN 'AND'
       JMP  FINAOX
DOOR   SOC  TEMP3,TEMP0  PERFORM OR
       JMP  FINAOX
DOXOR  XOR  TEMP3,TEMP0  PERFORM XOR
FINAOX LI   TEMP1,1
       MOV  LINK,TEMP7
       BL   @FILL1
       MOV  TEMP7,LINK
       B    @BKLINK
*
*=========================================================
*
*== KEY ROUTINE  CODE= -2  ================================
*
KY     MOV  @$ALTI(U),TEMP0
       JEQ  KEY0
       CLR  TEMP7
       MOVB TEMP7,@KYSTAT
       INC  TEMP0
       BLWP @VSBR
       ANDI TEMP1,>1F00
       BLWP @VSBW
       MOV  TEMP0,TEMP1
       AI   TEMP1,8
       MOV  TEMP1,@SUBPTR
       BLWP @DSRLNK
       DATA >8
       DECT TEMP0
       BLWP @VSBR
       SRL  TEMP1,8
       MOV  TEMP1,TEMP0
       B    @BKLINK
KEY0   MOV  @KEYCNT,TEMP7
       INC  TEMP7
       JNE  KEY1
       MOV  @CURPO$(U),TEMP0
```

```
       BLWP @VSBR             READ CHARACTER AT CURSOR POSITION
       MOVB TEMP1,@CURCHR     AND SAVE IT
       LI   TEMP1,>1E00       PLACE CURSOR CHARACTER ON SCREEN
       BLWP @VSBW
*
KEY1   LI   TEMP4,>2000       MASK TO CHECK STATUS
       BLWP @KSCAN
       MOVB @KYSTAT,TEMP0
       COC  TEMP4,TEMP0
       JEQ  KEY2              JMP IF KEY WAS PRESSED
*
       CI   TEMP7,100         NO KEY PRESSED
       JNE  KEY3
       MOVB @CURCHR,TEMP1
       JMP  KEY5
*
KEY3   CI   TEMP7,200
       JNE  KEY4
       CLR  TEMP7
       LI   TEMP1,>1E00       CURSOR CHAR
KEY5   MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
KEY4   MOV  TEMP7,@KEYCNT
       MOV  @INTACT,TEMP7
       JNE  KEY6
       LIMI 2
KEY6   DECT IP                THIS WILL RE-EXECUTE KEY
       B    *NEXT
*KE     DATA KEY,SEMIS
*
*
KEY2   SETO @KEYCNT           KEY WAS PRESSED
       MOV  @CURPO$(U),TEMP0  RESTORE CHARACTER AT CURSOR LOCATION
       MOVB @CURCHR,TEMP1
       BLWP @VSBW
       MOVB @KYCHAR,TEMP0     PUT CHAR IN RIGHT HALF OF TEMP0
       SRL  TEMP0,8
       B    @BKLINK
*
*== EMIT ROUTINE  CODE= -4  ================================
*
EMT    MOV  TEMP2,TEMP1
       MOV  @$ALTO(U),TEMP0
       JEQ  EMIT0
       CLR  TEMP7             ALTOUT ACTIVE
       MOVB TEMP7,@KYSTAT
       DEC  TEMP0
       SWPB TEMP1
       BLWP @VSBW
       INCT TEMP0
       BLWP @VSBR
```

```
       ANDI TEMP1,>1F00
       BLWP @VSBW
       AI   TEMP0,8
       MOV  TEMP0,@SUBPTR
       BLWP @DSRLNK
       DATA >8
       B    @BKLINK
*
EMIT0  CI   TEMP1,7      IS IT A BELL?
       JNE  NOTBEL
       CLR  TEMP2
       MOVB TEMP2,@KYSTAT
       MOVB @GRMSAV,@GRMWA    RESTORE GROM ADDRESS
       NOP
       MOVB @GRMSAV+1,@GRMWA
       BLWP @GPLLNK
       DATA >0036        EMIT ERROR TONE
       JMP  EMEXIT
*
NOTBEL CI   TEMP1,8      IS IT A BACKSPACE?
       JNE  NOTBS
       LI   TEMP1,>2000
       MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
       JGT  DECCUR
       JMP  EMEXIT
DECCUR DEC  @CURPO$(U)
       JMP  EMEXIT
*
NOTBS  CI   TEMP1,>A     IS IT A LINE FEED?
       JNE  NOTLF
       MOV  @$SEND(U),TEMP7
       S    @$SWDTH(U),TEMP7
       C    @CURPO$(U),TEMP7
       JHE  SCRLL
       A    @$SWDTH(U),@CURPO$(U)
       JMP  EMEXIT
SCRLL  MOV  LINK,TEMP7
       BL   @SCROLL
       MOV  TEMP7,LINK
       JMP  EMEXIT
*
*** SCROLLING ROUTINE
*
SCROLL MOV  @$SSTRT(U),TEMP0     VDP ADDR
       LI   TEMP1,LINBUF      BUFFER
       MOV  @$SWDTH(U),TEMP2     COUNT
       A    TEMP2,TEMP0       START AT LINE 2
SCROL1 BLWP @VMBR
       S    TEMP2,TEMP0       ONE LINE BACK TO WRITE
```

```
       BLWP @VMBW
       A    TEMP2,TEMP0       TWO LINES AHEAD FOR NEXT READ
       A    TEMP2,TEMP0
       C    TEMP0,@$SEND(U)      END OF SCREEN?
       JL   SCROL1
       MOV  TEMP2,TEMP1       BLANK BOTTOM ROW OF SCREEN
       LI   TEMP0,>2000       BLANK
       S    @$SEND(U),TEMP2
       NEG  TEMP2             NOW CONTAINS ADDRESS OF START OF LAST LINE
       MOV  LINK,TEMP6
       BL   @FILL1            WRITE THE BLANKS
       B    *TEMP6
*
NOTLF  CI   TEMP1,>D     IS IT A CARRIAGE RETURN?
       JNE  NOTCR
       CLR  TEMP0
       MOV  @CURPO$(U),TEMP1
       MOV  TEMP1,TEMP3
       S    @$SSTRT(U),TEMP1  ADJUSTED FOR SCREEN NOT AT 0
       MOV  @$SWDTH(U),TEMP2
       DIV  TEMP2,TEMP0
       S    TEMP1,TEMP3
       MOV  TEMP3,@CURPO$(U)
       JMP  EMEXIT
*
NOTCR  SWPB TEMP1        ASSUME IT IS A PRINTABLE CHARACTER
       MOV  @CURPO$(U),TEMP0
       BLWP @VSBW
       MOV  @$SEND(U),TEMP2
       DEC  TEMP2
       C    TEMP0,TEMP2
       JNE  NOTCR1
       MOV  @$SEND(U),TEMP0
       S    @$SWDTH(U),TEMP0 WAS LAST CHAR ON SCREEN. SCROLL
       MOV  TEMP0,@CURPO$(U)
       JMP  SCRLL
NOTCR1 INC  TEMP0        NO SCROLL NECESSARY
       MOV  TEMP0,@CURPO$(U)
*
EMEXIT B    @BKLINK
*
*== CRLF ROUTINE  CODE= -6  ================================
*
CLF    MOV  LINK,TEMP5
       LI   TEMP2,>000D
       BL   @EMT
       LI   TEMP2,>000A
       LIMI 0            PREVIOUS CALL TO EMT ALTERED INT MASK
       BL   @EMT
       MOV  TEMP5,LINK
       B    @BKLINK
```

```
*
*== ?TERMINAL ROUTINE CODE= -8  ============================
*
QTM     BLWP @KSCAN
        MOVB @KYCHAR,TEMP0
        SRL  TEMP0,8
        CI   TEMP0,>2
        JEQ  QTERM1
        CLR  TEMP0
QTERM1  B    @BKLINK
*
*== ?KEY ROUTINE CODE= -10  ================================
*
QKY     BLWP @KSCAN
        MOVB @KYCHAR,TEMP0
        SRL  TEMP0,8
        CI   TEMP0,>00FF
        JNE  QKEY1
        CLR  TEMP0
QKEY1   B    @BKLINK
*
*== GOTOXY ROUTINE CODE= -12  ==============================
*
GXY     MPY  @$SWDTH(U),TEMP3
        A    TEMP2,TEMP4       POSITION WITHIN SCREEN
        A    @$SSTRT(U),TEMP4     ADD VDP OFFSET TO SCREEN TOP
        MOV  TEMP4,@CURPO$(U)
        B    @BKLINK
*
*** ENTRY POINT FOR DISK HANDLING ROUTINES
*
DSK     MOV  TEMP1,TEMP7  SAVE CODE OF DISK ROUTINE
        MOV  @$PABS(U),TEMP0
        LI   TEMP1,>0100  SET UP VDP FOR DISK DSR
        BLWP @VSBW        LEVEL ONE R/W ACCESS
        INC  TEMP0
        LI   TEMP1,>1000
        BLWP @VSBW
        MOV  TEMP4,TEMP1  ADDRESS TO TEMP1
        CI   TEMP7,-14
        JNE  NOTWD
*
*** DISK WRITE ROUTINE  CODE=-14
*
        C    TEMP2,@$DKFLO(U)  TEST DISK FENCES
        JHE  DKWT1
DKWT0   LI   TEMP0,>B
        JMP  DKEXIT
DKWT1   C    TEMP2,@$DKFHI(U)
        JHE  DKWT0
```

```
       MOV  @$DKBUF(U),TEMP0 VDP BUFFER
       MOV  TEMP2,TEMP5  SAVE BLOCK #
       MOV  TEMP3,TEMP2  BYTES/BLOCK
       BLWP @VMBW
       CLR  TEMP4
       DIV  @$DKSIZ(U),TEMP4  BLOCK # IN TEMP5
       INC  TEMP4
       SWPB TEMP4
       MOV  TEMP4,@FAC+2
       SLA  TEMP5,2      CONV BLOCK # TO SECT #
       MOV  TEMP5,@FAC+6
       LI   TEMP6,4      SET COUNTER
       MOV  @$DKBUF(U),@FAC+4
WTLOOP MOV  @$PABS(U),@SUBPTR
       BLWP @DSRLNK
       DATA >A
       MOVB @DSKERR,TEMP7
       JNE  DKERR
       DEC  TEMP6        CHECK LOOP COUNT
       JEQ  DSKDON
       MOV  @FAC,@FAC+6
       INC  @FAC+6
       LI   TEMP0,256
       A    TEMP0,@FAC+4
       JMP  WTLOOP
DKERR  LI   TEMP0,6
       JMP  DKEXIT
DSKDON CLR  TEMP0
*
DKEXIT B    @BKLINK
*
*
NOTWD  CI   TEMP7,-16
       JNE  NOTRD
*
*** DISK READ ROUTINES  CODE=-16
*
       MOV  TEMP2,TEMP5  SAVE BLK#
       CLR  TEMP4
       DIV  @$DKSIZ(U),TEMP4  BLOCK # IN TEMP5
       INC  TEMP4
       SWPB TEMP4
       INC  TEMP4        SET BIT FOR READ
       MOV  TEMP4,@FAC+2
       SLA  TEMP5,2
       MOV  TEMP5,@FAC+6
       MOV  @$PABS(U),TEMP4
       LI   TEMP6,4      INIT COUNTER
       MOV  @$DKBUF(U),@FAC+4
RDLOOP MOV  TEMP4,@SUBPTR
       BLWP @DSRLNK
```

```
       DATA >A
       MOVB @DSKERR,TEMP7
       JNE  DKERR
       DEC  TEMP6
       JEQ  RDDONE
       MOV  @FAC,@FAC+6
       INC  @FAC+6
       LI   TEMP0,256
       A    TEMP0,@FAC+4
       JMP  RDLOOP
RDDONE MOV  @$DKBUF(U),TEMP0 VDP BUFFER
       MOV  TEMP3,TEMP2  #BYTES/BLOCK
       BLWP @VMBR
       JMP  DSKDON
*
*
NOTRD  EQU  $
*
*** DRIVE SELECTION ROUTINE   CODE =-18
*
       CLR  TEMP0
DRV1   DEC  TEMP2
       JLT  DRV2
       A    @$DKSIZ(U),TEMP0
       JMP  DRV1
DRV2   B    @BKLINK
*
*=============================================================
*
UBASE0 BSS  6             BASE OF USER VARIABLES
       DATA UBASE0        06 USER UCONS
       DATA SPBASE        08 USER S0
       DATA RBASE         0A USER R0 { R0$
       DATA $UVAR         0C USER U0
       DATA SPBASE        0E USER TIB
       DATA 31            10 USER WIDTH
       DATA DPBASE        12 USER DP
       DATA $SYS$         14 USER SYS$
       DATA 0             16 USER CURPOS
       DATA INT1          18 USER INTLNK
       DATA 1             1A USER WARNING
       DATA 64            1C USER C/L$ { CL$
       DATA $BUFF         1E USER FIRST$
       DATA $LO           20 USER LIMIT$
       DATA 1024          22 USER B/BUF$ { BBUF$
       DATA 1             24 USER B/SCR$ { BSCR$
       DATA 1             26 USER DISK_LO  (LOW DISK FENCE)
       DATA 90            28 USER DISK_HI  (HIGH DISK FENCE)
       DATA 90            2A USER DISK_SIZE (IN SCREENS)
       DATA >1000         2C USER DISK_BUF  (BUFFER LOC IN VDP. SIZE=1K)  1K)
```

```
       DATA >460          2E USER PABS  (AREA FOR PABS ETC.)
       DATA 40            30 USER SCRN_WIDTH
       DATA 0             32 USER SCRN_START
       DATA 960           34 USER SCRN_END
       DATA INT1          36 USER ISR
       DATA 0             38 USER ALTIN
       DATA 0             3A USER ALTOUT
*
$UVAR  BSS  >80           USER VARIABLE AREA
*
       COPY "DSK2.UTILROM"
       COPY "DSK2.UTILRAM"
*=============================================================
LINBUF BSS  40           BUFFER FOR SCROLLING
KEYCNT DATA -1           USED IN CURSOR FLASH LOGIC
CURCHR BSS  2            CHAR AT CURSOR POSITION
GRMSAV BSS  2            SAVE GROM ADDRESS DURING DSRLNK
INTACT DATA 0            NON-ZERO DURING INTERRUPT SERVICE
*
*=============================================================
*
       END
```

Disk 10. Contents of File UTILEQU

```
SCNKEY EQU  >000E
XMLTAB EQU  >0CFA        XML TABLES (BASE)
FLAG2  EQU  >8349
SCLEN  EQU  >8355
SCNAME EQU  >8356
SUBSTK EQU  >8373
CRULST EQU  >83D0
SADDR  EQU  >83D2
GPLWS  EQU  >83E0        GPL/EXTENDED BASIC WORKSPACE
PAD    EQU  >8300
VDPRD  EQU  >8800        VDP read data address
VDPWD  EQU  >8C00        VDP write data address
VDPWA  EQU  >8C02        VDP write address address
R0LB   EQU  >83E1
R1LB   EQU  >83E3
R3LB   EQU  >83E7
```

Disk 10. Contents of File UTILRAM

```
SVGPRT DATA 0            Save GPL return address
SAVCRU DATA 0            CRU address of peripheral
SAVENT DATA 0            Entry address of DSR
SAVLEN DATA 0            Save device name length
SAVPAB DATA 0            Ptr into device name in PAB
SAVVER DATA 0            Version number of DSR
NAMBUF DATA 0,0,0,0
*
*** General utility workspace registers (Overlaps next WS)
UTILWS DATA 0,0
       BYTE 0
R2LB   BYTE 0
*
*** DSR link routine workspace registers (Overlaps prev. WS)
DLNKWS DATA 0,0,0,0,0
TYPE   DATA 0,0,0,0,0,0,0,0,0,0,0
*
```

Disk 10. Contents of File UTILROM

```
C100   DATA 100
H20    EVEN
H2000  DATA >2000
DECMAL TEXT '.'
HAA    BYTE >AA
       EVEN
*
* Utility Vectors
*
GPLLNK DATA UTILWS,GLENTR     Link to GROM routines
XMLLNK DATA UTILWS,XMLENT     Link to ROM routines
KSCAN  DATA UTILWS,KSENTR     Keyboard scan
VSBW   DATA UTILWS,VSBWEN     VDP single byte write
VMBW   DATA UTILWS,VMBWEN     VDP multiple byte write
VSBR   DATA UTILWS,VSBREN     VDP single byte read
VMBR   DATA UTILWS,VMBREN     VDP multiple byte read
VWTR   DATA UTILWS,VWTREN     VDP write to register
DSRLNK DATA DLNKWS,DLENTR     Link to device service routine
*
*=============================================================
ENTLNK MOV  R11,@SVGPRT  Save return to GPL interpreter
       MOVB @GRMRA,R1    Save GROM address of XML to
       SWPB R1           this routine.
       MOVB @GRMRA,R1
       SWPB R1
       AI   R1,-3
       MOV  R1,@GRMSAV
       LI   R1,RTFGPL    Change XML vectors
       MOV  R1,@>2000    for Extended BASIC
       MOV  R1,@>2002    for Editor/Assembler
       B    @FORTH       Go to FORTH
*
*=============================================================
*      LINK TO SYSTEM XML UTILITIES
*
XMLENT MOV  *R14+,@GPLWS+2    Get argument
       LWPI GPLWS             Select GPL workspace
       MOV  R11,@UTILWS+22    Save GPL return address
       MOV  R1,R2             Make a copy of argument
       CI   R1,>8000          Direct address in ALC?
       JH   XML30             We have the address
       SRL  R1,12
       SLA  R1,1
       SLA  R2,4
       SRL  R2,11
       A    @XMLTAB(R1),R2
       MOV  *R2,R2
XML30  BL   *R2
```

```
       LWPI UTILWS            GET BACK TO RIGHT WS
       MOV  R11,@GPLWS+22     Restore GPL return address
       RTWP
*
*==============================================================
*** Link to GPL utilities
*
GLENTR MOVB @SUBSTK,R2        Fetch GPL subroutine stack ptr
       SRL  R2,8              Make it an index
       AI   R2,PAD
       INCT R2
       MOV  @GRMSAV,R1        Push XML address for return
       MOVB R1,*R2
       SWPB R1
       MOVB R1,@1(R2)
       SWPB R2                Adjust stack pointer
       MOVB R2,@SUBSTK
       MOVB *R14+,@GRMWA      Set up address to call
       MOVB *R14+,@GRMWA      and second byte, adjusting return
       LWPI GPLWS
       MOV  @SVGPRT,R11
       RT                     Return to GPL
*
*** Return to assembly language from GPL
*
RTFGPL LWPI UTILWS       Select utility workspace
       RTWP              Return to calling AL routine
*
*==============================================================
*      KEYBOARD SCAN
*
KSENTR LWPI GPLWS
       MOV  R11,@UTILWS+22   Save GPL return address
       BL   @SCNKEY
       LWPI UTILWS
       MOV  R11,@GPLWS+22    Restore GPL return address
       RTWP
*
*==============================================================
*      VDP UTILITIES
*
** VDP single byte write
*
VSBWEN BL   @WVDPWA          Write out address
       MOVB @2(R13),@VDPWD   Write data
       RTWP                  Return to calling program
*
** VDP multiple byte write
*
VMBWEN BL   @WVDPWA          Write out address
VWTMOR MOVB *R1+,@VDPWD      Write a byte
```

```
       DEC  R2               Decrement byte count
       JNE  VWTMOR           More to write?
       RTWP                  Return to calling Program
*
** VDP single byte read
*
VSBREN BL   @WVDPRA          Write out address
       MOVB @VDPRD,@2(R13)   Read data
       RTWP                  Return to calling program
*
** VDP multiple byte read
*
VMBREN BL   @WVDPRA          Write out address
VRDMOR MOVB @VDPRD,*R1+      Read a byte
       DEC  R2               Decrement byte count
       JNE  VRDMOR           More to read?
       RTWP                  Return to calling program
*
** VDP write to register
*
VWTREN MOV  *R13,R1          Get register number and value
       MOVB @1(R13),@VDPWA   Write out value
       ORI  R1,>8000         Set for register write
       MOVB R1,@VDPWA        Write out register number
       RTWP                  Return to calling program
*
** Set up to write to VDP
*
WVDPWA LI   R1,>4000
       JMP  WVDPAD
*
** Set up to read VDP
*
WVDPRA CLR  R1
*
** Write VDP address
*
WVDPAD MOV  *R13,R2          Get VDP address
       MOVB @R2LB,@VDPWA     Write low byte of address
       SOC  R1,R2            Properly adjust VDP write bit
       MOVB R2,@VDPWA        Write high byte of address
       MOV  @2(R13),R1       Get CPU RAM address
       MOV  @4(R13),R2       Get byte count
       RT                    Return to calling routine
*
*=========================================================
*      CIF - Convert integer to floating                  *
*
CIF    LI   R4,FAC            Will convert into the FAC
       MOV  *R4,R0            Get integer into register
```

```
       MOV  R4,R6             Copy ptr to FAC to clear it
       CLR  *R6+              Clear FAC,FAC+1
       CLR  *R6+              IN CASE HAD A STRING IN FAC
       MOV  R0,R5             IS INTEGER EQUAL TO ZERO?
       JEQ  CIFRT             YES - ZERO RESULT AND RETURN
       ABS  R0                GET ABS VALUE OF ARG
       LI   R3,>40            GET EXPONENT BIAS
       CLR  *R6+              CLEAR WORDS IN RESULT THAT
       CLR  *R6                 MIGHT NOT GET A VALUE
       CI   R0,100            IS INTEGER < 100?
       JL   CIF02             YES-JUST PUT IN 1ST FRACTION
*                                PART
       CI   R0,10000          NO-IS ARG < 100,2?
       JL   CIF01             YES-JUST 1 DIVISION NECESSARY
*                             NO - 2 DIVISIONS ARE NECESSARY
       INC  R3                ADD 1 TO EXPONENT FOR 1ST DIV
       MOV  R0,R1             PUT # IN LOW ORDER WORD FOR
*                               THE DIVIDE
       CLR  R0                CLEAR HIGH ORDER WORD FOR THE
*                                DIVIDE
       DIV  @C100,R0          DIVIDE BY THE RADIX
       MOVB @R1LB,@3(R4)      MOVE THE RADIX DIGIT IN
CIF01
       INC  R3                ADD 1 TO EXPONENT FOR DIVIDE
       MOV  R0,R1             PUT IN LOW ORDER FOR DIVIDE
       CLR  R0                CLEAR HIGH ORDER FOR DIVIDE
       DIV  @C100,R0          DIVIDE BY THE RADIX
       MOVB @R1LB,@2(R4)      PUT NEXT RADIX DIGIT IN
CIF02
       MOVB @R0LB,@1(R4)      PUT HIGHEST ORDER RADIX DIGIT
*                             IN
       MOVB @R3LB,*R4         PUT EXPONENT IN
       INV  R5                IS RESULT POSITIVE?
       JLT  CIFRT             YES - SIGN IS CORRECT
       NEG  *R4               NO - MAKE IT NEGATIVE
CIFRT  RT
*
*=========================================================
*** Link to device service routine
*
DLENTR MOV  *R14+,R5          Fetch program type for link
       SZCB @H20,R15          Reset equal bit
       MOV  @SCNAME,R0        Fetch pointer into PAB
       MOV  R0,R9             Save pointer
       AI   R9,-8             Adjust pointer to flag byte
       BLWP @VSBR             Read device name length
       MOVB R1,R3             Store it elsewhere
       SRL  R3,8              Make it a word value
       SETO R4                Initialize a counter
       LI   R2,NAMBUF         Point to NAMBUF
LNK$LP INC  R0                Point to next char of name
```

```
       INC  R4                Increment character counter
       C    R4,R3             End of name?
       JEQ  LNK$LN            Yes
       BLWP @VSBR             Read current character
       MOVB R1,*R2+           Move it to NAMBUF
       CB   R1,@DECMAL        Is it a decimal point?
       JNE  LNK$LP            No
LNK$LN MOV  R4,R4             Is name length zero?
       JEQ  LNKERR            Yes, error
       CI   R4,7              Is name length > 7?
       JGT  LNKERR            Yes, error
       CLR  @CRULST
       MOV  R4,@SCLEN-1       Store name length for search
       MOV  R4,@SAVLEN        Save device name length
       INC  R4                Adjust it
       A    R4,@SCNAME        Point to position after name
       MOV  @SCNAME,@SAVPAB   Save pointer into device name
*
*** Search ROM CROM GROM for DSR
*
SROM   LWPI GPLWS             Use GPL workspace to search
       CLR  R1                Version found of DSR etc.
       LI   R12,>0F00         Start over again
NOROM  MOV  R12,R12           Anything to turn off
       JEQ  NOOFF             No
       SBZ  0                 Yes, turn it off
NOOFF  AI   R12,>0100         Next ROM'S turn on
       CLR  @CRULST           Clear in case we're finished
       CI   R12,>2000         At the end
       JEQ  NODSR             No more ROMs to turn on
       MOV  R12,@CRULST       Save address of next CRU
       SBO  0                 Turn on ROM
       LI   R2,>4000          Start at beginning
       CB   *R2,@HAA          Is it a valid ROM?
       JNE  NOROM             No
       A    @TYPE,R2          Go to first pointer
       JMP SGO2
SGO    MOV  @SADDR,R2         Continue where we left off
       SBO  0                 Turn ROM back on
SGO2   MOV  *R2,R2            Is address a zero
       JEQ  NOROM             Yes, no program to look at
       MOV  R2,@SADDR         Remember where we go next
       INCT R2                Go to entry point
       MOV  *R2+,R9           Get entry address
*
*** See if name matches
*
       MOVB @SCLEN,R5         Get length as counter
       JEQ  NAME2             Zero length, don't do match
       CB   R5,*R2+           Does length match?
```

```
       JNE  SGO              No
       SRL  R5,8             Move to right place
       LI   R6,NAMBUF        Point to NAMBUF
NAME1  CB   *R6+,*R2+        Is character correct?
       JNE  SGO              No
       DEC  R5               More to look at?
       JNE  NAME1            Yes
NAME2  INC  R1               Next version found
       MOV  R1,@SAVVER       Save version number
       MOV  R9,@SAVENT       Save entry address
       MOV  R12,@SAVCRU      Save CRU address
       BL   *R9              Match, call subroutine
       JMP  SGO              Not right version
       SBZ  0                Turn off ROM
       LWPI DLNKWS           Select DSRLNK workspace
       MOV  R9,R0            Point to flag byte in PAB
       BLWP @VSBR            Read flag byte
       SRL  R1,13            Just want the error flags
       JNE  IOERR            Error!
       RTWP
*
*** Error handling
*
NODSR  LWPI DLNKWS       Select DSRLNK workspace
LNKERR CLR  R1           Clear the error flags
IOERR  SWPB R1
       MOVB R1,*R13      Store error flags in calling R0
       SOCB @H20,R15     Indicate an error occurred
       RTWP              Return to caller
```

Disk 11. Graphics Design System

Version:
Author: J. Peter Hoddie
Requires: XB
Language: XB, AL
Updated:

Complete system for designing graphics screens In full color for use in your programs. Includes full documentation. Very early JPH program.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-11
Sectors total           = 360
Sectors used            = 96
Sectors available       = 262
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHAR         45 PROGRAM      >022 044
002 >006 LOAD          3 PROGRAM      >079 002
003 >003 READ-THIS    22 DIS/VAR 80   >04e 021
004 >005 SAMPLE       18 INT/FIX128   >068 017
005 >007 WRITER        8 PROGRAM      >07b 007
```

Disk 11. Contents of File READ-THIS

SCREEN CHARACTER DEFINITION PROGRAM

BY J. PETER HODDIE

The following instructions will help you in designing graphics screens to which can be uploaded to the Boston Computer Society Bulletin Board Service or used in your own programs.

System

Extended basic, Memory expansion, Disk drive, Joystick optional.

Start up

1. Make sure all memory has been cleared from memory, if you are not sure, start the program from the main memory screen.

2. When the program starts up you are at the MAIN SCREEN, Screen #1. The cursor is at the "HOME" position in the center of the screen. The cursor also indicates where the next graphics character will be placed.

3. At this point you may place graphics characters in as many screen locations as you wish. The cursor may be moved around the screen by the arrow keys, (**FCTN** key not required) or by pressing **J** for joystick and using the joystick. To return to keyboard press **K**.

Main Screen Key Presses

B	Set background color, (screen color).
C	Set color of character sets, foreground and background for each set.
D	Move cursor to left. (FCTN not needed.)
E	Move cursor up. (FCTN not needed.)
G	Turn grid on and off.
H	Recall character by ASCII code, enter definition in HEX code.
J	Toggle joystick on for moving cursor.
K	Toggle keyboard on for moving cursor.
L	Leave a character picked up at the current cursor position. (See P for pickup.)

N	Go to Character Definition Screen to define a new character.
O	Loads character presently under cursor.
P	Pickup character presently under cursor.
R	Recall and place at present cursor a predefined character, recalled by ASCII code.
T	Start TEXT mode for placing text at current cursor position, leave TEXT mode by pressing ENTER.
. (period)	Returns cursor to HOME position in the middle of the screen.
Enter	Returns to GRAPHICS mode from TEXT mode.
#1	Recalls screen #1
#2	Recalls screen #2
#3	Recalls screen #3
#4	Recalls screen #4
FCTN 1	Shows character set definitions.
FCTN 2	Catalogs drive #1
FCTN 3	Present Status Screen
FCTN 5	Load an old screen from disk.
FCTN 7	Instructions screen.
FCTN 9	Save screen at main screen to disk.

Character Definition Screen

Defining characters:

Define the characters by positioning the cursor on the 8 × 8 grid using the arrow keys. Press 1 to fill in a pixel, and 0 to leave a pixel blank. The cursor moves to the right after each square, but can be overridden by the use of the arrow keys.

Character Definition Screen Key Presses

C	Clears character grid.
F	Flips the character over.
I	Inverts the image (black is white and white becomes black.
K	Confirms a pictorial view as a single character and as a group of the same character. Press Q to assign the character an ASCII code.
L	Puts all black pixels on to the end of that row.
O	Recalls a previously defined character to the grid by using its ASCII code.
Q	Assign the character to an ASCII code and returns to Main Screen, with that character under the cursor. The character assigned is the character last confirmed by the K key at the Define Character Screen.
R	Rotates the character by 90 degrees clockwise.

The WRITE Program

The WRITER Program reads a data file saved by **FCTN 9** in the Screen Builder Program, and from the data, creates a Basic program on disk. This generated program is always stored on disk one under the filename "DSK1.HC" in Extended Basic's MERGE format. The program may be loaded and run, or incorporated into other programs. To load the generated program, type the command 'MERGE "DSK1.HC" ', not 'OLD DSK1.HC'. However, once you have loaded the program using the MERGE command, it may be saved, edited, renumbered, etc. just like any other console Basic program. You may want to add a statement such as 9000 GOTO 9000 at the end of your program so it will not end and have the graphics disappear.

There is a sample file on the disk called SAMPLE. It is a data file that can be user by CHAR or WRITER and is the basis of the first graphics display shown on the BCS TI99 BBSs.

Note: If you are designing characters to be sent through TEII for bulletin a board do not redefine characters above number #127. It may cause problems. The program uses some of the lower defined characters for it's use.

Disk 12. Masscopy

Version: 325 *Author*: Steve Lawless
Requires: EA *Language*: AL *Updated*: 09/16/86

Latest version of this versatile backup utility. Handles all disk formats, copies to two disks at once, supports 128K card, and more. Disk also includes a disassembler, an article on the disassembler to get you started, a program to quickly initialize disks, utilities for the Foundation 128K card, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = MASSCOPY
Sectors total           = 360
Sectors used            = 334
Sectors available       = 24
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00c D/ARTICLE    26 DIS/VAR 80   >10c 025
002 >00d DASSEM       29 DIS/FIX 80   >125 028
003 >00e DASSEM/DOC   33 DIS/VAR 80   >141 032
004 >00b DISK-INIT    13 DIS/FIX 80   >100 012
005 >00f LOAD          2 PROGRAM      >161 001
006 >002 MASSCOPY     11 DIS/FIX 80   >022 010
007 >003 MC/DOC       32 DIS/VAR 80   >02c 031
008 >004 MC3          29 PROGRAM      >04b 028
009 >005 MC3-1DISK    29 PROGRAM      >067 028
010 >006 MCINST       31 PROGRAM      >083 030
011 >007 TRANSX       35 DIS/FIX 80   >0a1 034
012 >00a WRITER       21 DIS/FIX 80 Y >0ec 020
013 >008 WTR128       28 DIS/FIX 80   >0c3 027
014 >009 WTR128/DOC   15 DIS/VAR 80   >0de 014
```

Disk 12. Contents of File D/ARTICLE

USING A TI DISASSEMBLER

By Greg Knightes

After learning something about assembly language on the TI 99/4A computer, it was natural to hear about these programs called disassemblers. They are programs that take a set of instructions (program) in machine format (1's and 0's) and constructs the assembly language program, that when assembled, produces those instructions. Most of us are familiar with the computer taking a BASIC program, interpreting it and then running that BASIC program. Well, disassembling a program is like going backwards. Here are hopefully some helpful hints that will help you get started using a disasembler.

First of all, the better disassemblers are written in assembly language (of course!). This means that to best use these is to obtain an Editor/Assembler cartridge. Using the Load and Run option of the E/A cartridge, load THE PROGRAM THAT YOU WANT TO DISASSEMBLE! When that is done loading, load the Disassembler that you will be using when it says FILENAME. At this point, you may get a multiple definition error. That's OK. It means that the program that you want to disassemble and the disassembler have the same program name in their DEF statements. Press **ENTER** and for PROGRAM NAME, enter the name of the disassembler program and press **ENTER**.

The disassembler that I have allows a memory dump, which prints values at memory locations that you provide. This is helpful for viewing the contents of the REF/DEF table to find out where your program is located. The REF/DEF table starts at address >3FFF and works it's way down to >3000. (the '>' means that the number is in hexadecimal or base 16 notation. More on this later.) If you put in these values as start (>3000) and end (>3FFF) addresses, then the disassembler should print out the name of the programs that are available to be used which should include your program. You may be able to get a printout of the table or copy it off of the screen.

The program that I have (DASEM) prints out 5 columns of numbers and a column of names. The first number is the address where the program name is found in the REF/DEF table. I don't know what the 2nd, 3d and fourth columns represent but the last column of numbers are the address of the start of the program. This is the address that you want to start disassembly from. The ending address is found by looking at the starting address of the program listed ABOVE the program that you will be disassembling. Write these addresses down so that you will remember the, figuring out how to find the starting and ending addresses was the one problem that I had with using the disassemblers.

Now TURN ON OUTPUT DEVICE if you want the output to go to your printer or leave it off if you just want to view it on your screen. You will then see the code of the program that you are disassembling. However, it will not quite look like the source code when it was originally written. There will be no REF, DEF, statements, DATA, BYTE, TEXT, BSS, BES directives, etc. Also, there will be no labels to JMP or BLWP to. Instead, the address of where the computer is to find the instruction is used instead. For example, if your program uses a statement such as BLWP @VSBW, then you would see BLWP @>210C. >210C is the address where the VSBW routine resides. This routine, among others is already predefined so you don't need to EQU the name VSBW with >210C. You will also see the opcode for the instruction and the values of any data that the instruction is using. This is similar to the listing produced when you specify an output device name when you assemble the program.

Now that we have seen how you can start to use a disassembler, to make it easier to use, it would be helpful to know how to convert a number from hexadecimal to base ten and back. The following explanation can be used to convert from one base to another very easily.

To convert from hexadecimal to decimal, do the following. Say the number you want to convert is >30FA. First, notice that there are four digits in the hexadecimal number. Also, look at the position of the digits within that number. Each place has significance. The 'A' has a value of 10 in base ten, the 'F' has a value of 15*16**1, the value of the '0' is 0*16**2 and the '3' has a value of 3*16**3. Do the multiplication and add the results together and you have the answer in base 10. The equations are determined by a couple of factors. First, Choose what digit that you want to figure out an equivalent value for. Then multiply it by the base of the number system that the digit is in raised to the power of the number of digit to the RIGHT of the digit you are looking at. Let's examine the '3' more closely. First we write it down. 3. Then we multiply it by the base which is 16. We now have 3*16. There are three digits to the right of the three so we raise the 16 to the 3rd power. This leaves us with 3*16**3. To evaluate this expression, we figure out what 16**3 is equal to first. 16**3=4096. Multiply that by 3 and we end up with 12288, so 3*16**3=12288. I leave it up to you to figure out how to get the correct answer of 12554. Just because the TI computer uses only numbers up to >FFFF, that doesn't mean that that is high as you can go. >F0A455011BC is a perfectly legal base sixteen number!

I hope that this has been some help to you in your quest to use a disasembler and base 16 number conversions. As I learn more about disasemblers, hopefully I will get a chance to improve this article and make it more useful to you.

Disk 12. Contents of File DASSEM/DOC

Text as appeared in the December 1984 Pomona Valley 99ers' Computer Group newsletter.

DISASSEMBLER HELP

by DOUG WARREN

In the past, some of you have asked about the TI-99/4A Disassembler program (named DASSEM) which is in the club library. We thought we would take some time here to explain what the program is and how it is used. There are two versions of the Disassembler and we will examine only Ver. 1.1 which corrects some mistakes occurring in Ver. 1.0.

A Disassembler is simply a program that converts machine code, the computer's own "language", into assembly language which is easier to read and understand. This allows us to examine coding that resides in lower console ROM, DSR ROMs in the expansion cards, or object code from a program whose source code, for example, has accidentally been destroyed.

The Disassembler is easy to load and has several features available to the user. Loading the program is done by selecting the "LOAD AND RUN" option of either the Editor/Assembler or the Mini Memory cartridge. After selecting the option, just type in DSK1.DASSEM for the filename (assuming the diskette containing the Disassembler is in drive one), press enter, and loading of the program should begin. After loading, simply press enter when another filename is requested. Now "PROGRAM NAME?" will appear and you can enter the name of the Disassembler program; DASSEM. If everything is working properly then the menu selection should be on the screen at this time.

There are six options to choose from so we will take a look at each one individually:

1-OP CODES

> This option disassembles machine code into assembly language. A start and finish address in HEX will be requested of the user. Pressing the space bar starts and stops execution to allow scanning of code.

2-MEMORY DUMP

> Selecting memory dump displays the HEX values and their ASCII representation of memory locations requested by the user. Again, the space bar starts and stops execution of this option.

3-MEMORY OFFSET->0000

This selection allows you to move a block of code to another memory location and fool the program into thinking it had never been moved. What is that good for? An example would be the disassembly of DSR ROM code. With the TI Debugger we can examine DSR ROMs by turning a specific card on and addressing memory space >4000 to >5FFF. With the card on we could also jump to the Disassembler and disassemble the DSR. A problem arises if we want a printout of our work. When the console outputs a record to the printer it expects all DSRs to be off otherwise the system locks up. We get around that by moving the DSR program into an unused part of expansion memory with the Debugger and turning the card off before jumping to the Disassembler. Then executing the memory offset option tells the Disassembler where the DSR was and where its new address is. The assembly code and memory dumps at the new location will now have correct addressing as if they were still in their old location.

4-CHARACTER BIAS NORMAL

This option adds >60 to the ASCII representation of the memory dump selection. This is useful for looking at text in an assembly program written for a BASIC screen.

5-TURN OUTPUT DEVICE ON

Little explanation is needed here. This option turns an output device on (such as a printer) or turns it off if it is already on. When selection 1 or 2 is running, all that appears on the screen will also be output to the output device.

6-EXIT

Exit and return to the cartridge environment.

A few other things need to mentioned about the program operation. Pressing **FCTN 9 (BACK)** will always return you to menu selection screen. Illegal entries of any type will do likewise. Besides exiting the Disassembler through the EXIT option, **FCTN = (QUIT)** will return you to the master title screen. This can only be done from the menu selection screen but be careful — open files are not closed with **FCTN =**.

Now let's try a quick example of how the program actually runs by looking at part of the MONITOR code in console ROM. Load the Disassembler as previously described and select the OP-CODES option. Put in the addresses 24 and 34 in response to the start and finish address prompts and assembly mnemonics will scroll across the screen. Return to the menu screen and this time select the MEMORY DUMP option. Put in 3F20 and 3FFF for the start and stop addresses. This will dump the DEF table which resides in lower expansion RAM. One more test, return to the menu and select the output device option putting in the device name for your printer (PIO or RS232). Now try a memory dump or op-code dump and you should receive a printout of everything that prints to the screen.

The Disassembler can be a very useful tool but it does take practice to learn how it's used. It is helpful in exploring your computer and is a companion to the Editor/Assembler package. Experiment with it and we're sure you will find many other applications for the Disassembler.

Disk 12. Contents of File MC/DOC

The following is the text of the MCINST file, the MASSCOPY instructions. To obtain your own copy of MASSCOPY you need to download 4 files from the public access area. These are: MASSCO. MC3. MCINST. and LOADMC. Once in your possession you will need to change the name of MASSCO to MASSCOPY. The other files (MC3 and MCINST) must be called these names only. That is if you want to work. LOADMC can be changed to LOAD to make it auto start from X-BASIC. You can alternately load MASSCOPY directly using the A/E option 3. In case you are interested MASSCOPY is an assembly language loader that loads the MC3 file. then divides it up in memory. MCINST is the instruction file called by MC3 if you ask for it. Anyhow here is the instruction file.

MASSCOPY will not work with the Myarc double density card. This card is not a true DD card. it only uses 16 sectors per track whereas the CorComp card uses 18.

MASSCOPY

by Steve Lawless

This program is being distributed by the Freeware concept. This concept is based on the old adage "people are willing to pay a reasonable price for a good program." This program cost only ten dollars. You are simply trying the program before you buy it. Although the program is copyrighted I have granted permission for distribution under this system. You are not permitted to sell this program for your own or others profit. You are also on your honor to pay me for the program. I have no recourse other than threaten to never release any other program I write, and to discourage others from supplying FREEWARE.

INSTRUCTIONS

You will be returned to the title screen in the program. Simply respond **N** to continue with the program.

Next MASSCOPY will display a relatively blank master screen and ask you to load the master disk (the disk you want to copy) in drive #1. Do this then press the space bar.

MASS COPY will then read in sector 0 and get the information it needs from it to know the disk parameters. The program will then fill in the master screen and display a bunch of options. These options allow you to change almost all of the master screen.

Below is a description of the screen

DISK SYSTEM 1=IN 1=OUT

This will allow you to tell MASS COPY that you have one or more disk drives. If only one drive switching will be necessary. the second part tells which drive is to be input (IN) and which is to be output (OUT).

MEMORY

MASS COPY will work with the regular 32K memory, or the FOUNDATION 128K memory. The 128K is especially convenient if you only have one disk.

COPIES OF ORIGINAL

If you have three disk drives you can make two copies of the original at once. The second copy is always placed drive #3.

I believe the disk parameters are obvious. They are read from your master disk. If the master disk has gibberish SS/SD 40 TR is default.

The disk name of the master disk is also displayed. If the disk is a proprietary disk asterisk are displayed. Below is a description of the options, an what they do.

Y and R are inverted color because these keys start the copying process.

Y Starts copying the entire disk. Everything is copied.

R Starts copying only the sectors the disk records as occupied. This option looks on sector 0 for the sector bit map. then it counts backward until it finds an occupied sector. Knowing the number of the last occupied sector it copies only this amount.

N Changes the number of disks to allow MASS COPY to use more than one disk drive. Also when this option is selected, Input is on drive 1, output on drive 2.

M Will allow MASS COPY to use the extra memory found in the 128K memory card. MASS COPY looks for this card at CRU address >1E00. If another device uses this address MASSCOPY might say you have it when you don't. Just reset it with this key.

A Alters disk I/O assignment. When pressed the input disk changes from 1 to 2 and the output from 2 to 1. If you have not pressed N and MASS COPY thinks you have only one disk, the key is ignored.

C If you have three disk drives, MASS COPY will allow you to make 2 copies at once. The second

copy is always on drive 3. Again, if you indicate a single drive system, this command is ignored.

H Will again show this instruction file. But, only if it can be found on the drive you first loaded MASS COPY. You will then be returned to the title screen. *not* where you were.

S D and T should be obvious.

Whenever the input parameter is changed the output is also changed. You will find this most convenient. Output only is changed by pressing FCTN S, D or T

IMPORTANT

If the input and output do not correspond exactly, sector 0 is automatically changed to account for this difference. For example: If you are copying a SS/SD disk to a DS/DD disk, when you catalog the copy, you will find you now have 1438 sectors available (tot) This will enable you to take advantage of the additional space available on the bigger disk.

WARNING

When going to a smaller disk, the final sectors are truncated. This means, if something was on them, it is gone.

Remember, if you don't have DS or DD capability MASS COPY cannot give it to you.

All commands (except Y and R) can be reversed by just pressing the key again.

You will notice the screen changing color. Blue indicates the program is waiting for your response. Green the program is reading, red writing

First on a two-disk system MASSCOPY will check to see if the copy disk is initialized properly. If not, it will ask your permission to do so. On a single disk system, MASSCOPY will simply ask if you want the copy disk initialized. If yes it then ask for the copy to be put in drive one to be initialized. MASSCOPY will then proceed with copy process.

When MASS COPY has completed all copying the program will display an error report if any read or write errors have occurred. This is only a gross error detection routine. But if any errors have occurred, especially write errors the fidelity of the copy cannot be guaranteed.

WARNING

This is a sector by sector copy all original contents of the copy disk are destroyed.

— END

Disk 12. Contents of File WTR128/DOC

128-WRITER

by Stephen Lawless

Introduction

128-WRITER has been designed to allow your TI-Writer* to take advantage of the extra memory found on the Foundation 128K card. On start-up the program will load the editor, formatter and character set to bank 3. The program then operates out of bank three acting much like your TI-Writer cartridge. When finished with the editor or formatter you will be returned to the loader program. This will allow you to switch between editor and formatter in seconds. MMM and disk directory functions are also included.

Important messages

1. As you see, this is version 1. I hope it is perfect, however little gremlins appear out of nowhere. If you find one feel free to let me know.

2. If you leave the editor for any reason ALWAYS save your text to DSKX or floppy. You CANNOT REENTER THE EDITOR IN PROGRESS. I hope to fix this in a later version.

3. DSKX file 3 is always deleted on startup. This is where I put the editor, formatter and program. The request to initialize DSKX only pertains to files 1 and 2.

Instructions

Place the disk containing WTR128, EDITA1, EDITA2, FORMA1, FORMA2, CHARA0 or1 on disk #1. Load the program using E/A Option 3 (Load and Run) DSK1.WTR128. The program is auto start. If you need a program name it is BEGIN. The loader program will load itself into bank 3 and display a title screen. You are then asked if you would like DSKX initialized. Remember file 3 is always reset. Once you respond Y or N, this program will begin loading the editor and formatter. When loading is complete a master title screen will be displayed. To perform the described function, simply press the number beside it. They should be obvious.

If you execute MMM, you will find file 3 occupied with a D/F 80 file called RESTART. This is a short program that will allow you to restart the loader program without reloading the TI-Writer files. You may need this with some Utilities.

At the present time you cannot reenter the editor as you left it. Therefor, ALways save your text to DSKX or floppy before you leave the editor. I hope to correct this later.

As I have said earlier, I hope to allow reentering the editor in progress. I also would like to add a D/V 80 DSKX to floppy transfer function. For now, you can use TRANSX. Lastly I plan to place the Spell checker in memory also. However, the dictionary will always be disk bound. It is too large for the 128K card.

* TI and TI-Writer are registered trademarks of the Texas Instruments Corporation. They are copyrighted by them. Please observe their rights.

— END

Disk 13. Disk Manager 1000

Version: 3.5 *Author*: Bruce Caron, Ralph Romans
Requires: XB or EA *Language*: AL *Updated*: 11/19/86

A great disk manager similar to the CorComp disk manager with extras such as recovery of deleted files. A TI classic.

```
dskdir. v2.0. 12-dec-96
Disk name               = DM10003/5
Sectors total           = 360
Sectors used            = 354
Sectors available       = 4
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01b DM10003/5    21 DIS/VAR 80   >022 010 >002 001 >012 009
002 >003 DMDOCPT1     54 DIS/VAR 80   >02c 053
003 >01c DMDOCPT2     70 DIS/VAR 80   >061 068 >004 001
004 >005 DMDOCPT3     94 DIS/VAR 80   >0a5 093
005 >006 DMDOCPT4     45 DIS/VAR 80   >102 044
006 >007 LOAD          7 PROGRAM      >12e 006
007 >008 MGR1         33 PROGRAM      >134 032
008 >01d MGR2         30 PROGRAM      >154 020 >009 009
```

Disk 13. Contents of File DM10003/5

DM1000 REVISION RECORD

Modified by Ralph Romans:

Ver 3.0 fixes to Ver 2.4:
- Incorrect file count when going from 'M' to 'C'.
- File copy would give you a bad copy if the file being copied was stored on the master disk as a non-continuous file and the size of the first segment was exactly 39 sectors with additional sectors in another segment on the disk.

Ver 3.1 fixes to Ver 3.0:
- File copy would give you a bad copy if the master file was a fractured file of exactly 39 sectors and the same filename was on the copy disk.
- When entering a file name in various modes, it was possible to mess it up.

Unfixed bugs in ver 3.1:
- Unable to display some dis/var 80 files that are full of control characters. Computer hangs up!

Ver 3.3:
- Changed defaults on sweep and disk initialization.
- Disk initialization works for Myarc and CorComp.
- Read/Write errors gets cleared after 1st use on disk copy.
- File 'MGR1' may now be called any name and all features of DM1000 will work.!! This will only work with TI controller and CorComp controller.
- The loader for Myarc controller is called LOADMY.
- During disk initialization menu, you can use the up arrow to go back to previous question.

Ver 3.4:
- Able to delete/move/copy 1-sector files
- Added 'up arrow active' notice when up arrow will take you back to previous question.

Ver 3.5:
- Able to type/print display VAR 80/FIXED 80 files while the file listing is on the screen by pressing a 'T' for type (display) file to screen or 'P' for print to list device with optional control codes sent to printer first. The 'P' and 'T' for print or type are only valid in the left most field.
- 'EOF" notice added in lower left corner of screen
- DISPLAY VAR 80/FIXED 80 Menu removed

DM1000 HELP

At Main Menu:

FCTN 3	Enter list device and control codes (default=PIO) option to save codes and screen colors to disk with '(PROGRAM NAME)' i.e. MGR1 name on it.

With Catalog on Screen:

FCTN 7	Print catalog to list device (default=PIO)

With File List on Screen:

FCTN 6	Proceed with given Commands

In Left Field:

C	Copy file
D	Delete file
M	Copy/Delete
T	Type DIS/FIX 80 file to screen
P	Print DIS/FIX 80 file to list device sending predefined control codes

When Typing file:

FCTN 8	Retype file
FCTN 9	Return to disk menu
FCTN 4	ABORT. An 'EOF' notice is given in lower left corner of screen when End Of File is reached

When Printing file:

FCTN 8	Reprint file
FCTN 9	Return to disk menu
FCTN 4	ABORT

In Right Field:

T	Unprot/temporary
U	Unprotect file
P	Protect file

Type over name to rename file.

With Disk Initialization Menu:

FCTN E	Go back to previous question

Valid at # of sides:

S/D	Density
Verify	Y/N

Run Image Program:

Note: File must exist or computer will hang up.

Xbasic 'CALL INIT' is done if Xbasic module installed.

'XB VDP' means make VDP registers the same as Xbasic.

'E/A VDP' means make registers the same as E/A.

Misc:

FCTN 1	Delete character
FCTN 2	Insert 1 character
FCTN 5	Go to main menu
FCTN 8	REDO selected menu
FCTN 9	Go back one menu level
FCTN S	Move cursor left
FCTN D	Move cursor right
FCTN E	Move cursor up
FCTN X	Move cursor down

Caution! With Myarc controller do not rename MGR1 or MGR2

— END

Disk 13. Contents of File DMDOCPT1

DISK MANAGER 1000

FILE UTILITIES

Copy File
Move File
Delete File
* Type(display) File
* Print File
Rename File
Modify File Protection
Recover Lost File
Run EA Program Image Files

DISK UTILITIES

Catalog Disk
Set Copy to Bit Map
Set Copy to Sector
Copy Disk
Sweep Disk
Rename Disk
Initialize Disk
Box Format

MISC UTILITIES

Install Disk Protection
Remove Disk Protection
Remove XB Protection
Change Screen/Text Color

Written by Bruce Caron, 1 May 1985 for

The Ottawa TI-99/4A Users' Group
Box 2144, Station D
Ottawa Ontario Canada
K1P 5W3

* Features altered in 3.5 Update By Ralph Romans of the O.U.G., Release 3.3, July 1986

TABLE OF CONTENTS

START UP INSTRUCTIONS

This utility program can execute a number of different WRITE operations on your diskettes. Read through the instructions to familiarize yourself with the various operations before inserting diskettes or you may inadvertently DESTROY part of a good diskette. You should always use a Write Protect tab on your Master diskettes.

Make a backup copy of the Disk Manager 1000 diskette and place it along with your other Masters, in case your work disk should ever get damaged.

EQUIPMENT REQUIRED

Firmware	Extended Basic module or Editor/Assembler module
Hardware	32K memory expansion Disk memory system 1 or more disk drives
Optional	RS232 card Printer

LOADING DISK MANAGER 1000

EXTENDED BASIC LOAD

> Insert the Disk Manager 1000 diskette in disk drive 1 and select Extended Basic from the TI menu screen. The disk contains a file that will Load and Run the Disk Manager 1000 program.

EDITOR/ASSEMBLER LOAD

> You can Load and Run the Disk Manager 1000 program from the Editor/Assembler module by selecting option 5, RUN PROGRAM FILE from the main menu screen, and entering DSK1.MGR1.

HORIZON RAMDISK LOAD

DM1000 has been the resident disk manager of Horizon Computer Ltd. since November 1985. Once all initial start-up routines for your RamDisk are performed you may use CALL DM to quickly access the program at any time in Basic or Extended Basic. (both upper and lower case work for the call statements).

DISK MANAGER 1000 QUICK REFERENCE GUIDE

FCTN	*KEY*	*DESCRIPTION*
DEL	1	Delete a character
INSERT	2	Insert a character
ERASE	3	Configure List Device
CLEAR	4	Halt Disk Drive I/O operation
BEGIN	5	Return to Disk Manager 1000 main menu screen.
PROC'D	6	Request EXECUTE COMMANDS Y/N prompt on File Utilities option 1
AID	7	Print Catalog to List Device
REDO	8	Move cursor to beginning of Selected Option. Re-enter inputs.
BACK	9	Back-up one level of menu each time it's depressed; till at level 1.
	0	Not Used
QUIT	=	Exit to Main Title Screen
	E	Move cursor up one field
	X	Move cursor down one field
	S	Move cursor left one character or back one field.
	D	Move cursor right one character or ahead one field.
CTRL	*KEY*	
	E	Move cursor back one page.
	X	Move cursor ahead one page.

FILE UTILITIES

1. COPY/MOVE/DELETE/TYPE/PRINT/RENAME/CHANGE FILE PROTECTION

The Disk Manager 1000 file utilities will enable you to perform all of the above functions on your disk files. Disk Manager 1000 will build a catalog of all files and programs on your diskette and by using a powerful screen editor it will allow you to enter all the file commands at the same time. Once the file commands have been entered, Disk Manager 1000 takes over and executes each function specified, without repetitive interruptions. The main exception here is with T and P operations. It has proved impractical to chain these two operations with the C, M and D ones, so it will do T and P operations as soon as the letter is entered into the field 1 area.

// COPY //	Copy files from one diskette to another.
// MOVE //	Move files from one diskette to another.
// DELETE //	Delete unprotected files.
// TYPE //	Display DV/DF-80 files to screen.
// PRINT //	Send DV/DF-80 files to printer.
// RENAME //	Rename unprotected files.
// CHANGE FILE PROTECTION //	Enable or Disable the file WRITE protection feature either permanently or temporarily.

2. RECOVER FILE

Should you ever happen to delete a file or program on a diskette, either accidentally or otherwise, this option will allow you to recover that file in its entirety. Disk Manager 1000 will scan a selected drive for the lost file and if found, it will restore the file by updating both the Directory Link Map, and the Volume Information Block (Disk Bit Map).

Note: Files that have been partially over written by other files, or files lost due to bad magnetic media are not recoverable with this option.

All of the utilities are self-prompting and incorporate extensive error checking. Disk Manager 1000 displays a message informing the user of each function that it is currently executing. Any error that may occur will cause a message to be displayed in plain everyday English.

Disk 13. Contents of File DMDOCPT2

FILE UTILITIES SCREEN EDITOR

The Screen Editor allows you to select, and enter, all your file commands at one time. The Screen Editor commands, and fields, are described below.

```
Screen ->   DSK1 : SAMPLE     + Free   127 Used 591
Header      CMD Filename   Size Type/No. 24 P <1/2>
            --- ---------- ---- ----------- -
Directory   N   A             7 PROGRAM     U
            N   ASORT/S      64 DIS/VAR 80  U
            N   BGRAF         7 PROGRAM     U COPY
            C   BGRAF/S      14 DIS/VAR 80  P MOVE
            N   BSCKEY/S     10 DIS/VAR 80  P
            M   BSORT/S      30 DIS/VAR 80  U   44
            N   BUBBLE/S     11 DIS/VAR 80  U
            D   CHARDF       26 DIS/FIX 127 T
            N   CRASH/S       6 DIS/VAR 80  P DELE
            N   DELOBJ        3 PROGRAM     U
            N   DISP          2 PROGRAM     U   56
            T   DISP/S       19 DIS/VAR 80  P
            P   DSRLNK/S     27 DIS/VAR 80  U
            ^
```

Note: A T or P in this field will be acted on at once. Therefore you should do all of your perusing of disk contents first and M, C, D, Rename and P, T U operations after reading or printing files.

EXAMPLE OF A FILE UTILITIES SCREEN

SCREEN HEADER

The first 2 lines of the Screen Editor, or Disk Catalog, is called the Screen Header. The Screen Header displays the following information about your diskette.

Current Drive in operation	DSK1
Current Disk Name	SAMPLE
Proprietary Disk Protection	+ = Yes, - = No
Amount of FREE and USED space	
Number of Files on Disk	Type/No. 24
Number of Pages in Directory	<1/2>

The screen will hold the information for 20 files at a time, additional files are stored in memory and are paged in and out with the CTRL E and CTRL X keys. The upper right corner displays which page is on display and how many pages are stored in memory. For example; <1/2> means page 1 of 2.

CMD FIELD

You will type in the CMD field to <C>OPY, <M>OVE or <D>ELETE a file. The "N" means NO operation in CMD field.

<C>OPY
When copying a file from one disk to another a check is made to verify that the destination drive has enough free space on it to accommodate the new file. If a file is TOO LARGE to be transferred, the Copy operation is HALTED, and 'File too large for Backup Disk' will be displayed. This is a reminder that the file has NOT been copied. Pressing any key will return you to the File Utilities Menu.

Should the Backup disk become FULL the Copy process is again HALTED, and 'Backup Disk Full. Press any Key.' will appear.

Attempting to Copy a file onto a WRITE PROTECTED file with the SAME NAME will also HALT operation and 'DUPLICATE FILE WRITE PROTECTED' will appear on your screen. Again; pressing any key will transfer control to File Utilities main menu.

<M>OVE
This command allows you to Copy a file from one disk to another just like the <C>opy command. Once the file has been copied, it is then DELETED from the Source disk, whether it is write protected or not.

<D>ELETE
Files that are not write protected are deleted with this command. Files that are write protected must be changed before they can be deleted.

Note: Should the Backup disk have insufficient space for a file, or if it becomes full, all commands left pending are aborted. Files that were successfully transferred to the Backup disk will remain there, and files that were moved will NOT BE DELETED from the Master disk.

<T>YPE
This operand replaces the Read DV/DF-80 option that was on earlier versions of DM1000 at the next higher level menu. It may only be used for DV-DF type files and its entry is only allowed by the program for these type files. As soon as you type the T in field one DM opens the file and writes it to your screen. EOF will appear in the lower left corner of your screen when the entire file has been displayed.

<P>RINT
This option allows you to dump a DV/DF-80 file to a printer. It uses the printer parameters you set using FCTN 3 at the level 1 menu of DM1000 (explained elsewhere). PIO with standard typeset is the default setting here.

FILENAME FIELD

Typing in this field will cause those files not protected to have their Filenames changed. Both upper and lowercase characters are accepted however the space and period are invalid characters.

Note: File Names with Lowercase characters are not accepted as valid file names when using TI Basic. All cursor movement keys are active in this field, including the **DELETE** and **INSERT** keys.

P FIELD (Write Protection)

Type in the P field to change a files protection. If a file is unprotected a "U" is displayed, for a protected file a "P" is displayed. To either protect or unprotect a file select and enter the appropriate character. If you want to temporarily change a file's protection in order to rename it just enter a "T".

CHANGING FIELDS

Moving from field to field is accomplished by using the cursor control keys. Hold the **FCTN** key down and use the arrow keys to move. Only valid characters unique to their field are accepted. The cursor will not enter the Size or Type field. Holding a key down for longer that 1 second will cause that key to auto-repeat until it is released.

EXECUTE FILE COMMANDS

Once all your commands have been selected pressing **FCTN 6** will display: 'Execute File Commands (Y/N)? N'. Pressing **Y** will start execution of the file commands selected. Pressing **N** or pressing **ENTER** will return you to the last field that the cursor was located in. Pressing **ENTER** when the cursor is at the last field has the same effect as pressing **FCTN 6**.

Disk Manager 1000 will keep you informed of it's status at all times by displaying the NAME OF THE OPERATION, and the NAME OF THE FILE, at the bottom of your screen. When Copying or Moving files it will also keep track of the number, and size, of files being transferred. It also automatically updates counters for both the Master and Backup disks. Copying files displays the following prompts.

```
Total Files to Copy :
Total Size of Files :
```

"Totals" Fields

Introduced in V.3.1 of DM1000 are the "Total" fields on the right side of your screen when the disk is cataloged in file utility mode. These fields labeled COPY/MOVE and DELE will keep track of the total sectors being operated upon on your master disk. You will notice that both totals will be updated if you use the Move option. This new feature will be a big help to those users that are trying to maximize the use of disk space by filling every sector on the copy disk.

ORDER OF OPERATION

The File Utility commands are executed in the following order.

1. Unprotect all files marked with a "U" or "T".
2. Delete all files marked with a "D".
3. Rename all files changed in Filename field.
4. Protect all files marked with a "P" or "T".
5. Copy all files marked with a "C" or "M".
6. Delete all files marked with a "M".

RECOVER FILE

This option allows you to recover a file or program which has been deleted from a diskette. Recovering a file should be done before performing any other operations on that disk to prevent the file from being over-written.

Selecting the Recover File option from the File Utilities menu will display the following prompts.

```
Recover File on Drive : 1

Enter Name of File:
```

Enter the number of the disk drive that has the lost file or press **ENTER** to accept the default drive number. When the drive has been selected the cursor will be located at the next prompt. Type in the name of the lost file and press **ENTER**. The disk drive will start up and 'SEARCHING DISK' will be displayed. If the lost file is found and is intact, then your screen will display: 'RE-BUILDING LOST FILE'. Once the file has been restored to the disk, 'FILE RECOVERED' is displayed. Pressing any key at this time will return you to the File Utilities main menu.

If the lost file cannot be found, or if the lost file has been over-written the following messages will be displayed.

```
FILE NOT FOUND or
FILE HAS BEEN OVER-WRITTEN
```

Recovering files, or portions of a file, that have been over written or destroyed by a damaged diskette is beyond the scope of this utility.

READ DV/DF-80 FILES

Replaced with the 'T' option in file utilities.

RUN E/A PROGRAM IMAGE FILES

Two new options have been added to the File Utility sub-menu in V.3.1 of the package

```
4. Run Program Image...XB VDP

5. Run Program Image...EA VDP
```

These new options will only work if you've loaded DM1000 in the XB environment as they require a LOADER that is imbedded within the Extended Basic LOAD file accompanying your disk.

The two new options differ as follows:

> Option 4(...XB VDP) leaves the VDP registers as they should be for proper operation in the Extended Basic environment
>
> Option 5(...EA VDP) adds an offset to the VDP registers. This option will be the one to select if the program image file normally operates using option 5 of the EA only.

IMPORTANT NOTE:

The Extended Basic LOAD file accompanying this disk has one other exciting feature built into it. It will remember the number of the last disk drive you accessed!!! If you want to operated the manager from drive 2, 3 or 4 simply place the disk in the appropriate drive. . . Type in OLD DSKX.LOAD (Where X is the Drive in which the Diskette resides) then type RUN. The program will boot the MGR1 file directly from the drive containing the Manager diskette!

SUMMARY OF FILE COMMANDS

* CMD FIELD	**C**	copy file
	M	move file
	D	delete file
	T	display file
	P	print file
	N	no operation
P FIELD	**P**	protect file
	U	unprotect file
	T	temporarily unprotect file then re-protect file
FILENAME FIELD	Type in new file name	
** INITIALIZATION	**FCTN E** (arrow up) is now active when initializing diskettes in the file utilities sub-system. Should you make an incorrect entry and press **ENTER** too fast you may now go back up and correct your mistake using **FCTN E** rather than starting all over.	

Note: The CMD functions now work properly with 1 sector files that have been inadvertently placed on your disk as of release of V.3.5 of DM1000.

Note: This new feature added in V.3.3 will allow users to easily correct inaccurate input in DISKNAME; # of SIDES; S/D DENSITY; and VERIFY(Y/N) fields. Prior to this release if you inadvertently entered incorrect data you would have to go through the entire selection process from the 2nd level menu again to correct the erroneous entry.

FILE UTILITY ERRORS

For a complete description of the error messages check the section entitled ERROR MESSAGES.

Page 9

Disk 13. Contents of File DMDOCPT3

DISK UTILITIES

The Disk Manager 1000, Disk Utilities section, allows you to perform the following functions on your diskettes.

1. CATALOG
Reads the disk directory and provides you with a list of all files and programs on a diskette, along with the write protection status, and the file size. The files are listed to the screen, output can be redirected to a printer.

2. COPY DISK
Copies an entire diskette from one to another, on a sector by sector copy basis. This copy function will copy a SSSD diskette in four passes or less, depending on whether the "Bit Map" or "Sector" copy option is selected.

Note: This sector by sector copy process will completely over-write any data on the Backup disk. To prevent data from being lost on the Backup disk, use the Copy function in the File Utilities section.

3. SWEEP DISK
This function will restore Sectors 0 and 1 on a Selected Disk to that of a standard, freshly formatted disk. Cleans your disk without having to re-initialize it.

4. RENAME
Allows you to change the name of a diskette.

5. INITIALIZE
Disk Manager 1000 enables you to initialize a diskette in single or double density with both single and double sided drives. This function is configured for 40 track disk drives only, 35 track initialization is not supported.

Note: In order to initialize a disk double density and double sided you must have a Disk Controller and a Disk Drive that support these formats.

Currently only Disk controller cards manufactured by Myarc and CorComp support the double density feature.

6. BOX FORMAT
This function will allow you to initialize an infinite number of diskettes without having to re-enter the disk name and initialization parameters each time.

CATALOG DISK

To Catalog a disk; select option 1 from the Disk Utilities menu. The following prompt will be displayed.

```
Drive No.: 1
Diskname :
Free 0000  Used 0000
```

Enter the number of the drive that contains the diskette to be cataloged. Disk Manager will display the Name of the diskette along with its amount of FREE and USED space, and then proceed to build a directory of all files on the diskette.

When the directory of all files is completed the files are displayed 20 to a page (See File Utilities/Screen Editor). If the directory contains more that 20 files you can page through the other files by pressing the **CTRL E** and **CTRL X** keys.

To re-direct the output to a printer press **FCTN 7**. Before you start listing catalogs to a printer, you may have to configure the list device. Instructions on how to configure Disk Manager 1000 to your system are given in the section titled ; Configure List Device.

COPY DISK

Selection of option 2 on the disk utilities menu will bring to the screen the following sub-menu:

1. Set Disk copy "BIT MAP"
2. Set Disk Copy "SECTOR"

We have selected Option 2 as the default option as we feel most people who are unsure of the protections installed on a disk will select this option anyway. If you are copying a number of diskettes and wish to skip this sub-menu; you may do so by pressing **FCTN 8** after each copy rather than **FCTN 9**. If you do decide to change copy modes **FCTN 9** will revert you back to this menu. A short description of the two options available follows:

> Option 1 is used in conjunction with the "Copy Disk" Utility, on the Disk Utilities menu. With the BIT MAP option set, only those sectors that are mapped as USED will be copied to the Backup diskette.
>
> Option 2 is used in conjunction with the "Copy Disk" Utility, on the Disk Utilities menu. With the SECTOR option set, each and every sector on the Master diskette will be copied to the Backup diskette. This will enable you to make an exact duplicate of the Master diskette.

Note: Techniques for removing proprietary protection from Diskettes and Extended Basic programs have been published in various magazines and books for the TI. As these are no longer considered to be "SECRET" information, I have included them as utilities, which you may find useful.

When you have selected your desired mode of operation the following prompts are displayed.

```
      COPY DISK "BIT MAP"

      Master Disk
      Drive No.: 1
      Sector  0000

      Backup Disk
      Drive No.: 2
      Sector  0000

WARNING: Backup Disk will be erased.
```

Enter the drive number of the Master and Backup disk drives, single drive users will be prompted when to change disks. Once the number of the Backup drive has been entered, Disk Manager 1000 will start to read in, and write out the sectors in 90 sector chunks, until the diskette has been completely copied. The Sector counters will start incrementing as the diskettes are being read from and written to.

Should the Backup disk not be initialized, or be in the wrong format, then Disk Manager 1000 will initialize the diskette to the correct format and continue on with the copy process.

When the disk copy process is finished the number of READ and WRITE errors will be displayed. A READ error indicates a bad sector on the Master Diskette. A WRITE error indicates a bad sector on the Backup Diskette.

Note: These READ/Write Error counters will now properly reset each time you perform a disk copy(V.3.3+).

SWEEP DISK

Selecting option 3 from the Disk Utilities menu will allow you to "fresh format" a diskette without having to initialize it again. This process only takes a second. The following prompts are displayed.

```
      Drive No.: 1
      Diskname :
      Free 0000  Used 0000

      Sweep Disk (Y/N)? N
```

Note: With V.3.3 this default is changed from Y to N so that you must consciously press Y to perform the sweep.

Enter the name of the drive that contains the diskette. The Name of the diskette and the amount of FREE and USED space will be displayed. To Sweep the Disk select the default entry of "Y". Pressing "N" will return you to the Sweep Disk drive number field.

RENAME DISK

To change the name of a diskette select option 4 from the Disk Utilities menu. The following prompt is displayed.

```
Drive No.: 1
Diskname :
Free 0000  Used 0000

New Disk Name :
```

Enter the number of the drive that contains the diskette that will have its name changed. The Old disk name and the amount of FREE and USED space on the diskette will be displayed. Enter the New disk name and press **ENTER**. The name of the diskette will be changed.

INITIALIZE DISK

To initialize a diskette select option 5 from the Disk Utilities menu. The following prompts are displayed.

```
Drive No.: 1
Diskname :
Free 0000  Used 0000

No. Sides : 2
Density   : S
Verify Y/N: N <0000>
```

Entering the number of the drive that has the disk to be initialized will cause Disk Manager 1000 to see if the diskette is already initialized, if so, it will display the disk name and the amount of FREE/USED space it contains. It will also display this prompt.

```
INITIALIZE DISK (Y/N)? Y
```

If you answer "Y" to the prompt, or if the disk is not initialized, you will be prompted to enter the rest of the initialization parameters.

```
Diskname  : Enter name of diskette
No. Sides : 2 Number of sides 1 or 2
Density   : S Density <S>ingle or <D>ouble
Verify    : N Verify all sectors <Y>es or <N>o
```

Answering "N" to verify option will reduce initialization time by half. However should the diskette have defective sectors on it, they will not be mapped as defective. If your drives are reliable and you haven't had problems with any diskettes then this option will save you time.

Note: With the release of V.3.3 of DM1000 you'll find that you can now edit the input fields of any 'initialize' process within the package(i.e.: in both file utilities and disk utilities). This is to allow a user to very easily and conveniently change incorrect entries. We have also changed the default entry in the # of Sides field from 1 to 2 as the majority of DM1000 users(at least those that have made fairware contributions) seem to have Double Sided drives.

BOX FORMAT

Option 6 from the Disk Utilities menu will allow you to initialize an infinite number of diskettes without having to re-enter all the initialization parameters.

Enter the Diskname and initialization parameters as you would with the previous option. After the first diskette has been initialized the following prompt is displayed.

Insert Next Disk. Press **ENTER**.

Insert the next diskette to be initialized and press the **ENTER** key. The following prompt will be displayed.

```
Initializing Next Disk.
```

This process will continue until you exit this function by pressing **FCTN 9 (BACK)** or **FCTN 5 (BEGIN)**.

DISK UTILITY ERRORS

For a complete description of the error messages check the section entitled ERROR MESSAGES.

MISC UTILITIES

The Disk Manager 1000, Misc Utilities section, allow you to modify proprietary protection flags and to change Disk Copy parameters.

1. INSTALL DISK PROTECTION

This option will allow you to set the proprietary disk protection flag that is located on sector 0 of every diskette. Setting this flag will prevent diskettes from being copied with the TI Disk Manager module.

2. REMOVE DISK PROTECTION

This option will allow you to remove the proprietary disk protection flag that may be set on sector 0 of a diskette. Removing this flag will allow a diskette to be copied with the TI Disk Manager module.

3. REMOVE XB PROTECTION

This option allows you to remove the proprietary Extended Basic protection from a program image file. This option may be useful for modifying protected programs, and for making Backup copies of protected programs.

INSTALL DISK PROTECTION

Selecting option 1, Install Disk Protection , will allow you to set the proprietary protection flag that is located on sector 0, offset address >0010 of a diskette. The following prompts will be displayed.

```
Drive No: 1
Diskname:
Free 0000 Used 0000
```

Entering the number of the disk drive that contains the diskette to be protected will cause the diskette NAME and the amount of FREE and USED space to be displayed, along with a capital "P" right after the Disk name. The "P" indicates that the proprietary protection flag has been set.

REMOVE DISK PROTECTION

Selecting option 2, Remove Disk Protection , will allow you to remove the proprietary protection flag that is located on sector 0, offset address >0010 of a diskette. The following prompts will be displayed.

```
Drive No: 1
Diskname:
Free 0000 Used 0000
```

Entering the number of the disk drive that contains the diskette to be unprotected will cause the diskette NAME and the amount of FREE and USED space to be displayed. The absence of the letter "P" right after the Disk name indicates that the proprietary protection flag has been removed.

REMOVE XB PROTECTION

Option 3; allows you to change the proprietary XB protection flags that prevent an Extended Basic program from being copied or listed. Select this option and the following prompts are displayed.

```
XB PROGRAM ON DRIVE : 1
ENTER XB PROGRAM NAME:
```

Enter the number of the drive that contains the XB program, followed by the name of the XB program. The disk drive will start, and Disk Manager 1000 will search for the location of the proprietary protection flags. When the XB program flags have been changed the following message is displayed.

```
PROGRAM IS UNPROTECTED
```

Pressing any key will return you to the Misc Utilities menu.

Should you attempt to change the flags of a file that is not an Extended Basic program the following error message is displayed.

```
NOT IN PROGRAM FORMAT
```

If the program cannot be found on the diskette specified the following message is displayed.

```
FILE NOT FOUND
```

Pressing any key will return you to the Misc Utilities menu.

Note: This option should NEVER be used on anything but an Extended Basic program. A directory entry for an Extended Basic program is identical to a directory entry for an assembly language program image file. Proprietary protection flags are NOT located on the Directory entry, but are imbedded in the program file itself. Using this option on an assembly language program image file may render that file unuseable.

LIST DEVICE

This function will allow you to customize the Disk Manager 1000 program to match the requirements of your particular printer. You will be able to enter your RS232 or PIO options; you can also select whether or not to send control codes to your printer. Once all selections have been made you have the option of saving them to your Disk Manager 1000 disk.

CONFIGURE LIST DEVICE

In order to configure your system, first go to the Disk Manager 1000 main menu and press **FCTN 3** (ERASE). The screen will be cleared and the following prompt displayed.

```
Enter List Device:
```

Enter the name of your printer driver and options, for example here are a few valid entries:

```
RS232.BA=1200.TW. or PIO.LF
```

After entering your List Device the following prompt is displayed.

```
Send Control Codes (Y/N):
```

This option will allow you to enter any control codes that you would like sent to your printer. Users with Dot Matrix printers will be able to send control codes to select their preferred type fonts, and print styles.

Pressing the **Y** key will display the prompt to allow you to enter your string of control codes. The maximum length of your string is 30 control codes or ASCII characters.

Enter Control Codes: Ex. 27 83 01 *

Enter the decimal value of the control code and separate each code number with 1 space. When you have finished entering your control codes enter 1 more space followed by an asterisk "*" and press **ENTER**. The asterisk is the control code terminator.

At this point 1 of 2 things will happen. The prompt to save the options to disk will be displayed, or the line will go blank and you will have to re-enter your control codes. If the line goes blank it is because the format was not correct or the codes were not decimal values.

If you have entered your control codes correctly, or if you had answered "N" to the enter control code prompt then the following option is displayed.

```
Save to Disk (Y/N):
```

If you want to save the options to disk then press **Y** this prompt will be displayed.

```
Insert Disk Manager 1000 Disk
In Drive #1 and press ENTER.
```

Follow the instructions to save the list device parameters and control codes to disk. If you choose not to save the options to disk enter **N**.

Whenever you first load the Disk Manager 1000 program it checks to see if there is a configuration file on the disk. If there is, then it will be loaded into the program.

The configuration file is imbedded in the directory entry for the Disk Manager program.

EXAMPLE

```
Enter List Device :
PIO

Send Control Codes (Y/N): Y

Enter Control Codes : (Note 1)
15 27 71 27 83 01 27 48 *

Save to Disk (Y/N): Y
```

Note: Gemini printer control codes. Condensed // Double Strike on // Subscript // 8 LF/inch

Disk 13. Contents of File DMDOCPT4

ERROR MESSAGES

The Disk Manager 1000 program is unique in that all error messages are displayed in English, instead of returning a number that the user must refer to in a manual. The Disk Manager 1000 program handles both Hardware and Software errors in the same way.

HARDWARE ERRORS are those that occurred during execution of a device I/O operation, such as attempting to catalog a drive with no diskette in it. SOFTWARE ERRORS are those that occurred as a feature or restriction of the program, such as trying to copy a file onto a diskette that has already been filled with programs.

WHAT HAPPENS?

When a hardware or software error has been detected the command currently being executed is HALTED, an error message is displayed, and any commands left pending are aborted.

WHY?

Attempting to recover from a hardware or software error is extremely difficult if not impossible. One cannot predict what the user will do when confronted with an error, and not all TI computers return the same error code for the same error.

The SOFTWARE ERROR MESSAGES are documented throughout this manual and all are self-explanatory.

The HARDWARE ERROR MESSAGES which are listed below are also self-explanatory with the exception of the DEVICE ERROR message. This is the "CATCH ALL" message for almost anything that goes wrong with a TI-99/4A.

```
No Diskette in Drive

Disk Write Protected

Disk Not Initialized

Device Error
```

APPENDIX A

Whenever an error message is displayed on the screen just press any key to return you to either the main menu or one of the sub menus, depending on where the error occurred.

WARNING: Single drive users must be careful to insert the correct disk when prompted. Failure to do so may cause the contents of the diskette to be over-written, rendering it completely useless.

NEVER CHANGE DISKS WITHOUT FIRST RETURNING TO ONE OF THE MENU'S, OR UNLESS PROMPTED TO CHANGE DISKS.

DM1000 IS A FREEWARE PRODUCT DISTRIBUTED BY THE OTTAWA TIUG NO OTHER GROUP; ORGANIZATION OR COMPANY MAY DISTRIBUTE THIS PRODUCT FOR GAIN. FREE DISTRIBUTION OR DISTRIBUTION AT COST THROUGH USER GROUP LIBRARIES OR EXCHANGES IS ENCOURAGED.

BRUCE CARON HAS SOLD ALL RIGHTS; BUT NOT OWNERSHIP; OF THIS PACKAGE TO THE OTTAWA TI-99/4A USER GROUP. DONATIONS; IF ANY; SHOULD BE MADE TO THE TREASURER OF THE GROUP.

Questions or comments concerning the Disk Manager 1000 program can be directed to me at the following address.

Bob Boone c/o

The Ottawa TI-99/4A Users's Group
P.O. Box 2144
Station "D"
Ottawa, Ont.
K1P 5W3

NOTES ON DM-1000 VERSION 3.3

Maintenance and upkeep for this package is now being done by Ralph Romans of our Ottawa TI-99/4A User Group. Bruce Caron has sold his TI and left our immediate area too. Comments re his program are being forwarded to him through us in Ottawa, Canada.

A. Versions released prior to V.3.3 will not correctly reset the READ/WRITE Error counter in Disk copying operations. We have corrected this problem with this release.

B. DM-1000 now works with all three of the major controllers on the market (TI, CorComp and Myarc. When initially run up it peaks into the controller card, sets a flag depending on which card it finds then initializes disks etc. in the format required by the card in question.

C. Users of the CorComp or TI disk controllers now have the option to rename the DM-1000 program files if they so wish. Prior to release of V.3.3 users had to leave the files with names MGR1 and MGR2 so that the program would know where to save configuration info on the disk. This limitation no longer exists unless you have a Myarc controller in your system.

D. The SWEEP DISK(Y/N) prompt has had its default entry changed from Y to N. You will now have to deliberately select the Y to perform this operation. — Additional safety.

E. More users with DS drives have responded with donations for our efforts. For this reason we've changed defaults where applicable to favor double sided operation rather than single sided ones.

F. The big change on this update is the ability to screen edit inputs of all initialization processes within the package by using **FCTN E** to switch input fields!

THANK YOU ALL FOR THE FANTASTIC FEEDBACK WE HAVE BEEN GETTING CONCERNING THIS PROGRAM AND OUR 2D-GRAPHICS PACKAGE.

BOB BOONE
INTER-GROUP CO-ORDINATOR
OTTAWA TI-99/4A USER GROUP
OTTAWA, ONTARIO, CANADA

APPENDIX B

ADDENDUM TO DISK MANAGER 1000

After Disk Manager 1000 was released for testing it was realized that a few changes were needed, and also a few things needed better explanation.

LOWERCASE LETTERS

As DISK and FILE NAMES containing lowercase characters are not considered valid when using TI Basic, any lowercase characters that are typed will be converted to uppercase.

HALTING DISK DRIVE I/O

To HALT the disk drives during execution of the Disk Manager program, press and hold FCTN 4 until the following message is displayed.

```
USER HALTED I/O
```

Certain operations are under control of the Disk Controller ROM so you may have to keep **FCTN 4** pressed for a few seconds till program control returns to Disk Manager 1000.

DISK COPY UTILITY

The Disk Copy Utility will always initialize the Backup diskette to the same format as the Master diskette. If you are copying from one format to another use the File Copy Utility.

LIST DEVICE

Entering an invalid device name will cause the Disk CATALOG function to behave erratically. To correct this situation simply re-enter a VALID list device and parameters. You can also list a catalog to disk by entering the drive number and filename as a list device, for example the following will create a DISPLAY,VARIABLE 80 file of a catalog listing under the name TEST.

```
DSK1.TEST
```

LISTING A CATALOG

Listing a catalog is only possible from the CATALOG DISK Utility, you CANNOT list out a catalog from the File Utilities section.

To stop the printing of a catalog to your list device just press and hold any key, except QUIT, the following message will be displayed.

```
USER HALTED I/O
```

IMPORTANT INFORMATION

Disk Manager 1000 was written completely from scratch and does not operate the same as other disk manager programs. Although Disk Manager 1000 may look similar to the CorComp disk manager, it is a completely different program with different features.

As Disk Manager 1000 makes use of every free area of memory not taken up by itself, other programs cannot be co-located in memory at the same time.

The Disk Manager 1000 program was written entirely by Bruce Caron with the exception of the Extended Basic loader. The Extended Basic PROGRAM IMAGE loader was written and given by Mr. Art Green, also of the Ottawa TI-99/4A User Group.

IN CASE OF DIFFICULTY

When all else fails, read these instructions.

Disk 14. XB-Forth

Version: *Author*:
Requires: XB *Language*: *Updated*:

A version of TI-Forth set up to load from XB.

```
dskdir. v2.0. 12-dec-96
Disk name               = FORTH-LOAD
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 FORTHSAVE    39 PROGRAM      >022 038
002 >003 FORTHXB       8 DIS/FIX 80   >048 007
003 >004 FSYS-SCRNS  302 DIS/FIX128   >04f 281 >005 020
004 >019 LOAD          2 PROGRAM      >01a 001
005 >01b _DSRLNK       7 DIS/FIX 80   >01c 006
```

Disk 15. Labeller

Version: *Author*:
Requires: *Language*: BASIC *Updated*:

Set of programs that create mailing and disk envelopes. disk labels, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = NORTH-99ER
Sectors total           = 360
Sectors used            = 200
Sectors available       = 158
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CS1*DOC      62 INT/VAR254 Y >022 061
002 >003 CS1*DOC1     43 PROGRAM    Y >05f 042
003 >004 CS1*DOC2     31 PROGRAM    Y >089 030
004 >005 CS1*FINDEX   17 PROGRAM    Y >0a7 016
005 >006 DSKJACKET+   16 PROGRAM    Y >0b7 015
006 >007 DSKLABEL+    14 PROGRAM    Y >0c6 013
007 >008 ENVELOPE+    13 PROGRAM    Y >0d3 012
008 >009 LOAD          4 PROGRAM      >0df 003
```

Disk 16: Neatlist

Version:
Author: Danny Michael
Requires: XB
Language: AL
Updated:

Creates listings of your XB programs with each statement on a separate line and variable cross reference table. Very fast. Includes source code.

```
dskdir. v2.0. 12-dec-96
Disk name               = NEATLIST
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 DEFAULT       9 DIS/FIX 80   >022 008
002 >003 DRIVE#        5 PROGRAM      >02a 004
003 >004 LDSRLNK       8 DIS/VAR 80   >02e 007
004 >005 LINIT        17 DIS/VAR 80   >035 016
005 >006 LIST/TXT      4 DIS/VAR 80   >045 003
006 >007 LISTOBJ      23 INT/FIX255   >048 022
007 >008 LKEY         13 DIS/VAR 80   >05e 012
008 >009 LMOVBUF      10 DIS/VAR 80   >06a 009
009 >00a LPARAM       35 DIS/VAR 80   >073 034
010 >00b LPRINT       25 DIS/VAR 80   >095 024
011 >00c LREF         55 DIS/VAR 80   >0ad 054
012 >00d LTABLES      25 DIS/VAR 80   >0e3 024
013 >00e MAINLIST     43 DIS/VAR 80   >0fb 042
014 >00f NEATDOC      72 DIS/VAR 80   >125 067 >010 004
015 >014 NEATDOCBAS    3 PROGRAM      >015 002
016 >017 NEATLIST     11 DIS/FIX 80   >018 010
```

Disk 16. Contents of File LDSRLNK

```
**********************************
* DSRLNK - PRINTS TO DEVICE IN PAB *
**********************************
DSRL1  MOV  *R14+,R5
       SZCB @MASK,R15
       MOV  @>8356,R0
       MOV  R0,R9
       AI   R9,>FFF8
       BLWP @VSBR
       MOVB R1,R3
       SRL  R3,8
       SETO R4
       LI   R2,DSRBUF
DSRL2  INC  R0
       INC  R4
       C    R4,R3
       JEQ  DSRL3
       BLWP @VSBR
       MOVB R1,*R2+
       CB   R1,@PERIOD
       JNE  DSRL2
DSRL3  MOV  R4,R4
       JEQ  DSRL10
       CI   R4,>0007
       JGT  DSRL10
       CLR  @>83D0
       MOV  R4,@>8354
       INC  R4
       A    R4,@>8356
       LWPI >83E0
       CLR  R1
       LI   R12,>0F00
DSRL4  MOV  R12,R12
       JEQ  DSRL04
       SBZ  >00
DSRL04 AI   R12,>0100
       CLR  @>83D0
       CI   R12,>2000
       JEQ  DSRL9
       MOV  R12,@>83D0
       SBO  >00
       LI   R2,>4000
       CB   *R2,@VALID
       JNE  DSRL4
       AI   R2,8
       JMP  DSRL6
DSRL5  MOV  @>83D2,R2
       SBO  >00
```

```
DSRL6  MOV  *R2,R2
       JEQ  DSRL4
       MOV  R2,@>83D2
       INCT R2
       MOV  *R2+,R9
       MOVB @>8355,R5
       JEQ  DSRL8
       CB   R5,*R2+
       JNE  DSRL5
       SRL  R5,8
       LI   R6,DSRBUF
DSRL7  CB   *R6+,*R2+
       JNE  DSRL5
       DEC  R5
       JNE  DSRL7
DSRL8  INC  R1
       BL   *R9
       JMP  DSRL5
       SBZ  >00
       LWPI REG3
       MOV  R9,R0
       BLWP @VSBR
       SRL  R1,13
       JNE  DSRL11
       RTWP
DSRL9  LWPI REG3
DSRL10 CLR  R1
DSRL11 SWPB R1
       MOVB R1,*R13
       SOCB @MASK,R15
       RTWP
```

Disk 16. Contents of File LINIT

```
*LINIT-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
       DEF  LIST,KEYMSK,ENDBUF
GPLWS  EQU  >83E0      GPL WORKSPACE
KSCAN  EQU  >201C      KEY SCAN UTILITY
KEYVAL EQU  >8375      KEY CODE FROM KSCAN
STATUS EQU  >837C      GPL STATUS BYTE
VSBR   EQU  >2028      VIDEO SINGLE BYTE READ
VSBW   EQU  >2020      VIDEO SINGLE BYTE WRITE
VMBR   EQU  >202C      VIDEO MULTI BYTE READ
VMBW   EQU  >2024      VIDEO MULTI BYTE WRITE
GRMRA  EQU  >9802      GROM READ ADDRESS
GRMWA  EQU  >9C02      GROM WRITE ADDRESS
PNTR   EQU  >8356      POINTER BYTE FOR DSRLNK
PABLOC EQU  >1000      LOCATION OF PAB
COUNT  EQU  PABLOC+5   BYTE COUNT FOR DSR ROM
BUFFER EQU  PABLOC+72  READ/WRITE BUFFER IN VDP
BEGTBL EQU  >8330      BEGINNING OF LINE # TABLE
ENDTBL EQU  >8332      END OF LINE # TABLE
FAC    EQU  >834A      FLOATING POINT ACC
************************************
*            ## WARNING ##          *
* THE NEXT SIX LABELS MUST REMAIN   *
* TOGETHER, AND IN THIS ORDER FOR   *
* THE PROGRAM TO FUNCTION CORRECTLY *
************************************
FSTLIN BSS  2    FIRST LINE TO BE PRINTED
LSTLIN BSS  2    LAST LINE TO BE PRINTED
LINLEN BSS  2    LINE LENGTH
LFTMGN BSS  2    LEFT MARGIN
PGLEN  BSS  2    PAGE LENGTH
TBMGN  BSS  2    TOP & BOTTOM MARGIN
PABFLG BSS  2    FLAG FOR OPEN FILE
TMPLEN BSS  2    LENGTH OF DATA IN TMPBUF
SPACES BSS  2    # OF SPACES TO INDENT
REG1   BSS  >20  SPACE FOR REGISTER SET 1
REG2   BSS  >20  SPACE FOR REGISTER SET 2
REG3   BSS  >20  SPACE FOR REGISTER SET 3
SUBFLG BSS  2    FLAG FOR SUBPROGRAMS
MSGPTR BSS  2    POINTS TO VARIABLE MESSAGE
GPLBUF BSS  >20  SPACE TO SAVE GPL REGS
REMFLG BSS  2    FLAG FOR REMARK LINE
SVPNTR BSS  2    BUFFER USED IN DSR CALLS
SAVE11 BSS  2    SAVING PLACE FOR R11
DSRBUF BSS  8    BUFFER FOR VARIOUS ROUTINES
VBUF   BSS  6    BUFFER FOR HEXASC SUB.
XBUF   BSS  16   BUFFER FOR VARIABLE NAMES DURING REF
GROMSV BSS  2    BUFFER FOR GROM ADDRESS
PABBUF BSS  212  BUFFER TO SAVE VDP RAM USED
```

```
TMPBUF BSS  150  TEMPORARY BUFFER FOR OUTPUT
CODEBF BSS  10   FOR SPECIAL CODES
INDENT BSS  2    REGULAR INDENT
LONGLN BSS  2    LONG LINE INDENT
SKIP   BSS  2    FLAG FOR FORMFEED IN REF
STRTLN BSS  2    SPACE FOR POINTER IN LINE# REF
KEYMSK DATA >200         MASK BIT FOR KSCAN
DSRLNK DATA REG3,DSRL1   DATA FOR BLWP @DSRLNK
MOVBUF DATA REG2,MOVBF   DATA FOR BLWP @MOVBUF
KEY    DATA REG2,KEY1    DATA FOR BLWP @KEY
PARAM  DATA REG2,CKPAR   DATA FOR BLWP @PARAM
HEXASC DATA REG2,HEX00   DATA FOR BLWP @HEXASC
PAB    DATA >0012,BUFFER,>FE00,0,0   DATA FOR PAB
WNCODE BYTE 0    SPECIAL CODE FLAG
HOWREF BYTE 0    HOW TO REFERENCE VAR.
DOWHAT BYTE 0    WHAT TO DO
N      BYTE >AE  OFFSET CHAR N
Y      BYTE >B9     "     "  Y
OFFSET BYTE >60  SCREEN OFFSET
OPEN   BYTE 0    OPEN OPCODE FOR DSR
CLOSE  BYTE 1    CLOSE OPCODE
WRITE  BYTE 3    WRITE OPCODE
MASK   BYTE >20  MASK TO SET/RESET EQUAL BIT
PERIOD BYTE >2E  ASCII FOR PERIOD
VALID  BYTE >AA  VALIDATION BYTE FOR DSR ROMS
************************************
*            ## WARNING ##          *
* THE NEXT FIVE LABELS MUST REMAIN  *
* TOGETHER, AND IN THIS ORDER FOR   *
* THE PROGRAM TO FUNCTION CORRECTLY *
************************************
ENTER  BYTE 13   CHAR CODE FOR ENTER
PRCD   BYTE 12     "    "   "  PROCEED
QUIT   BYTE 15     "    "   "  FCTN 9
UA     BYTE 11     "    "   "  UP ARROW
DA     BYTE 10     "    "   "  DOWN ARROW
LA     BYTE 8      "    "   "  LEFT ARROW
RA     BYTE 9      "    "   "  RIGHT ARROW
CURSOR BYTE >7E    "    "   "  CURSOR
SPACE  BYTE 32     "    "   "  SPACE
QUOTE  BYTE 34     "    "   "  QUOTE
ZERO   BYTE >30    "    "   "  ZERO
DEVICE BYTE 3    KEYBOARD SELECT
       EVEN
```

Disk 16. Contents of File LIST/TXT

```
       TITL 'NEATLIST V1.0'
*
* NEATLIST IS A PUBLIC DOMAIN PROGRAM
* WRITTEN BY DANNY MICHAEL
* RT 9, BOX 460, FLORENCE, AL. 35630
* (205)764-7881  CIS 75116,1225
* THIS PROGRAM NOT TO BE SOLD!
* ANYONE WHO WISHES MAY HAVE IT FREE.
* RELEASED TO PUBLIC DOMAIN, JAN. 1985
*
       COPY "DSK1.LINIT"
       COPY "DSK1.LTABLES"
       COPY "DSK1.LMOVBUF"
       COPY "DSK1.LDSRLNK"
       COPY "DSK1.LKEY"
       COPY "DSK1.LPARAM"
       COPY "DSK1.LPRINT"
       COPY "DSK1.LREF"
       COPY "DSK1.MAINLIST"
       END
```

Disk 16. Contents of File LKEY

```
*LKEY-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
KEY1   CLR  R5               SET R5 TO ZERO
       MOVB @DEVICE,@>8374   SELECT UCASE KBD
       LI   R6,>0100         TIMER VALUE
       MOV  @2(R13),R2       R2 POINTS TO POSITION TABLE
       MOV  *R2+,R0          R0=VDP RAM ADDRESS
       MOV  R0,R3            SAVE BEGINNING ADDRESS
       MOV  *R2,R4           CHAR COUNT TO R4
KEY6   BLWP @VSBR            GET CHAR INTO R1
KEY5   SWPB R1               PUT IT IN RIGHT BYTE
       MOVB @CURSOR,R1       CURSOR CHAR IN LEFT SIDE
       BLWP @VSBW            WRITE TO SCREEN
KEY2   DEC  R6               DECREMENT TIMER
       JNE  KEY3             NOT ZERO YET
       SWPB R1               SWAP CHAR & CURSOR
       BLWP @VSBW              ON SCREEN
       LI   R6,>0100         RESTORE TIMER
KEY3   BLWP @KSCAN           DO KBD SCAN
       MOVB @STATUS,R5       STATUS BYTE TO R5
       COC  @KEYMSK,R5       NEW KEY?
       JNE  KEY2             NO, TRY AGAIN
       CB   @KEYVAL,@SPACE   SPACE?
       JL   KEY4             GO ON LESS
       MOVB @KEYVAL,R1       NEW CHAR TO R1
       AB   @OFFSET,R1       ADD SCREEN OFFSET
       BLWP @VSBW            PUT IT ON SCREEN
       CI   R4,2             R4<2?
       JL   KEY5             LAST CHAR IN COUNT
       DEC  R4               CHAR COUNT -1
       INC  R0               NEXT SCREEN POSITION
       JMP  KEY6             DO IT AGAIN
KEY4   MOVB @KEYVAL,R7       KEY TO R7
       CB   R7,@LA           LEFT ARROW?
       JEQ  LARROW           YES
       CB   R7,@RA           RIGHT ARROW?
       JEQ  RARROW           YES
       LI   R8,ENTER         R8 POINTS TO FIRST BYTE TO CHECK
       LI   R9,5             # OF BYTES TO CHECK
KEY7   CB   R7,*R8+          COMPARE TO TERMINATOR LIST
       JEQ  KEYEND           JUMP IF VALID
       DEC  R9               COUNT -1
       JNE  KEY7             NO MATCH YET
       JMP  KEY2             NOT VALID TERMINATOR
KEYEND BL   @RSTCHR          RESTORE CHAR
       RTWP                  RETURN
LARROW BL   @RSTCHR          RESTORE SCREEN CHAR
       C    R3,R0
       JEQ  LAR1             CAN'T DO, ALREADY AT LEFTMOST POS.
```

```
       DEC  R0              PREVIOUS SCREEN POS.
       INC  R4              ADD ONE TO CHAR COUNT
LAR1   JMP  KEY6            GET ANOTHER KEY
RARROW BL   @RSTCHR         RESTORE SCREEN CHAR
       CI   R4,1            R4=1?
       JEQ  RAR1            YES, CAN'T GO RIGHT
       DEC  R4              BYTE COUNT -1
       INC  R0              NEXT SCREEN POSITION
RAR1   JMP  KEY6            GET ANOTHER KEY
RSTCHR CB   R1,@CURSOR      CURSOR ON SCREEN?
       JNE  RST1            NO
       SWPB R1              YES, GET CHAR
       BLWP @VSBW           PUT IT ON SCREEN
RST1   B    *R11            RETURN
```

Disk 16. Contents of File LMOVBUF

```
*LMOVBUF-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
*********************************
* BUFFER TO BE OUTPUT TO SCREEN *
* SCREEN CONTENTS SAVED HERE    *
*********************************
SCRBUF TEXT '  NEATLIST - by Danny Michael   '
       TEXT '                                '
       TEXT '  OUTPUT: PROGRAM ?      Y      '
       TEXT '          VARIABLES ?    Y      '
       TEXT '          LINE REF ?     N      '
       TEXT '                                '
       TEXT '  OUTPUT TO: PIO.CR             '
       TEXT '                                '
       TEXT '                                '
       TEXT '  BEGIN LISTING AT LINE  0      '
       TEXT '  END LISTING AT LINE    32767  '
       TEXT '                                '
       TEXT '  LINE LENGTH (MAX=132)  80     '
       TEXT '  LEFT MARGIN (MAX=20)   0      '
       TEXT '  PAGE LENGTH (MAX=255)  66     '
       TEXT '  TOP & BOTTOM MARGIN    3      '
       TEXT '                                '
       TEXT '  SPECIAL CODES:                '
       TEXT '                                '
       TEXT '  OUTPUT CODES EACH LINE? N     '
       TEXT '                                '
       TEXT '                                '
       TEXT '                                '
       TEXT '"PRCD" TO BEGIN, "BACK" TO ABORT'
       EVEN
****************************************
* SUBROUTINE TO SWAP SCREEN WITH SCRNBUF *
****************************************
MOVBF  CLR  R0           VDP RAM POINTER
       LI   R2,768       BYTE COUNT
       LI   R3,SCRBUF    POINT TO BUFFER
MOVBF1 BLWP @VSBR        GET BYTE FROM VDP
       SB   @OFFSET,R1   STRIP OFFSET
       SWPB R1           MOVE TO RIGHT BYTE
       MOVB *R3,R1       BUFFER BYTE TO LEFT BYTE, R1
       AB   @OFFSET,R1   ADD OFFSET
       BLWP @VSBW        WRITE TO SCREEN
       SWPB R1           SCREEN BYTE BACK TO LEFT
       MOVB R1,*R3+      MOVE TO BUFFER & INC POINTER
       INC  R0           POINT TO NEXT SCRN LOCATION
       DEC  R2           BYTE COUNT -1
       JNE  MOVBF1       DO AGAIN
       RTWP              RETURN
```

Disk 16. Contents of File LPARAM

```
*LPARAM-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
**********************************
* CHECK PARAMETERS                *
* ENTER WITH BLWP @PARAM          *
* EXITS WITH EQU. BIT SET IF ERROR *
* R3 POINTS TO ERROR MESSAGE      *
* R4 POINTS TO PTABLE ENTRY       *
**********************************
ERMSG1 BYTE 26
       TEXT '       ** BAD PARAMETER **'
ERMSG2 BYTE 24
       TEXT '         ** I/O ERROR **'
ERMSG3 BYTE 27
       TEXT '      ** BAD DEVICE NAME **'
ERMSG4 BYTE 29
       TEXT '  ** LINE LENGTH TOO SHORT **'
ERMSG5 BYTE 29
       TEXT '  ** PAGE LENGTH TOO SHORT **'
       EVEN
CKPAR  SZCB @MASK,R15    RESET EQUAL BIT IN CALLING WS
       LI   R0,704-32    POINT TO ERROR MSG LINE
       LI   R1,>8000     SPACE CHAR TO R1
       LI   R2,32        BYTE COUNT
CKPAR1 BLWP @VSBW        SEND SPACE
       INC  R0           NEXT SCREEN POS.
       DEC  R2           COUNT -1
       JNE  CKPAR1       LOOP TILL DONE
       CLR  R5           R5=0
       MOVB R5,@DOWHAT   CLEAR DOWHAT
       LI   R3,1         R3=1
       LI   R4,PTABLE    R4 POINTS TO POSITION TABLE
CK02   MOV  *R4,R0       R0=VDP ADR.
       BL   @CKYN        CHECK FOR Y OR N
       JEQ  RTER1        ERROR
       JLT  CK01         N
       SWPB R3           COUNT TO LEFT BYTE
       AB   R3,@DOWHAT   ADD R3
       SWPB R3           COUNT BACK TO RIGHT
CK01   INC  R3           R3=R3+1
       AI   R4,4         NEXT TABLE ENTRY
       CI   R3,3         R3=3?
       JNE  CK02         NO, JUMP
       MOV  *R4,R0       SCREEN ADR.
       BL   @CKYN        CHECK FOR Y OR N
       JEQ  RTER1        ERROR
       JLT  CK07         N
       SWPB R1           TO LEFT BYTE
       MOVB R1,@HOWREF   SET FLAG
```

```
CK07    LI   R5,6          LOOP COUNTER
        LI   R7,FSTLIN     FIRST DATA AREA
        LI   R8,MAXTBL     TABLE OF MAXIMUM VALUES
        AI   R4,4          BUMP PAST OUTPUT DEVICE
CK08    AI   R4,4          NEXT TABLE ENTRY
        LI   R9,BUFFER     POINT TO VDP BUFFER
        MOV  *R4,R0        SCREEN POS. TO R0
        BL   @CK09         MOVE CHARS TO BUFFER
        BL   @CK11         CONVERT TO INTEGER
        JEQ  RTER1         ERROR
        C    @FAC,*R8+     COMPARE TO MAX VALUE
        JH   RTER1         ERROR
        MOV  @FAC,*R7+     STORE
        DEC  R5            LOOP COUNT -1
        JNE  CK08          LOOP TILL DONE
        MOV  @LINLEN,R0    LINE LENGTH TO R0
        S    @LFTMGN,R0    TAKE AWAY MARGIN
        CI   R0,29         30 SPACES LEFT?
        JGT  CK16A         YES
        LI   R3,ERMSG4     POINT TO ERROR MESG
        LI   R4,PTABLE+24  POINT TO INPUT LINE
        JMP  RTER0         RETURN W/ERROR
CK16A   MOV  @TBMGN,R0     MARGINS TO R0
        SLA  R0,1          R0*2
        INC  R0            R0+1
        C    R0,@PGLEN     COMPARE WITH PAGE LENGTH
        JLT  CK16          GO ON IF LESS
        LI   R3,ERMSG5     POINT TO ERROR MSG
        LI   R4,PTABLE+32  POINT TO INPUT FIELD
        JMP  RTER0         RETURN W/ERROR
RTER3   LI   R3,ERMSG3
        JMP  RTER0
RTER1   LI   R3,ERMSG1
RTER0   MOV  R3,@6(R13)    PUT MSG POINTER IN OLD R3
        MOV  R4,@8(R13)    PUT PTABLE IN OLD R4
        SOCB @MASK,R15     SET EQUAL BIT
        RTWP               RETURN
CK16    AI   R4,4          NEXT SCREEN POS.
        MOV  *R4,R0          TO R0
        CLR  R5            R5=0
        LI   R7,CODEBF+1   POINT TO CODE BUFFER
CK12    LI   R9,BUFFER     POINT TO BUFFER
        BL   @CK09         MOVE TO BUFFER
        BL   @CK11         CONVERT
        JEQ  RTER1         ERROR
        DEC  R0            CHECK FIRST
        C    R0,*R4          FOR NON NUMERIC
        JEQ  CK13A           CHAR
        INC  R0            RESTORE POINTER
        C    @FAC,*R8      COMPARE TO MAX
```

```
        JH   RTER1         ERROR
        MOV  @FAC,R3       INT. TO R3
        SWPB R3              LEFT BYTE
        MOVB R3,*R7+       AND ON TO BUFFER
        CI   R1,32         WAS LAST CHAR A SPACE?
        JEQ  CK13          YES
        INC  R5            COUNT +1
        CI   R5,9          IF LESS THAN 8
        JLT  CK12            THEN LOOP
        JMP  CK13A
CK13    INC  R5            ADJUST COUNT
CK13A   SWPB R5            TO LEFT BYTE
        MOVB R5,@CODEBF    MOVE TO BUFFER
        AI   R4,4          NEXT SCREEN POS.
        MOV  *R4,R0         TO R0
        BL   @CKYN         CHECK FOR Y OR N
        JEQ  RTER1         ERROR
        SETO R5
        JGT  CK14          Y
        CLR  R5
CK14    MOVB R5,@WNCODE    SET WNCODE
CK03    LI   R0,PABLOC     POINT TO PAB LOCATION
        LI   R1,PAB        R1 POINTS TO INFO
        LI   R2,10         BYTE COUNT
        BLWP @VMBW         WRITE IT
        CLR  R1
        CLR  R2
        LI   R3,PABLOC+10 R3 POINTS TO DEV. DESCRIPT LOC
        LI   R4,PTABLE+12 POINT TO NEXT TABLE ENTRY
        MOV  *R4,R0        R0 POINTS TO DEV NAME ON SCRN
CK04    BLWP @VSBR         READ BYTE
        SWPB R1            MOVE RIGHT
        CI   R1,>80        SPACE?
        JEQ  CK05          YES
        INC  R2            NO, COUNT +1
        SWPB R1            BACK LEFT
        SB   @OFFSET,R1    SUBTRACT OFFSET
        INC  R0            READ POINTER +1
        MOV  R0,R5         SAVE
        MOV  R3,R0         WRITE POINTER TO R0
        BLWP @VSBW         WRITE IT
        INC  R3            WRITE POINTER +1
        MOV  R5,R0         RESTORE POINTER
        JMP  CK04          LOOP
CK05    MOV  R2,R2         ERROR IF
        JEQ  RTER1            R2=0
        MOV  R2,R1         COUNT TO R1
        SWPB R1              LEFT BYTE
        LI   R0,PABLOC+9   COUNT BYTE
        MOV  R0,@PNTR      POINTER FOR DSRLNK
        MOV  R0,@SVPNTR    SAVE IT
```

```
       BLWP @VSBW        WRITE COUNT BYTE
       SETO @PABFLG      SET FLAG FOR OPEN FILE
       BLWP @DSRLNK      OPEN FILE
       DATA 8
       JNE  CK15         GOOD DEVICE
       SRL  R0,8         ERROR CODE TO RIGHT
       JEQ  RTER3        BAD DEVICE
CK17   LI   R0,PABLOC    PREPARE TO CLOSE FILE
       MOVB @CLOSE,R1    CLOSE OPCODE
       BLWP @VSBW          TO PAB
       MOV  @SVPNTR,@PNTR
       BLWP @DSRLNK      CLOSE FILE
       DATA 8
       CLR  @PABFLG
       LI   R3,ERMSG2    POINT TO DEVICE ERROR
       JMP  RTER0
CK15   RTWP              RETURN
****************************************
* THIS ROUTINE CONVERTS A STRING LOCATED *
* AT BUFFER TO A ONE WORD INTEGER LEFT   *
* IN FAC. EQUAL BIT SET ON ERROR.        *
****************************************
CK11   LI   R9,BUFFER    RESTORE
       CLR  @FAC+10
       MOV  R9,@FAC+12   MOVE POINTER FOR ROM
       LWPI GPLWS        CONVERT STRING TO NUMBER
       BL   @>11AE
       BL   @>12B8       CONVERT NUMBER TO INTEGER
       LWPI REG2
       MOVB @FAC+10,R3   ERROR CODE TO R3
       CB   @DEVICE,R3   IS IT A 3?
       B    *R11         RETURN
****************************************
* THIS ROUTINE MOVES CHARS FROM SCREEN *
* (STRIPS OFFSET) TO BUFFER IN VDP RAM *
* UNTIL A NON NUMERIC CHAR IS MOVED    *
* ENTER WITH R0=SCREEN, R9=BUFFER      *
****************************************
CK09   BLWP @VSBR        READ FIRST BYTE
       SB   @OFFSET,R1   STRIP OFFSET
       MOV  R0,R10       SAVE R0
       MOV  R9,R0        GET BUFFER ADDR.
       BLWP @VSBW        WRITE TO BUFFER
       INC  R0           BUMP BUFFER POINTER
       MOV  R0,R9          AND STORE
       MOV  R10,R0       GET SCREEN POINTER
       INC  R0             AND BUMP
       SRL  R1,8         SHIFT TO RIGHT BYTE
       CI   R1,>30       ZERO
       JLT  CK10         TOO LOW
```

```
       CI   R1,>39       NINE
       JLE  CK09
CK10   B    *R11
************************************
* THIS ROUTINE CHECKS VDP FOR Y OR N *
* ENTER R0=VDP LOCATION. EXIT EQUAL  *
* BIT SET IF ERROR, AGT BIT SET IF Y *
************************************
CKYN   BLWP @VSBR        GET BYTE IN R1
       CB   @N,R1        IS IT N?
       JNE  CKY          NO
       LI   R1,-1        YES, SET R1= -1
       JMP  CKYN1        GO BACK
CKY    CB   @Y,R1        IS IT Y?
       JNE  CKYN2        NO, ERROR
       LI   R1,1         R1=1
       JMP  CKYN1
CKYN2  CLR  R1
CKYN1  MOV  R1,R1        SET STATUS BITS
       B    *R11         RETURN
```

Disk 16. Contents of File LPRINT

```
*LPRINT-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
************************************
* PRINT - ENTER WITH R0=ADR OF       *
* STRING TO PRINT. R1=LEN OF STRING. *                *
************************************
PRINT  MOV  R11,R15      SAVE RETURN
       MOVB @DEVICE,@>8374  KBD SELECT
       BLWP @KSCAN       CHECK FOR QUIT
       CB   @KEYVAL,@QUIT
       JNE  PRNT02       NO QUIT
       B    @EXIT        YES, QUIT
PRNT02 MOV  @TMPLEN,R2   BUFFER LEN TO R2
       MOV  R2,R10       SAVE A COPY
       A    R1,R2        ADD STRING LEN
       A    @LFTMGN,R2   ADD IN MARGIN
       A    @SPACES,R2   ADD INDENT
       C    R2,@LINLEN   LINE TOO LONG?
       JLE  PRNT01       NO
       MOV  R0,R8        SAVE R0
       MOV  R1,R9        SAVE R1
       BL   @SENDIT      PRINT THE BUFFER
       CLR  R10          BUFFER EMPTY NOW
       MOV  @LONGLN,@SPACES   SET BIG INDENT
       MOV  R8,R0        RESTORE R0
       MOV  R9,R1        RESTORE R1
PRNT01 LI   R3,TMPBUF    POINT TO BUFFER
       A    R10,R3       BUMP TO END OF BUFFER
       A    R1,R10       ADD CHAR COUNT
       MOV  R10,@TMPLEN    AND STORE IT
PRNT03 MOVB *R0+,*R3+    MOVE BYTE TO BUFFER
       DEC  R1           COUNT -1
       JNE  PRNT03       LOOP TILL DONE
       CI   R0,TABLE+5   CHECK FOR ::
       JNE  PRNT04       NO
       BL   @SENDIT      PRINT LINE
       CLR  @TMPLEN      BUFFER EMPTY
       MOV  @INDENT,@SPACES   SET REGULAR INDENT
PRNT04 B    *R15         RETURN
****************************************
* SENDIT - PRINTS TO OUTPUT DEVICE, TAKES *
* CARE OF SPECIAL CODES, MARGINS & FORM   *
* FEEDS. R14 KEEPS TRACK OF PRINTED LINES *
****************************************
SENDIT LI   R0,BUFFER    POINT TO PAB BUFFER
       MOVB @CODEBF,R2   SPEC CODE COUNT TO R2
       SRL  R2,8         ADJUST
       MOV  R2,R12       CHECK FOR ZERO AND STORE
       JEQ  S01          GO IF NO CODE
```

```
        LI   R1,CODEBF+1  POINT TO CODES
        BLWP @VMBW        WRITE TO BUFFER
        A    R2,R0        ADJUST BUFFER POINTER
S01     MOV  @LFTMGN,R3   MARGIN COUNT
        A    @SPACES,R3   ADD INDENT
        JEQ  S04          JUMP IF NONE
        A    R3,R12       INC COUNT
        MOVB @SPACE,R1    SPACE CHAR TO R1
S03     BLWP @VSBW        WRITE TO BUFFER
        INC  R0           NEXT POSITION
        DEC  R3           COUNT -1
        JNE  S03          LOOP TILL DONE
S04     MOV  @TMPLEN,R2   LENGTH OF CPU RAM BUFFER
        JEQ  S05          NOTHING TO PRINT
        A    R2,R12       ADD COUNT
        LI   R1,TMPBUF    POINT TO BUFFER
        BLWP @VMBW        WRITE TO BUFFER
        A    R2,R0        BUMP TO END OF BUFFER
S05     LI   R1,>0D0A     CRLF
        BLWP @VSBW        SEND CR
        SWPB R1           SWAP BYTES
        INC  R0           NEXT POS
        BLWP @VSBW        SEND LF
        INCT R12          BUMP COUNT
        SWPB R12          COUNT TO LEFT
        MOV  R12,R1       MOVE TO R1
        LI   R0,COUNT     POINT TO COUNT BYTE
        BLWP @VSBW        WRITE IT
        LI   R0,PABLOC    POINT TO PAB
        MOVB @WRITE,R1    WRITE OPCODE
        BLWP @VSBW          TO PAB
        MOV  @SVPNTR,@PNTR FOR DSR
        BLWP @DSRLNK      PRINT IT
        DATA 8
        JNE  S06          NO ERRORS
        MOV  @GPLWS+24,R4 CRU CODE TO R4
        CI   R4,>1300     RS232 CARD?
        JNE  S11          NO
        LI   R11,EXIT     WHERE TO GO
        B    *R11         GO THERE
S11     LI   R13,REG1     WORKSPACE PNTR TO 13
        LI   R14,LIS02    TO PRINT ERROR MSG
        LI   R4,PTABLE+12 POINT TO OUTPUT DEVICE
        B    @CK17        CLOSE FILE ON ERROR
S06     MOV  R14,R14      SEE IF DOING FORMFEED
        JEQ  S08          YES, CONTINUE
        MOVB @WNCODE,R0   WHEN TO SEND SPEC CODE
        SRL  R0,8         ALIGN
        JNE  S07          MUST DO EACH TIME
        CLR  @CODEBF      ZERO COUNT FOR CODES
S07     DEC  R14          COUNT -1
```

```
       JNE  S09          RETURN
       MOV  @TBMGN,R3    LINE COUNT TO R3
       SLA  R3,1         R3=R3*2
       INC  R3           ADJUST
S08    DEC  R3           COUNT -1
       JEQ  S10          DONE
       LI   R0,BUFFER    POINT TO PAB BUFFER
       CLR  R12          SET TO ZERO
       JMP  S05          LINE FEED
S10    MOV  @PGLEN,R14   RESTORE COUNT
S09    B    *R11         RETURN
***********************************************
* HEXASC - CONVERTS 2 BYTE NUMBER IN VBUF TO    *
* 6 BYTE ASCII STRING. STRING IS LEFT IN VBUF   *
***********************************************
HEX00  LI   R4,10000     DIVISOR
       LI   R5,10        CONSTANT DIVISOR
       LI   R7,VBUF      POINT TO BUFFER
       MOV  *R7,R2       HEX TO R2
HEX01  CLR  R1           CLEAR MSW
       DIV  R4,R1        DIVIDE HEX
       AI   R1,>30       MAKE IT ASCII
       SWPB R1           PUT IN LEFT BYTE
       MOVB R1,*R7+      PUT IT IN BUFFER
       CI   R4,1         DIVISOR 0?
       JNE  HEX02        NO, CONTINUE
       MOVB @SPACE,*R7   MAKE LAST CHAR A SPACE
       RTWP              RETURN
HEX02  CLR  R3           CLEAR MSW
       DIV  R5,R3        DIVIDE DIVISOR BY 10
       MOV  R3,R4        PUT IT BACK
       JMP  HEX01        LOOP
****************************************
* STRING - PRINTS A STRING. STRING LENGTH *
* IS POINTED TO BY R5, THE STRING FOLLOWS *
****************************************
STRING MOV  R11,R7       SAVE RETURN ADR
       MOVB *R5+,R6      COUNT TO R6
       JEQ  STRIN2       RETURN IF NULL STRING
       SRL  R6,8         TO RIGHT BYTE
STRIN1 MOV  R5,R0        POINTER TO R0
       INC  R5           BUMP
       LI   R1,1         COUNT
       BL   @PRINT       PRINT IT
       DEC  R6           COUNT -1
       JNE  STRIN1       LOOP TILL DONE
STRIN2 B    *R7          RETURN
```

Disk 16. Contents of File LREF

```
*LREF-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
VRMSG1 BYTE 30
       TEXT 'VARIABLES USED IN MAIN PROGRAM'
VRMSG2 BYTE 29
       TEXT 'VARIABLES USED IN SUBPROGRAM '
VRMSG3 BYTE 28
       TEXT '    ** TOO MANY VARIABLES **'
       EVEN
REFR   MOV  R11,@SAVE11  SAVE RETURN ADR
       CLR  @SUBFLG      CLEAR SUBPROGRAM FLAG
       CLR  @SKIP        CLEAR FF FLAG
       LI   R1,VRMSG1    POINT TO MAIN TITLE
       MOV  R1,@MSGPTR     "    "   "    "
       LI   R6,VARBUF    POINT TO BIG BUFFER
       MOV  @ENDTBL,R4   FIRST ENTRY
       AI   R4,-3        ADJUST
REF01A MOVB @1(R4),R5      LINE
       SWPB R5             # TO
       MOVB *R4,R5         R5
       C    R5,@FSTLIN   CHECK FOR FIRST LINE
       JHE  REF01B       GOOD LINE
       AI   R4,-4        NEXT ENTRY
       JMP  REF01A       LOOP
REF01B MOV  R4,@STRTLN   FOR LINE# REF
       INCT @STRTLN      ADJUST
REF01  C    R5,@LSTLIN   PAST LAST LINE?
       JH   REF16        YES, SORT
       INCT R4           POINT TO LINE POINTER
       MOVB @1(R4),R5    MAKE R5
       SWPB R5            POINT TO
       MOVB *R4,R5        LINE
REF02  BL   @FNDVAR      CHECK LINE
       JEQ  REF11        END OF LINE, GET NEXT ONE
       JGT  REF02A       GO IF VAR NAME
       SETO @SUBFLG      SET FLAG
       MOV  R5,@REG2+16  SAVE R5
       MOV  R4,@REG2+18  SAVE R4
       B    @REF16       GO PRINT
REF02A MOV  R5,R0        POINTER TO R0
       CLR  R1           ZERO COUNTER
REF03  CB   *R5,R2       NON ALPHA?
       JHE  REF04        YES
       MOVB *R5,*R5      CHECK FOR END OF LINE
       JEQ  REF04        YES
       INC  R5           NEXT CHAR
       INC  R1           INC COUNT
       JMP  REF03        LOOP TILL END OF NAME
REF04  LI   R2,VARBUF    BUFFER BEGINNING
```

```
       MOV  R1,R3        COUNT COPY TO R3
       SWPB R3           COUNT TO LEFT
REF05  C    R2,R6        END ?
       JNE  REF07        NO
****************************************
* POINTER TO VARIABLE NAME IS PLACED    *
* IN BUFFER HERE. THE POINTER IS 3      *
* BYTES. THE FIRST IS A CHARACTER COUNT *
* FOR THE NAME, THE 2nd AND 3rd ARE A   *
* POINTER TO THE NAME IN CPU RAM.       *
****************************************
       MOVB R3,*R6+      COUNT IN BUFFER
       MOVB R0,*R6+       POINTER
       SWPB R0            TO
       MOVB R0,*R6+       BUFFER
       CI   R6,ENDBUF-3  BUFFER FULL?
       JLE  REF02        NO
       LI   R3,VRMSG3    YES, GOTO
       BL   @ERROR1      ERROR ROUTINE
       B    @LIST01      START OVER
REF07  CB   *R2+,R3      COMPARE BYTE COUNTS
       JNE  REF09        UNEQUAL, NO MATCH
       MOV  R0,R7        FIRST POINTER TO R7
       MOVB @1(R2),R8     SECOND
       SWPB R8            POINTER
       MOVB *R2,R8        TO R8
REF08  CB   *R7+,*R8+    COMPARE
       JNE  REF09        NO MATCH
       DEC  R1           COUNT -1
       JNE  REF08        LOOP
       JMP  REF02        MATCH MADE, CONTINUE
REF09  CLR  R1           CLEAR R1
       MOVB R3,R1         RESTORE
       SWPB R1            COUNT
       INCT R2           BUMP BUFFER POINTER
       JMP  REF05        TRY NEXT ONE
REF11  AI   R4,-6        NEXT ENTRY
       C    R4,@BEGTBL   END OF TABLE?
       JL   REF16        FINISHED
       MOVB @1(R4),R5      LINE
       SWPB R5             # TO
       MOVB *R4,R5         R5
       JMP  REF01        LOOP
****************************************
* AT THIS POINT, ALL VARIABLES IN MAIN  *
* PROGRAM OR SUBPROGRAM HAVE BEEN FOUND.*
****************************************
REF16  CI   R6,VARBUF    ANY VARIABLES?
       JNE  REF16A       YES, CONTINUE
       MOV  @SUBFLG,R0   CHECK FOR SUBPROGRAM
```

```
       JEQ  REF16B
       B    @REF98C
REF16B B    @REF99       NO, END
REF16A MOVB @HEXFF,*R6   MARK BUFFER END
***************************
* SORT ROUTINE BEGINS HERE *
* IT IS A BUBBLE SORT.     *
* SORTA CRUDE, HUH?        *
***************************
REF17  LI   R1,VARBUF    POINT TO BUFFER
       CLR  @REMFLG      CLEAR SWAP FLAG
REF21  MOV  R1,R2        SAVE IT
       MOVB *R1+,R3      FIRST COUNT TO R3
       MOVB @1(R1),R4     FIRST
       SWPB R4            POINTER
       MOVB *R1,R4        TO R4
       SRL  R3,8         ALIGN
       INCT R1           BUMP BUFFER POINTER
       CB   *R1,@HEXFF   END OF BUFFER?
       JEQ  REF30        YES, THRU THIS PASS
       MOVB *R1+,R5      SECOND COUNT TO R5
       MOVB @1(R1),R6     SECOND
       SWPB R6            POINTER
       MOVB *R1,R6        TO R6
       SRL  R5,8         COUNTS
       C    R5,R3        COMPARE LENGTHS
       JH   REF18        LAST ONE EQUAL OR LONGER
       SETO @VBUF        SET FLAG
       MOV  R5,R7        COUNT TO R7
       JMP  REF19
REF18  CLR  @VBUF        CLEAR OTHER SWAP FLAG
       MOV  R3,R7        COUNT TO R7
REF19  CB   *R4+,*R6+    COMPARE BYTES
       JH   REF20        MUST SWAP
       JNE  REF22        DON'T SWAP
       DEC  R7           COUNT -1
       JNE  REF19        LOOP
       SETO R8           R8=>FFFF
       C    R8,@VBUF     OTHER FLAG SET?
       JEQ  REF20        MUST SWAP
REF22  DEC  R1           ALIGN BUFFER POINTER
       JMP  REF21        LOOP
REF20  SETO @REMFLG      SET SWAP FLAG
       MOV  R2,R3        BUFFER POINTER TO R3
       MOVB *R3+,R4      COUNT TO R4
       MOVB *R3+,R5      1st BYTE OF POINTER TO R5
       MOVB *R3+,R6      2nd   "   "    "     " R6
       MOVB *R3+,*R2+     MOVE 2nd
       MOVB *R3+,*R2+     ENTRY
       MOVB *R3+,*R2+     BACK
       MOVB R4,*R2+        MOVE 1st
```

```
       MOVB R5,*R2+        ENTRY
       MOVB R6,*R2+        FORWARD
       DEC  R1            ADJUST BUFFER POINTER
       JMP  REF21         GO ON
REF30  SETO R8            R8=>FFFF
       C    R8,@REMFLG    SWAP BEEN MADE?
       JEQ  REF17         YES, DO AGAIN
** BEGIN PRINT ROUTINE
       CLR  @SPACES       SPACES =0
       CLR  @LONGLN       LONGLN=0
       MOVB @HOWREF,R1    CHECK FOR LINE REF
       JEQ  REF30D        NO LINE# REF
       LI   R1,16         INDENT
       MOV  R1,@LONGLN    INTO POSITION
REF30D CLR  @INDENT       INDENT=0
       CLR  @TMPLEN       TMPLEN=0
       MOV  @SKIP,R1      CHECK TO SEE IF FF DONE
       JNE  REF30A        YES, SKIP
       INC  @SKIP         DO IT THIS TIME ONLY
       MOV  @TBMGN,R3     MARGIN TO R3
       JEQ  REF30A        GO IF NO T&B MGN
       CLR  R14           FOR FF ROUTINE
       BL   @S08          DO TOP MARGIN
REF30A MOV  @MSGPTR,R5    POINT TO HEADER
       BL   @STRING       PRINT IT
       MOV  @MSGPTR,R5    MESSAGE POINTER TO R5
       CI   R5,VRMSG1     MAIN PROGRAM?
       JEQ  REF30B        YES
       MOV  @REG2+20,R5   POINT TO NAME
       LI   R1,>C800      CHECKER
       CB   *R5,R1        IS IT >C8?
       JEQ  REF30C        YES
       JMP  REF30B        NO, SKIP IT
REF30C INC  R5            POINT TO LENGTH
       BL   @STRING       PRINT NAME
REF30B BL   @SENDIT       FINISH UP
       CLR  @TMPLEN       BUFFER EMPTY NOW
       BL   @SENDIT       BLANK LINE
       LI   R2,VARBUF     BUFFER POINTER
REF31  MOVB *R2+,R1       LENGTH TO R1
       CB   @HEXFF,R1     END?
       JEQ  REF98         YES
       SRL  R1,8          ALIGN
       MOVB @1(R2),R4      MOVE
       SWPB R4             POINTER
       MOVB *R2,R4         TO R4
       LI   R0,XBUF       POINT TO LITTLE BUFFER
       LI   R5,16         COUNT
REF32  MOVB @SPACE,*R0+   PUT A SPACE THERE
       DEC  R5            COUNT -1
```

```
       JNE  REF32        LOOP
       LI   R0,XBUF      RESTORE
REF33  MOVB *R4+,*R0+    CHAR TO BUFFER
       DEC  R1           COUNT -1
       JNE  REF33        LOOP
       LI   R0,XBUF      POINT TO STRING
       LI   R1,16        SET COUNT
       MOV  R2,@REMFLG   SAVE BUFFER POINTER
       BL   @PRINT       PRINT IT!
       MOVB @HOWREF,R1   CHECK FOR LINE# REF
       JEQ  REFX10       NO, SKIP IT
       MOV  @STRTLN,R4   GET STARTING LINE
REFX08 MOVB @1(R4),R5     POINTER
       SWPB R5            TO
       MOVB *R4,R5        R5
REFX07 BL   @FNDVAR      SEARCH FOR VARIABLE
       JEQ  REFX01       GET NEXT LINE
       JLT  REFX09       SUB TOKEN, CHECK FOR END
       LI   R2,XBUF      POINT TO VAR NAME
       LI   R1,>8000     ALPHA CHECK BYTE
REFX05 CB   *R5+,*R2+    COMPARE BYTE
       JNE  REFX03       NO MATCH
       CB   *R2,@SPACE   END OF NAME?
       JEQ  REFX04       YES, SEE IF THE OTHER ONE ENDED
       CB   *R5,R1       END OF THIS ONE?
       JL   REFX11       MAYBE NOT, BETTER CHECK LINE END
       JMP  REFX03       NO MATCH
REFX04 MOVB *R5,*R5      CHECK FOR END OF LINE
       JEQ  REFX06       MATCH
       CB   *R5,R1       DID THIS ONE END TOO?
       JHE  REFX06       YEP, THEY MATCHED!
REFX03 MOVB *R5,*R5      CHECK FOR LINE END
       JEQ  REFX01       YES, GET NEXT LINE
       CB   *R5,R1        BUMP
       JHE  REFX07        PAST
       INC  R5            BAD
       JMP  REFX03        NAME
REFX11 MOVB *R5,*R5      CHECK FOR LINE END
       JNE  REFX05       CHECK NEXT CHAR
REFX01 AI   R4,-6        POINT TO NEXT LINE ENTRY
       C    R4,@BEGTBL   END OF TABLE?
       JL   REFX02       YES, FINISHED
       MOVB @1(R4),R5     LINE#
       SWPB R5            TO
       MOVB *R4,R5        R5
       C    R5,@LSTLIN   PAST LAST LINE?
       JH   REFX02       YES, FINISHED
       INCT R4           POINT TO POINTER
       JMP  REFX08       LOOP
REFX06 LI   R0,VBUF      POINT TO BUFFER
       MOVB @-2(R4),*R0      MOVE INTEGER
```

```
       MOVB @-1(R4),@VBUF+1  TO BUFFER
       BLWP @HEXASC      CONVERT TO ASCII
       LI   R1,6         BYTE COUNT
       BL   @PRINT       PRINT LINE#
       JMP  REFX01       CHECK NEXT LINE
REFX09 C    R4,@STRTLN   FIRST LINE?
       JNE  REFX02       NO, BETTER QUIT
       INC  R5            YES, SUB TOKEN
       JMP  REFX07        OK THIS TIME
REFX02 BL   @SENDIT      FINISH UP
       CLR  @TMPLEN      CLEAR COUNT
       CLR  @SPACES      CLEAR INDENT
REFX10 MOV  @REMFLG,R2   RESTORE POINTER
       INCT R2           ADJUST
       B    @REF31       LOOP
REF98  BL   @SENDIT      FINISH LAST LINE
       MOV  @SUBFLG,R0   CHECK FOR SUBPROGRAM
       JEQ  REF98B       NO SUBPROGRAM
       CLR  @TMPLEN      CLEAR BUFFER COUNT
       BL   @SENDIT      TWO BLANK
       BL   @SENDIT      LINES
REF98C MOV  @REG2+16,R5  RESTORE R5
       MOV  @REG2+18,R4     "    R4
       INC  R5           BUMP POINTER
       MOV  R5,@REG2+20  SAVE FOR PRINT ROUTINE
       MOV  R4,@STRTLN   FOR LINE# REF
       CLR  @SUBFLG      CLEAR FLAG
       LI   R0,VRMSG2    POINT TO SUB MESSAGE
       MOV  R0,@MSGPTR   MOVE TO POINTER
       LI   R6,VARBUF    POINT TO BIG BUFFER
       B    @REF02       CONTINUE
REF98B MOV  R14,R3       LINES REMAINING
       MOV  @TBMGN,R1    MARGINS
       JEQ  REF98A       GO IF ZERO
       INC  R1           COUNT +1
REF98A A    R1,R3        ADD IN MARGINS
       INC  R3
       JEQ  REF99
       CLR  R14          FOR FF ROUTINE
       BL   @S08         DO FORM FEED
REF99  MOV  @SAVE11,R11  RESTORE RETURN ADR
       B    *R11
************************************************
* SUBROUTINE FNDVAR                            *
* SEARCHES A LINE FOR A VARIABLE NAME.         *
* ENTER WITH R5=BEGINNING POSITION IN LINE.    *
* EXITS WITH EQUAL BIT SET IF END OF LINE      *
* ARITHMETIC GREATER THAN BIT RESET IF SUB TOKEN *
*     "         "      "   "  SET IF VARIABLE   *
* R5 WILL POINT TO BEGINNING OF VARIABLE NAME.  *
```

```
************************************************
FNDVAR CLR  R1           R1=0
FNDV00 CB   *R5,@SUB     SUBPROGRAM?
       JNE  FNDV01       NO
       DEC  R1           R1=-1
       JMP  XIT          RETURN
FNDV01 LI   R2,>0080     CHECKING WORD
       CB   *R5,R2       CHECK FOR LINE END
       JNE  FNDV02       NOT END
       JMP  XIT          RETURN
FNDV02 SWPB R2           REVERSE CHECKER
       CB   *R5,R2       CHECK FOR ALPHA
       JHE  FNDV03       NON ALPHA
       INC  R1           R1=1, MUST BE VAR NAME
       JMP  XIT          RETURN
FNDV03 CB   *R5,@REM     REM STATEMENT?
       JEQ  XIT          YES, MAKE LIKE END OF LINE
       CB   *R5,@REM1    ! ?
       JEQ  XIT          YEP,   "    "   "   "   "
       LI   R2,>C7C8     LOAD CHECKER
       CB   *R5,R2       QUOTED STRING?
       JNE  FNDV05       NO, TRY NEXT ONE
FNDV04 INC  R5           GET COUNT
       MOVB *R5+,R2       IN R2
       SRL  R2,8         ADJUST TO RIGHT
       A    R2,R5        ADD COUNT
       JMP  FNDV00       LOOP TO BEGINNING
FNDV05 SWPB R2           REVERSE CHECKER
       CB   *R5,R2       UNQUOTED STRING?
       JEQ  FNDV04       YES, BUMP PAST AND LOOP
       LI   R2,>C900     ANOTHER CHECKER
       CB   *R5,R2       INTEGER?
       JNE  FNDV06       NO, MUST BE ONE BYTE TOKEN
       INCT R5           BUMP PAST INTEGER
FNDV06 INC  R5           NEXT BYTE
       JMP  FNDV00       LOOP
XIT    MOV  R1,R1        SET STATUS REG
       B    *R11         RETURN
```

Disk 16. Contents of File LTABLES

```
*LTABLES-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
*****************************************************************
* TABLE FOR TOKEN LIST                                          *
* EACH DATA ENTRY - 1st BYTE = TOKEN #, 2nd = LENGTH OF TOKEN *
* TABLE ENDED WITH DATA >FFFF                                   *
* DOUBLE COLON MUST BE THE 1st ENTRY AND OCCUPY 5 BYTES         *
* REM AND ! MUST BE THE LAST 2 ENTRIES                          *
*****************************************************************
TABLE  BYTE >82,3
       TEXT ' ::'
       BYTE >84,3
       TEXT 'IF '
       BYTE >B0,6
       TEXT ' THEN '
       BYTE >81,6
       TEXT ' ELSE '
       BYTE >8C,4
       TEXT 'FOR '
       BYTE >B1,4
       TEXT ' TO '
       BYTE >96,5
       TEXT 'NEXT '
       BYTE >C4,3
       TEXT ' / '
       BYTE >BE,3
       TEXT ' = '
       BYTE >C1,3
       TEXT ' + '
       BYTE >C2,3
       TEXT ' - '
       BYTE >B6,1
       TEXT ')'
       BYTE >B7,1
       TEXT '('
       BYTE >B8,3
       TEXT ' & '
       BYTE >BA,4
       TEXT ' OR '
       BYTE >BB,5
       TEXT ' AND '
       BYTE >BF,3
       TEXT ' < '
       BYTE >C0,3
       TEXT ' > '
       BYTE >B3,3
       TEXT ' , '
       BYTE >B4,3
       TEXT ' ; '
```

```
BYTE >B5,3
TEXT ' : '
BYTE >C3,3
TEXT ' * '
BYTE >A2,8
TEXT 'DISPLAY '
BYTE >9C,6
TEXT 'PRINT '
BYTE >F0,2
TEXT 'AT'
BYTE >9D,5
TEXT 'CALL '
BYTE >A4,7
TEXT 'ACCEPT '
BYTE >92,6
TEXT 'INPUT '
BYTE >93,5
TEXT 'DATA '
BYTE >94,8
TEXT 'RESTORE '
BYTE >9F,5
TEXT 'OPEN '
BYTE >A0,6
TEXT 'CLOSE '
BYTE >9B,3
TEXT 'ON '
BYTE >97,5
TEXT 'READ '
BYTE >8A,4
TEXT 'DIM '
BYTE >86,5
TEXT 'GOTO '
BYTE >87,6
TEXT 'GOSUB '
BYTE >88,7
TEXT 'RETURN '
BYTE >AA,7
TEXT 'LINPUT '
BYTE >BD,5
TEXT ' NOT '
BYTE >B2,6
TEXT ' STEP '
BYTE >D6,4
TEXT 'CHR$'
BYTE >D5,3
TEXT 'LEN'
BYTE >D8,4
TEXT 'SEG$'
BYTE >DA,3
TEXT 'VAL'
BYTE >DB,4
```

```
        TEXT 'STR$'
        BYTE >DC,3
        TEXT 'ASC'
        BYTE >E1,4
        TEXT 'RPT$'
        BYTE >E8,7
        TEXT 'NUMERIC'
        BYTE >E9,5
        TEXT 'DIGIT'
        BYTE >EA,6
        TEXT 'UALPHA'
        BYTE >EB,5
        TEXT ' SIZE'
        BYTE >EC,3
        TEXT 'ALL'
        BYTE >ED,6
        TEXT 'USING '
        BYTE >EE,6
        TEXT ' BEEP '
        BYTE >EF,6
        TEXT 'ERASE '
        BYTE >89,4
        TEXT 'DEF '
        BYTE >95,9
        TEXT 'RANDOMIZE'
SUB     BYTE >A1,4
        TEXT 'SUB '
        BYTE >A7,8
        TEXT 'SUBEXIT '
        BYTE >A8,6
        TEXT 'SUBEND'
        BYTE >A5,6
        TEXT 'ERROR '
        BYTE >A6,8
        TEXT 'WARNING '
        BYTE >FD,1
        TEXT '#'
        BYTE >A9,4
        TEXT 'RUN '
        BYTE >D7,3
        TEXT 'RND'
        BYTE >F3,9
        TEXT 'VARIABLE '
        BYTE >F4,9
        TEXT 'RELATIVE '
        BYTE >F5,8
        TEXT 'INTERNAL'
        BYTE >F6,10
        TEXT 'SEQUENTIAL'
        BYTE >F7,6
```

```
TEXT 'OUTPUT'
BYTE >F8,6
TEXT 'UPDATE'
BYTE >F9,6
TEXT 'APPEND'
BYTE >FA,6
TEXT 'FIXED '
BYTE >FC,3
TEXT 'TAB'
BYTE >BC,5
TEXT ' XOR '
BYTE >FE,9
TEXT ' VALIDATE'
BYTE >C5,1
TEXT '^'
BYTE >CA,3
TEXT 'EOF'
BYTE >CB,3
TEXT 'ABS'
BYTE >CC,3
TEXT 'ATN'
BYTE >CD,3
TEXT 'COS'
BYTE >CE,3
TEXT 'EXP'
BYTE >CF,3
TEXT 'INT'
BYTE >D0,3
TEXT 'LOG'
BYTE >D1,3
TEXT 'SGN'
BYTE >D2,3
TEXT 'SIN'
BYTE >D3,3
TEXT 'SQR'
BYTE >D4,3
TEXT 'TAN'
BYTE >D9,3
TEXT 'POS'
BYTE >DD,2
TEXT 'PI'
BYTE >DE,4
TEXT 'REC '
BYTE >DF,3
TEXT 'MAX'
BYTE >E0,3
TEXT 'MIN'
BYTE >8B,4
TEXT 'END '
BYTE >8D,4
TEXT 'LET '
```

```
       BYTE >8E,6
       TEXT 'BREAK '
       BYTE >8F,8
       TEXT 'UNBREAK '
       BYTE >90,5
       TEXT 'TRACE'
       BYTE >91,7
       TEXT 'UNTRACE'
       BYTE >98,5
       TEXT 'STOP '
       BYTE >99,7
       TEXT 'DELETE '
       BYTE >9E,7
       TEXT 'OPTION '
       BYTE >A3,6
       TEXT 'IMAGE '
       BYTE >F1,4
       TEXT 'BASE'
       BYTE >FB,9
       TEXT 'PERMANENT'
       BYTE >85,3
       TEXT 'GO '
REM    BYTE >9A,4
       TEXT 'REM '
REM1   BYTE >83,2
       TEXT ' !'
HEXFF  BYTE >FF,>FF
***********************************
* TABLE FOR SCREEN INPUT POSITIONS *
* TABLE ENTRIES ARE 2 WORDS        *
* FIRST IS CURSOR POSITION         *
* SECOND IS MAX # OF CHARS ALLOWED *
***********************************
       EVEN
PTABLE DATA 89,1         PROGRAM LIST Y OR N
       DATA 121,1        VARIABLE LIST  "
       DATA 153,1        LINE REF       "
       DATA 205,56       OUTPUT DEVICE
       DATA 313,5        BEGINNING LINE
       DATA 345,5        ENDING LINE
       DATA 409,3        LINE LENGTH
       DATA 441,2        LEFT MARGIN
       DATA 473,3        PAGE LENGTH
       DATA 505,1        TOP & BOTTOM MARGIN
       DATA 561,35       SPECIAL CODES
       DATA 666-32,1        CODES EACH LINE
       DATA >FFFF        END OF TABLE
*****************************
* TABLE OF MAX INPUT VALUES *
* FROM FSTLIN TO SPEC CODE  *
```

```
****************************
MAXTBL DATA >7FFF,>7FFF,132,20,255,9,255
```

Disk 16. Contents of File MAINLIST

```
*MAINLIST-PART OF NEATLIST, A PUBLIC DOMAIN PROGRAM BY DANNY MICHAEL
LIST   LWPI REG1             SET REGISTER POINTER
       LI   R1,GPLWS         R1 POINTS TO GPL REGISTERS
       LI   R2,GPLBUF        R2 POINTS TO BUFFER
       LI   R3,16            R3 = WORD COUNT
MOVE   MOV  *R1+,*R2+        MOVE WORD & INC POINTERS
       DEC  R3               COUNT -1
       JNE  MOVE             LOOP IF NOT DONE
       LI   R0,PABLOC        R0 POINTS TO PAB SPACE IN VDP RAM
       LI   R1,PABBUF        R1 POINTS TO RAM BUFFER
       LI   R2,212           R2 = BYTE COUNT
       BLWP @VMBR            MOVE TO BUFFER
       MOVB @GRMRA,@GROMSV   MOVE GROM
       NOP                     ADDRESS TO
       MOVB @GRMRA,@GROMSV+1   GROMSV
       DEC  @GROMSV          CORRECT ADDRESS
       CLR  @PABFLG          CLEAR OPEN FILE FLAG
       LI   R0,>8C1F         ERROR SOUND CODE
       MOVB R0,@>8400        TO SOUND
       SWPB R0
       MOVB R0,@>8400          PORT
*
       BLWP @MOVBUF
LIST01 LI   R1,PTABLE        R1 POINTS TO SCREEN POS. TABLE
LIST02 BLWP @KEY             GET KEYBOARD INPUT
       MOVB @KEYVAL,R2       TERMINATOR TO R2
       CB   R2,@QUIT         FCTN =?
       JNE  LIS01            NO
       B    @EXIT            QUIT
LIS01  CB   R2,@PRCD         PROCEED?
       JEQ  LIST03           YES, CHECK PARAMETERS
       CB   R2,@UA           UP ARROW?
       JNE  LIS03            NO
       AI   R1,-4            LAST POSITION
       CI   R1,PTABLE        ALREADY AT FIRST POS?
       JL   LIST01           YES, STAY THERE
       JMP  LIST02           NO, GO TO POS -1
LIS03  AI   R1,4             MUST BE ENTER OR DOWN ARROW
       MOV  *R1,R3
       CI   R3,>FFFF         CHECK FOR END OF TABLE
       JEQ  LIST01           END OF LIST, START OVER
       JMP  LIST02           NEXT ONE
LIST03 BLWP @PARAM           CHECK PARAMETERS
       JNE  LIST04           NO ERRORS
LIS02  BL   @ERROR1          PRINT ERROR MSG
       MOV  R4,R1            PTABLE POSITION
       JMP  LIST02           TRY AGAIN
*
```

```
LIST04 C    @ENDTBL,@BEGTBL CHECK FOR PROGRAM IN MEMORY
       JNE  XXXX            LIST PROGRAM
       B    @EXIT           END
XXXX   MOV  @TBMGN,R3       MARGIN LENGTH
       S    R3,@PGLEN       ADJUST FOR
       S    R3,@PGLEN        MARGINS
       MOV  @PGLEN,R14      PAGE LENGTH TO 14
       MOVB @DOWHAT,R1      INSTRUCTION BYTE
       ANDI R1,>0100        CHECK FOR LIST BIT
       JNE  LSTR01          LIST PROGRAM
       B    @LIST06         SKIP LIST
LSTR01 LI   R1,6             SET-UP
       MOV  R1,@INDENT       TABS
       LI   R1,11            FOR
       MOV  R1,@LONGLN       PRINTER
       CLR  @TMPLEN         CLEAR BUFFER COUNT
       CLR  @REMFLG         CLEAR REMARK FLAG
       MOV  R3,R3           TEST FOR NULL MARGIN
       JEQ  LSTR02
       CLR  R14              AND DO
       BL   @S08             FORMFEED
LSTR02 MOV  @ENDTBL,R4      GET END OF LINE# TBL
       AI   R4,-3           FIRST ENTRY
LSTR03 MOVB @1(R4),R5
       SWPB R5
       MOVB *R4,R5
       C    R5,@FSTLIN      CHECK FOR FIRST LINE
       JHE  LSTR04          GOT IT
       AI   R4,-4           NEXT ENTRY
       JMP  LSTR03          LOOP TILL RIGHT
LSTR04 MOVB @1(R4),R5
       SWPB R5
       MOVB *R4,R5
       C    R5,@LSTLIN      PAST LAST LINE?
       JLE  LSTR99          YEP, WE'RE DONE
       B    @LSTR14
LSTR99 MOVB *R4+,@VBUF      SET UP FOR
       MOVB *R4+,@VBUF+1    CONVERT
       BLWP @HEXASC         CONVERT TO ASCII
       CLR  @SPACES         ZERO INDENT
       CLR  @TMPLEN         ZERO BUFFER COUNT
       LI   R0,VBUF         POINT TO LINE # STRING
       LI   R1,6            COUNT
       BL   @PRINT          PRINT LINE #
       MOVB @1(R4),R5       MAKE R5 POINT TO LINE
       SWPB R5
       MOVB @0(R4),R5
LSTR06 MOVB *R5+,R2         TOKEN TO R2
       JNE  LSTR05          GO IF NOT LINE END
* HERE IF END OF LINE
       BL   @SENDIT         PRINT LINE
```

```
       CLR  @TMPLEN          BUFFER EMPTY NOW
       AI   R4,-6            NEXT TABLE ENTRY
       C    R4,@BEGTBL       END OF PMG?
       JL   LSTR14           YES, RETURN
       JMP  LSTR04           GET NEXT LINE
LSTR05 LI   R1,>80C7
       CB   R2,R1
       JL   LSTR18
       SWPB R1
       CB   R1,R2            QUOTED STRING?
       JNE  LSTR07           NO
       LI   R0,QUOTE         POINT TO QUOTE STRING
       LI   R1,1             COUNT
       BL   @PRINT           PRINT QUOTE
       BL   @STRING          SEND STRING
       LI   R0,QUOTE          CLOSE
       LI   R1,1
       BL   @PRINT            QUOTES
       JMP  LSTR06           NEXT BYTE
LSTR07 LI   R1,>C8C9
       CB   R1,R2            STRING?
       JNE  LSTR08           NO
       MOVB *R5,R1           COUNT
       SRL  R1,8             TO R1
       CI   R1,16            MAX VARIABLE
       JH   LSTX01           TOO LONG
       INC  R5               POINT TO STRING
       MOV  R5,R0            SO DOES R0
       A    R1,R5            BUMP PAST STRING
       BL   @PRINT           PRINT IT
       JMP  LSTR06           NEXT BYTE
LSTX01 BL   @STRING          SEND STRING
       JMP  LSTR06           NEXT BYTE
LSTR08 SWPB R1
       CB   R1,R2            INTEGER?
       JNE  LSTR09           NO
       MOVB *R5+,@VBUF       SET-UP
       MOVB *R5+,@VBUF+1     FOR CONVERT
       BLWP @HEXASC          CONVERT TO STRING
       LI   R1,5             MAX COUNT
       LI   R0,VBUF          POINT TO STRING
LSTR11 CB   *R0,@ZERO        LEADING ZERO?
       JNE  LSTR10           NO, PRINT IT
       DEC  R1               COUNT -1
       INC  R0               BUMP POINTER
       JMP  LSTR11           LOOP
LSTR10 BL   @PRINT           PRINT IT
       JMP  LSTR06           NEXT BYTE
LSTR09 LI   R0,TABLE         POINT TO TOKEN TABLE
       SETO R3
```

```
LSTR12 CB   *R0,R3          END OF TABLE?
       JEQ  LSTR15          YES, NO MATCH
       CB   *R0+,R2         MATCH?
       JEQ  LSTR13          YES, PRINT IT
       MOVB *R0+,R1         COUNT TO R1
       SRL  R1,8            TO RIGHT
       A    R1,R0           POINT TO NEXT ENTRY
       JMP  LSTR12          LOOP
LSTR15 LI   R0,SPACE        UNRECOGNIZED CHAR
       LI   R1,1            COUNT
       BL   @PRINT          PRINT SPACE
       JMP  LSTR06          NEXT BYTE
LSTR13 MOVB *R0+,R1         COUNT TO R1
       SRL  R1,8            TO RIGHT
       CI   R0,REM          REMARK?
       JL   LSTR19          NO
       SETO @REMFLG         SET FLAG
LSTR19 BL   @PRINT          PRINT TOKEN
       MOV  @REMFLG,R3      CHECK FOR REMARK
       JNE  LSTR20          DO REM
       JMP  LSTR06          NEXT BYTE
LSTR14 MOV  R14,R3          LINES REMAINING
       MOV  @TBMGN,R1
       JEQ  LSTR23
       INC  R1
LSTR23 A    R1,R3           ADD IN MARGIN
       INC  R3               SET-UP
       CLR  R14              AND DO
       BL   @S08             FORMFEED
       JMP  LIST06
LSTR18 DEC  R5              BACK UP 1
       MOV  R1,R3           CHECKER
       CLR  R1              CLEAR COUNT
       MOV  R5,R0           R0 POINTS TO VAR. NAME
LSTR17 CB   *R5,R3          NON ALPHA?
       JHE  LSTR16
       MOVB *R5,*R5         CHECK FOR ZERO
       JEQ  LSTR16
       INC  R5              POINTER +1
       INC  R1              ADD TO COUNT
       JMP  LSTR17          LOOP
LSTR16 BL   @PRINT          SEND IT
       B    @LSTR06         RETURN
LSTR20 MOV  R5,R0           POINTER TO R0
LSTR22 MOVB *R0,R3          CHECK FOR END
       JNE  LSTR21          NOT END
       MOV  R0,R5           RESTORE POINTER
       CLR  @REMFLG         CLEAR REMARK FLAG
       B    @LSTR06         GO BACK
LSTR21 LI   R1,1            COUNT
       BL   @PRINT          PRINT IT
```

```
       JMP  LSTR22          LOOP
LIST06 MOVB @DOWHAT,R1      INSTRUCTION BYTE
       ANDI R1,>0200        CHECK LIST BIT
       JEQ  EXIT            NO, MAY AS WELL QUIT
       BL   @REFR           DO REFRENCE
LIST05 JMP  EXIT
*****************
* ERROR ROUTINE *
*****************
ERROR1 LI   R0,704-32       LAST LINE ON SCREEN
       MOVB *R3+,R2         BYTE COUNT TO R2
       SRL  R2,8            ALIGN
ERROR3 MOVB *R3+,R1         MOVE BYTE TO R1
       AB   @OFFSET,R1      ADD OFFSET
       BLWP @VSBW           WRITE IT!
       DEC  R2              COUNT -1
       JEQ  ERROR4          DONE
       INC  R0              NEXT SCREEN LOCATION
       JMP  ERROR3          GET NEXT CHARACTER
ERROR4 LI   R0,>9000        TURN ON BYTE
       MOVB R0,@>8400         TO SOUND PORT
       LI   R0,>3000        DELAY COUNT
ERROR5 DEC  R0              COUNT -1
       JNE  ERROR5          LOOP TILL DONE
       LI   R0,>9F00        TURN OFF BYTE
       MOVB R0,@>8400         TO SOUND PORT
       B    *R11            RETURN
****************************************
* EXIT ROUTINE, RESTORES ALL USED MEMORY *
****************************************
EXIT   MOV  @PABFLG,@PABFLG CHECK FOR OPEN FILE
       JEQ  EXIT1           NO OPEN FILE
       LI   R0,PABLOC
       MOVB @CLOSE,R1
       BLWP @VSBW
       MOV  @SVPNTR,@PNTR
       BLWP @DSRLNK
       DATA 8
EXIT1  BLWP @MOVBUF
       MOVB @GROMSV,@GRMWA  RESTORE
       NOP
       MOVB @GROMSV+1,@GRMWA  GROM ADDRESS
       LI   R0,PABLOC       R0 POINTS TO PAB SPACE IN VDP RAM
       LI   R1,PABBUF       R1 POINTS TO RAM BUFFER
       LI   R2,212          R2 = BYTE COUNT
       BLWP @VMBW           MOVE FROM BUFFER
       LI   R1,GPLBUF       R1 POINTS TO BUFFER
       LI   R2,GPLWS        R2 POINTS TO GPL REGISTERS
       LI   R3,16           R3 = WORD COUNT
MOVE1  MOV  *R1+,*R2+       MOVE WORD & INC POINTERS
```

```
       DEC  R3              COUNT -1
       JNE  MOVE1           LOOP IF NOT DONE
       CLR  R0              R0=0
       MOVB R0,@STATUS      CLEAR STATUS BYTE
       LWPI GPLWS           RESTORE REGISTER POINTER
       LIMI 2
       B    @>0070          RETURN TO BASIC
VARBUF BSS  900             BUFFER FOR VAR POINTERS
ENDBUF EQU  $
```

Disk 16. Contents of File NEATDOC

NEATLIST INSTRUCTIONS

The disk you received contains several files. The two that are of the most concern are NEATLIST and LISTOBJ. NEATLIST is a loader program that loads the LISTOBJ file which contains the actual object code for the NEATLIST utility. To load the program, first select EXTENDED BASIC from the master menu, then issue the statement CALL INIT. It is important that you do not omit this step. Next, place the NEATLIST disk into disk drive #1, and CALL LOAD("DSK1.NEATLIST"). This will load the program, and return to the blinking cursor prompt. NEATLIST is now ready for use, and will remain useable until you exit EXTENDED BASIC, or issue another CALL INIT statement. To use NEATLIST, first load the program that you wish to list (OLD DSKx.filename). When you want to list the program, do a CALL LINK("LIST"). This will take you to the NEATLIST option screen. In order to understand the following explanation of your options, I suggest that you load the NEATLIST program and go to the options screen while reading the following section. The following command line will get you to this point: CALL INIT::CALL LOAD("DSK1.NEATLIST")::CALL LINK("LIST")

At this point, you should be looking at the NEATLIST option screen. The information in each of the input fields are the default values of the program. These defaults produce an acceptable listing on most printers, however your printer or personal preference may require different values in some locations. (You may change these default values if you wish. See the section on changing defaults.) The following keys on the keyboard are active: All letters and numbers, the arrow keys (**FCTN E**, **FCTN S**, **FCTN D**, and **FCTN X**), the **ENTER** key, **PROC'D (FCTN 6)**, and **BACK (FCTN 9)**. The arrow keys will move the cursor in the direction of the arrow, the **ENTER** key moves the cursor to the next input field, **BACK** will return you to basic without producing a printout, and **PROC'D** will cause the printing process to begin. If the cursor is in the last input field on the screen, and you press **ENTER** or the down arrow key, the cursor will move to the first position on the screen. If the cursor is in the first input field on the screen, and you press the up arrow key, nothing will happen.

The first three options deal with what the program will output, and should be answered with either a **Y** or an **N** for yes or no.

PROGRAM ? — a Y in this field will cause the program to output a listing of your Extended Basic program. An N here will omit the listing. Please note that you CAN enter any letter or number in this field, however an error will occur if you enter anything other than a Y or N. Error checking is not done until you press **PROC'D**.

VARIABLES ? — a Y in this field will cause the program to output a list of all variables in your program. If you also entered a Y in the PROGRAM ? field, the variable list will be printed after the program listing. Again, the only acceptable letters are Y or N.

LINE REF ? — a Y here will result in a list of line numbers after each variable in the variable list, provided you also entered a Y in the VARIABLES ? field. The line numbers represent the program lines which contain the referenced variable.

The OUTPUT TO: field is where you tell the program where to send the program listing and/or the variable reference list. This may be any valid output device except for the screen or cassette. Most of the time it will be a printer, however the listing may be sent to a disk file if you desire. The output is in DISPLAY VARIABLE 254 format. If you direct the output to a device connected to the RS232 card, i.e. PIO or RS232, you must use the .CR option.

The next two options determine how much of your basic program is listed, or referenced for variables. The listing will begin with the line number specified at BEGINNING LINE, or if that is not a valid line number, with the next higher numbered line. Listing will cease at the line specified at ENDING LINE, or if that is not a valid line number, with the last line numbered lower than the one specified. If you have asked for a variable reference, the variables listed will be only the ones contained in the lines between BEGINNING LINE and ENDING LINE. One note - The NEATLIST utility will list variables contained in subprograms separately from those in the main program. If your STARTING LINE happens to be a line inside a subprogram, the variables in that subprogram will be shown as VARIABLES USED IN MAIN PROGRAM. The variables will be correct, just the title will be wrong.

The information entered in the next four fields determines how the listing is formatted on the page.

LINE LENGTH tells the program how many characters to print per line. You may use this to create a right margin if you desire. If your printer prints 80 characters per line, and you want an 8 space right margin on the printout, just enter 72 at the LINE LENGTH prompt. Be sure not to enter a LINE LENGTH longer than your printer is capable of printing. If you do that, it will not be possible for NEATLIST to properly format the listing.

LEFT MARGIN is the number of blank spaces placed before each line is printed. Using a value of around 5 here will allow you to place your printout in a binder and still be able to read the left-most characters.

PAGE LENGTH tells NEATLIST how many printed lines there are per page. On a standard 11-inch page there are 66.

TOP & BOTTOM MARGIN is the number of blank lines left at the top and bottom of the page. If you do not want your program listing in pages, enter a zero here. The maximum values for all but TOP & BOTTOM MARGIN are shown on the options page (max. for T&B MARGIN is 9), however LINE LENGTH and PAGE LENGTH both have minimum values. The minimum for LINE LENGTH is 30 plus the value entered for LEFT MARGIN. PAGE LENGTH must be at least 2 times TOP & BOTTOM MARGIN plus 2.

The last two option fields allow you to send control codes to your printer. At the SPECIAL CODES: prompt, enter the decimal control codes that you want to send to your printer. If you enter more than one code, (you may have up to nine), separate them with a comma. Be sure not to leave a blank space between codes (more on that later). For instance if you have an EPSON or GEMINI printer and you put 27,69 in the SPECIAL CODES: field, your listing will be made in emphasized print. A 15 in this field will result in condensed print. Using this code, and entering 132 at the LINE LENGTH prompt will result in a very neat looking condensed listing. The OUTPUT CODES EACH LINE? prompt must be answered with either a Y or N. This determines when the special codes will be sent to the printer. An N in this field will result in the special code(s) being sent to the printer only once, and that will be at the beginning of the first line that contains printing characters. In other words, the special codes are NOT sent in the top and bottom margins. A Y in this field will output the special codes at the beginning of every line in the listing that contains printing characters. On some printers, some of the special print modes revert to normal print at the end of each line. That's the reason for allowing you to send the codes on every line. On most printers, the expanded print code (14) will have to be sent each line.

Now a little bit on how the program reads your input. The values that you enter in each field are not checked for errors until you press **PROC'D (FCTN 6)**. At that time, your entries are read from the screen. For this reason, it is not necessary to press the **ENTER** key at each prompt. You may use the arrow keys to move to the fields that you wish to change. All input fields will accept a limited number of characters. The ones that are answered with either Y or N will accept only one. The lengths of the others vary, depending on the maximum values allowed. In the OUTPUT TO: field, the program takes everything you enter up to the first space character. For example, if your last listing went to a disk file, in the next access to the NEATLIST program the OUTPUT TO: field might look like this:

```
OUTPUT TO: DSK1.LISTING
```

If you wanted to direct output to the printer again, you could change it to look like this:

```
OUTPUT TO: PIO.CR STING
```

Since the program only reads up to a space, it will direct the listing to PIO without giving an error message.

In the fields that require numeric input, the program reads everything up to a non-numeric character. For example, to change the default ending line number from 32767 to 100, you can enter 100 <space> and leave the input field looking like this: 100 7

Again, the program reads up to the first non-numeric character (a space in this example). The one exception to this is the SPECIAL CODES: field. Since you can have more than one number in this field, the program reads each number up to a non-numeric character. If that character is a space, no more is read from that field. If the non-numeric character is anything other than a space, the program looks for another number. I suggest that you use a comma to separate the numbers.

When you have the options page looking like you want it, press **PROC'D (FCTN 6)**, and your program listing and/or variable list will be sent to the specified output device provided there are no errors in any of the input fields. If there isn't a basic program in memory, the program will return to basic. Pressing **BACK (FCTN 9)** will return you to basic without outputting a listing. The screen will be restored to whatever was on the screen when the CALL LINK("LIST") statement was issued. You may also use **BACK** to abort the listing output after you have started it. If you have directed the output to a device connected to the RS232 card, you may also halt the listing by pressing **CLEAR (FCTN-4)**.

At this point I suggest that you experiment with the keyboard at the options page to get the feel of how the program works. One thing to keep in mind — if the basic program you are working on contains a CALL INIT statement, it will wipe out the pointers to the NEATLIST utility when you run it, which will require that you reload NEATLIST. To get around this, I suggest that you replace all CALL INIT statements with the following line:

```
CALL PEEK(8198,A)::IF A<>170 THEN CALL INIT
```

This line will only CALL INIT if it has not been called since entering extended basic.

ERROR MESSAGES

The program checks for errors when you press **PROC'D**. If an invalid entry has been made on the options page, an error tone is generated, an error message is printed on the line below the OUTPUT CODES EACH LINE? field, and the cursor will be blinking in the field where the bad entry was made. You may then correct the entry and try again. The error message you will probably see the most of is ** BAD PARAMETER **. This occurs when you enter a wrong letter in the Y or N fields, or enter a number above the maximum limit. There is also a ** PAGE LENGTH TOO SHORT ** and a ** LINE LENGTH TOO SHORT ** message. A ** BAD DEVICE NAME ** error occurs if the output device you specified is invalid. If an error occurs while the program is outputting the listing an ** I/O ERROR ** message is printed. This is more likely to happen when you have directed the output to a disk file. If the disk is write protected, or the disk fills up, or there is a write error while outputting, the ** I/O ERROR ** message is printed. Specific error code numbers are not printed. One error message that you probably won't see is the ** TOO MANY VARIABLES ** message. NEATLIST can reference up to 300 variables in a program or subprogram. This should be sufficient for most programs.

CHANGING PROGRAM DEFAULTS

As with most utility programs that have a lot of input variables, the default values of the NEATLIST program will probably not suit your particular tastes. For this reason, the DEFAULT program is furnished on your disk. This program allows you to set up NEATLIST to load with your favorite values already in place. The DEFAULT program is easy to use, but it does have rigid requirements. If you follow these instructions to the letter, you'll have no problems.

1. Load NEATLIST into memory with the following line:

```
CALL INIT::CALL LOAD("DSK1.NEATLIST")
```

2. Place a disk containing both the LISTOBJ file and the DEFAULT program into disk drive #1.

3. Access the NEATLIST options page:

```
CALL LINK("LIST").
```

4. Enter your favorite values on the options page, then exit by pressing **BACK**.

5. Issue the following command .. CALL LOAD("DSK1.DEFAULT"). The DEFAULT program will load and automatically execute. When finished, it will return to the blinking cursor prompt. If any errors occur, an error message will be printed, and you should start over again.

WARNING! The DEFAULT program loads into the section of memory that contains extended basic programs. Be sure to save any important program before loading the DEFAULT program!

If for some reason you would rather load the NEATLIST utility from a disk drive other than #1, the DRIVE# program will change the loader program to load from any drive you specify. DRIVE# is an extended basic program. All you have to do is RUN"DSK1.DRIVE#", and follow the instructions on the screen. Remember . . . the DEFAULT program changes the LISTOBJ file, and the DRIVE# program changes the NEATLIST loader.

MEMORY USAGE AND PROGRAM MODIFICATION

NEATLIST occupies most of the lower section of the 32K memory expansion, and is not relocatable when loaded with the NEATLIST loader program. Specifically, the area of memory used is from >24F4 through >3D9A. That leaves only about 600 bytes for other assembly language programs.

The bulk of the files on your disk are the source code for the NEATLIST utility. The source code is fully commented to assist you in case you want to modify the program, or just understand how it works. A brief explanation of each file:

LIST/TXT is the file that you would assemble. It contains COPY directives for the other files.

LINIT contains program equates, data, and buffer space.

LTABLES contains the basic token lookup table, a pointer table to the input field positions, and a table of the maximum input values for numeric fields.

LMOVBUF contains the option page screen and the code to swap it with the basic screen, and restore the basic screen at program completion.

LDSRLNK is the routine that outputs to the specified device.

LKEY reads the keyboard for input while in the options page.

LPARAM is the routine that checks the parameters you enter for errors, and sets up necessary program variables.

LPRINT contains the code for formatting the listing, keeping up with # of lines printed, etc.

LREF is the routine that does the variable reference.

MAINLIST is the controlling program. It also does the program listing.

If you make modifications to the program, it will not load with the NEATLIST loader program. You will have to load it as any other tagged object code file. The only drawback to this is loading time. . . about 2 minutes. Remember not to use the C option when assembling, as extended basic will not load compressed object code.

PERSONAL NOTES

A great deal of my time went into developing this program, and I think you will find it a unique and useful utility. As far as I know, it is the only one of it's kind for the TI-99/4A. As the loader screen says, please pass along a copy of this program to anyone who wants it. It is not copyrighted or protected in any way, however, I won't gripe if you decide to pay for it.

Users groups are given my permission to place this program in their libraries subject to the following conditions:

1. That the program be made available to members at NO CHARGE!

2. That the source code be made available along with the object code and loader.

I would enjoy hearing your comments on the program, especially from anyone who has made useful modifications. I would also like to know what other assembly language utility programs you would like to see marketed under the "FREEWARE" concept. You may contact me at:

Danny Michael
Rt. 9, Box 460
Florence, AL 35630

PHONE (205)764-7881

or EMAIL me on CompuServe, ID# 75116,1225

Disk 17. Screen Dump

Version:
Author: Danny Michael
Requires: XB or EA
Language: AL
Updated:

Dump graphics screen to Epson compatible printers at a key press. Does double-size, rotated, and inverse in any combination. Very fast. Includes source code. Works with load interrupt switch as well.

```
dskdir. v2.0. 12-dec-96
Disk name               = SCREENDUMP
Sectors total           = 360
Sectors used            = 308
Sectors available       = 50
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 DUMPCB       15 DIS/FIX 80   >022 014
002 >003 DUMPCB/TXT   46 DIS/VAR 80   >030 045
003 >004 DUMPDEFALT   15 PROGRAM      >05d 014
004 >005 DUMPDOC      65 DIS/VAR 80   >06b 064
005 >006 DUMPDOCBAS    3 PROGRAM      >0ab 002
006 >007 DUMPXB       30 DIS/FIX 80   >0ad 029
007 >008 DUMPXB/TXT   62 DIS/VAR 80   >0ca 061
008 >009 DUMP_MAIN    72 DIS/VAR 80   >107 071
```

Disk 17. Contents of File DUMPCB/TXT

```
       TITL 'SCREEN DUMP FOR EPSON & GEMINI PRINTERS'
*            CONSOLE BASIC - VERSION 3
************************************
* THIS PROGRAM WRITTEN DEC. 1983   *
* MODIFIED DEC. 1984               *
* BY : DANNY MICHAEL               *
*      RT. 9, BOX 460              *
*      FLORENCE, AL.               *
*              35630               *
*                                  *
*    TELEPHONE:  (205) 764-7881    *
*                                  *
* COMPUSERVE ID#: 75116,1225       *
*                                  *
* PROGRAM RELEASED INTO THE PUBLIC *
* DOMAIN DEC. 1984.                *
*                                  *
* THIS PROGRAM NOT TO BE SOLD!!    *
*                                  *
* ANYONE WHO WISHES MAY HAVE IT    *
* FREE.                            *
*                                  *
* MODIFIED MAY 1985 BY D.C. WARREN *
* POMONA, CA FOR FCTN 0 OUTPUT.    *
************************************
       DEF  DUMP,SET,CART,CKISR  DEFINE ENTRY POINT
       REF  GRMRA,GRMWA,VMBR,VMBW,VSBR
       REF  VSBW,KSCAN,GPLWS,DSRLNK
PABLOC EQU  >1000           PAB LOCATION IN VDP RAM
COUNT  EQU  PABLOC+5        POINTER TO BYTE COUNT BYTE
BUFFER EQU  PABLOC+72       READ/WRITE BUFFER IN VDP RAM
STATUS EQU  >837C           STATUS REGISTER
PNTR   EQU  >8356           POINTER BYTE FOR DSRLNK
FRSTHI EQU  >2024           ADDRESS OF FIRST AVAIL ADDRESS
PTABLE DATA >0400           BEGINNING ADDRESS OF PATTERN TABLE
F      DATA 15              MASK BITS
SXTEEN DATA 16              DITTO
INVERT DATA 0               FLAG FOR INVERTED PRINT
DOUBLE DATA 0               FLAG FOR DOUBLE SIZE PRINT
ROTATE DATA 0               FLAG FOR ROTATED PRINT
FLAG   DATA 0               FLAG WORD FOR VARIOUS ROUTINES
RESET  DATA >0D0A,>1B32     RESET DATA FOR PRINTER
REG1   BSS  >20             SPACE FOR REGISTER SET 1
GPLBUF BSS  >B6             SPACE TO SAVE FAC THRU GPL REGISTERS
PABBUF BSS  212             SPACE TO SAVE VDP RAM USED IN PMG
GROMSV BSS  2               SPACE TO SAVE GROM ADDRESS
DSRBUF BSS  8               BUFFER FOR DSRLNK
BUF1   BSS  8               BUFFER FOR CHAR PATTERNS
```

```
BUF2   BSS  16              BUFFER FOR PRINT CODES
SVPNTR BSS  2               BUFFER USED IN DSRLNK CALLS
SPNTR  BSS  2               SCREEN LOCATION POINTER
SPNTR1 BSS  2               FOR ROTATE ROUTINE
BPNTR  BSS  2               BUFFER LOCATION POINTER
*
* DATA FOR PERIPHERAL ACCESS BLOCK
*
PDATA  DATA >0012,BUFFER,>FF00,0,>0020
****************************************
* TO CHANGE OUTPUT DEVICE, TYPE IN YOUR   *
* PRINTER DEVICE NAME BETWEEN THE ' MARKS *
* IN THE NEXT LINE, THEN SAVE THIS FILE   *
* AND RE-ASSEMBLE. BE SURE THAT THERE ARE *
* 32 POSITIONS BETWEEN THE ' MARKS. PAD   *
* YOUR PRINTER NAME WITH BLANKS ON THE    *
* RIGHT SIDE TO MAKE 32 POSITIONS.        *
****************************************
       TEXT 'PIO.CR                          '
       EVEN              DO NOT DELETE THIS STATEMENT!!!!
*
* DATA TO SET PRINTER UP FOR GRAPHICS
*
SETUP  DATA >0D0A,>1B41,>081B,>4F00
*
* BEGIN NEW LINE DATA
*
CRLF   DATA >0D0A,>1B4B
GRPNUM DATA 1               THE NUMBER OF DOTS PER LINE
*                           WILL BE PLACED HERE
ZERO   DATA 0,0,0,0         DATA FOR BLANK CHARACTER
*
* BYTE VALUES FOR VARIOUS ROUTINES
*
MASK   BYTE >20             MASK TO SET/RESET EQUAL BIT
MASK1  BYTE >80             MASK TO STRIP SCREEN OFFSET
PERIOD BYTE >2E             ASCII FOR PERIOD
VALID  BYTE >AA             DSR ROM VALIDATION BYTE
OFFSET BYTE >60             SCREEN OFFSET
OPEN   BYTE 0               FILE OPEN BYTE
CLOSE  BYTE 1               FILE CLOSE BYTE
WRITE  BYTE 3               FILE WRITE BYTE
LOW    BYTE >80             LOWEST PRINTABLE CHARACTER
HIGH   BYTE >FF             HIGHEST PRINTABLE CHAR
TAB    BYTE 19              TAB SPACES FOR PRINTING
REGTAB BYTE 19              LEFT TAB FOR REGULAR DUMP
ROTTAB BYTE 25                "   "   "  ROTATED   "
DUBTAB BYTE 9                 "   "   "  DOUBLE SIZE DUMP
*
MESG2  BYTE 10              DATA FOR ERROR MESSAGE
       TEXT 'I/O ERROR!'
```

```
*
*BEGINNING OF PROGRAM
*
       EVEN
SET    LWPI REG1             LOAD WS REG
       SETO @FLAG            SET FLAG
       LI   R0,>80FF         FOR LOW & HIGH
       MOVB R0,@LOW          LOWEST CHAR
       SWPB R0               NEXT BYTE
       MOVB R0,@HIGH         HIGHEST BYTE
       LI   R0,>6080         SCREEN OFFSET
       MOVB R0,@OFFSET       SET OFFSET
       SWPB R0               NEXT BYTE
       MOVB R0,@MASK1        SET MASK FOR OFFSET CHARS
       LI   R0,>0400         PATTERN DESCRIPTOR TABLE
       MOV  R0,@PTABLE       IN PLACE
       JMP  DUMP01           CONTINUE
CART   LWPI REG1             LOAD WS REG
       SETO @FLAG            SET FLAG
       SETO R0               R0=>FFFF
       MOVB R0,@HIGH         SET HIGH BYTE
       INC  R0               R0=0
       MOVB R0,@LOW          SET LOW BYTE
       MOVB R0,@OFFSET       CLEAR OFFSET
       MOVB R0,@MASK1        CLEAR MASK FOR CHARS ABOVE >80
       LI   R0,>0800         PATTERN DESCRIPTOR TABLE
       MOV  R0,@PTABLE       IN PLACE
       JMP  DUMP01           CONTINUE
DUMP   MOV  R11,@>83F6       RETURN TO GPL R11
       LWPI REG1              LOAD WS REG
       CLR  @FLAG             CLEAR FLAG
DUMP01 SETO R0                 RESET
       MOVB R0,@TAB            TAB
       INCT R0
       MOV  R0,@GRPNUM        GRAPHICS COUNT FOR NORMAL DUMP
       CLR  @INVERT           RESET
       CLR  @DOUBLE           PROGRAM
       CLR  @ROTATE           FLAGS
****************************
* GET PARAMETERS FROM LINK *
****************************
       MOVB @>8312,R3        # OF ARG. TO R3
       SRL  R3,8             ALIGN TO RIGHT
       MOV  R3,R3            TEST FOR ZERO
       JEQ  TABSET           NO ARGUMENTS
*
* CHECK FOR RAM (MINIMEM) AT >7000
       MOV  @>7000,R1
       INV  @>7000
       MOV  @>7000,R2
```

```
       C    R1,R2
       JEQ  NOMINI
       LI   R7,>7002        ARGUMENT LIST IN MINIMEM
       MOV  R1,@>7000
       JMP  ARG01
NOMINI LI   R7,>200A        ARGUMENT LIST IN ED/ASSM
ARG01  LI   R6,16           OFFSET
       CI   R3,3            MAKE SURE ONLY
       JL   ARG02               TWO ARGUMENTS
       LI   R3,2
ARG02  MOV  @>8310,R0       STACK POINTER TO R0
       A    R6,R0           ADJUST
       LI   R1,DSRBUF       BUFFER POINTER TO R1
       LI   R2,8            BYTE COUNT
       BLWP @VMBR           READ STACK ENTRY
       MOVB *R7,R0          IDENT BYTE TO RO
       SRL  R0,8            ALIGN
       JEQ  NUMEXP          NUMERIC EXPRESSION
       AI   R1,4            POINT TO VALUE POINTER
       CI   R0,2            NUMERIC VARIABLE?
       JEQ  NUMVAR          YES
       CI   R0,4            CHECK FOR >4
       JLT  STRING          MUST BE STRING
       JMP  ARG09           BAD ARGUMENT
NUMEXP MOV  *R1,R4          FIRST WORD TO R4
       MOV  R4,R4           CHECK FOR ZERO
       JNE  ARG03           N0
       MOVB @ZERO,@TAB      SET TAB TO ZERO
       JMP  ARG09           THROUGH WITH THIS ONE
ARG03  SRL  R4,8            GET EXP
       CI   R4,>0040        EXP ONE?
       JNE  ARG09           NO, FORGET THIS ONE
       INC  R1              POINT TO NUMBER
       MOVB *R1,@TAB        MOVE TO TAB POINTER
       JMP  ARG09           GET NEXT ONE
NUMVAR MOV  *R1,R0          VALUE POINTER TO R0
       LI   R1,DSRBUF       POINT TO BUFFER
       BLWP @VMBR           READ VALUE INTO BUFFER
       JMP  NUMEXP          GET VALUE
STRING MOV  *R1+,R0         STRING POINTER TO R0
       MOV  *R1,R2          BYTE COUNT TO R2
       CI   R2,9            CHECK FOR GREATER THAN 8
       JL   ARG04           IT'S OK
       LI   R2,8            MAKE IT OK
ARG04  LI   R1,DSRBUF       R1 POINTS TO BUFFER
       BLWP @VMBR           READ STRING TO BUFFER
ARG08  MOVB *R1,R9          BYTE TO R9
       SRL  R9,8            ALIGN
       CI   R9,73           IS IT "I"?
       JNE  ARG05           NO
       SETO @INVERT         YES, SET FLAG
```

```
       JMP  ARG07           GO ON
ARG05  CI   R9,82           IS IT "R"?
       JNE  ARG06           NO
       MOV  @ROTATE,@ROTATE ALREADY SET?
       JNE  ARG07           YES, SKIP IT
       SETO @ROTATE         YES, SET FLAG
       LI   R15,>C000       GRAPHICS COUNT
       MOV  R15,@GRPNUM     MOVE INTO POSITION
       JMP  ARG07           GO ON
ARG06  CI   R9,68           IS IT "D"?
       JNE  ARG07           NO
       SETO @DOUBLE         YES, SET FLAG
       SETO @ROTATE         ASSURE ROTATED IF DOUBLE
       LI   R15,>8001       GRAPHICS COUNT FOR DOUBLE
       MOV  R15,@GRPNUM     INTO POSITION
ARG07  INC  R1              INC POINTER
       DEC  R2              COUNT -1
       JNE  ARG08           DO AGAIN IF NOT FINISHED
ARG09  DEC  R3              DEC ARG. COUNTER
       JEQ  TABSET          THAT'S ALL
       AI   R6,8            POINT TO NEXT ENTRY
       INC  R7              IDENT POINTER +1
       JMP  ARG02           DO IT AGAIN
******************
* SET TAB SPACING *
******************
TABSET SETO R0              R0=>FFFF
       CB   @TAB,R0         HAS TAB BEEN SET?
       JNE  TAB02           YES, GO ON
       C    R0,@DOUBLE      DOUBLE FLAG SET?
       JNE  TAB00           NO, TRY NEXT ONE
       MOVB @DUBTAB,@TAB    SET TAB
       JMP  TAB02           CONTINUE
TAB00  C    R0,@ROTATE      ROTATED DUMP?
       JNE  TAB01           REGULAR DUMP
       MOVB @ROTTAB,@TAB    SET TAB
       JMP  TAB02           CONTINUE
TAB01  MOVB @REGTAB,@TAB    SET TAB
TAB02  MOV  @FLAG,@FLAG     SEE WHERE WE STARTED
       JEQ  BEGIN           CAME FROM CALL LINK("DUMP")
       CLR  @FLAG           RESET FLAG
       LWPI GPLWS           CAME FROM CALL LINK("SET") OR ("CART")
       B    @>0070          RETURN TO BASIC
       COPY "DSK1.DUMP_MAIN" COPY MAIN BODY OF PROGRAM
* ENTRY POINT FROM LOAD. ASSURES PROGRAM
* WILL SERVICE ANY OTHER INTERRUPT
* ROUTINE ALREADY IN MEMORY.
*
ISR    DATA 0            SPACE TO SAVE OTHER ROUTINES ADDRESS
ISR1   DATA 0            SPACE TO SAVE RUN TIME ISR
```

```
CKISR  MOV  @>83C4,@ISR  SAVE ADDRESS
       LI   R0,DIRENT    THIS PROGRAMS ISR
       MOV  R0,@>83C4    TO VECTOR
       RT                BACK TO BASIC
ENDPMG EQU  $            MARK END OF PROGRAM
       END  CKISR        AUTO EXECUTE
```

Disk 17. Contents of File DUMPDOC

SCREEN DUMP Version 3

INSTRUCTIONS

The disk you received contains two screen dump programs. The file DUMPCB is for use in Console Basic with either the Editor/Assembler module, or Mini-Memory. The file DUMPXB is for use with Extended Basic. The files DUMPCB/TXT and DUMPXB/TXT are the source code for the respective programs, while DUMP_MAIN is a source module that is common to both versions. DUMPDOCBAS is an Extended Basic program that will print these instructions. DUMPDEFALT is an Extended Basic program that will allow you to change the output specification to suit your printer. Features of the screen dump are:

- Simple two keypress activation (**FCTN 0**), or activation with a CALL LINK statement from within your program.
- 6 different types of printouts through a combination of:
  - Standard 'straight up',
  - Rotated 90 deg. clockwise,
  - Double sized,
  - Inverted (white on black).
- Selectable left margin for printout.
- Screen dumps of module screens (requires additional hardware).

LOADING THE PROGRAM

To load the program from disk, you must be operating in either Extended Basic, or Console Basic with either the Mini-Memory or Editor/Assembler modules inserted into the computer. First, issue the statement CALL INIT. This readies the Extended Memory or Mini-Memory to accept the program. (This step may be omitted if a CALL INIT has already been issued since entering Basic.)

Next, load the program by issuing the statement CALL LOAD("DSK1.DUMPCB") if in Console Basic, or CALL LOAD("DSK1.DUMPXB") if in Extended Basic. (This assumes that you are loading the program from drive 1. If not, change the 1 to the correct number of the drive in use.) Please be sure to load the right program. DUMPXB will load with Console Basic, but it will not function correctly. The computer should load the program into memory, and return to the blinking cursor prompt. The program is now ready for use.

Note: The Screen Dump uses the user interrupt vector in order to provide output with a function keypress. If you load another program that uses this vector after loading the Screen Dump, it will probably prevent keyboard activation of the Screen Dump. In order to use both programs, load the other program first, then load the Screen Dump. Screen Dump checks the interrupt vector for the presence of another program, and makes sure that both programs get serviced during the interrupt period.

USING THE PROGRAM FROM BASIC

Anytime you wish to dump the screen graphics to a printer, either while a program is running or while in command mode, simply hold down the **FCTN** key, and press the **0** (zero) key. The program will output the screen to the printer and then return to your program. There are 4 options that you may wish to set for the screen dump program. These are tab spacing, rotate, double size, and inverted print. These options are set by using a CALL LINK("SET") statement either from within a Basic program, or from command mode. First an explanation of the options, then some examples. . .

The first option is the tab spacing, that is, how many blank spaces are left in the left margin of the page. The program will center the dump on the page unless you specify a tab. The program will accept tabs up to 99, however on most standard printers, a tab that large will result in either line wrap-around or a printer error condition. Tabs above 99 or less than 0 will cause a tab setting of zero. The tab can be specified with a number, numeric expression, a simple variable, or an array element. (See the examples section below.)

The next 3 options deal with the way the screen is output to the printer. The default is to print the graphics on paper just as you see them on the screen, that is left to right and top to bottom. By passing an upper-case R to the program in the LINK statement, the program will rotate the screen image 90 degrees clockwise on the paper. If you pass an upper-case D in the LINK statement, the screen will be printed double size. Most standard printers cannot print enough dots per line to handle a double size screen dump without rotating the image. Because of this, when you specify a double size print-out, the program automatically switches to the rotate mode. The last parameter you can specify is an inverse print-out. When you send an upper-case I to the program, the characters will appear as white on a black background. The I, D, and R commands can be passed as separate strings, but must be passed together if you want to specify a tab setting also. They can be passed as a string variable, string array element, or string expression. The examples below should illustrate the ways these parameters can be passed to the program.

```
CALL LINK("SET",5)
```
causes a standard dump with a tab setting of 5.

```
CALL LINK("SET","RI")
```
results in a rotated, inverted dump at the default tab setting.

```
CALL LINK("SET",A,"D")
```
provides a double size print-out with a tab setting equal to the variable A.

```
CALL LINK("SET",A$,R(X))
```
if A$ = ID and R(X) = 0, the print-out will be inverted, double size, and printed at the left margin.

One note about the way the program obtains these parameters. Only the first two parameters are checked. These parameters are analyzed for numeric or character format, and then the necessary program flags are set. The second parameter takes precedence over the first if they are both of the numeric type. Here are some examples:

```
CALL LINK("SET",5,A)
```
causes a regular screen dump with a tab setting equal to the value of variable A.

```
CALL LINK("SET","I","D",8)
```
ends up printing an inverted, double size screen at the default tab setting. The 8 is ignored by the program.

```
CALL LINK("SET",5,"I","R","D")
```
here the result is an inverted dump at a tab of 5. The program never sees the "R" and "D".

Once you do the CALL LINK("SET"), all screen dumps initiated with **FCTN 0** will be done with the same options. To change options you must do another CALL LINK("SET"). Keep in mind that any option not specified in the CALL LINK ("SET") command is reset to the default for that option.

The screen dump can also be initiated from within a basic program by including a CALL LINK("DUMP",<option list>) in your program at the point where you want the screen contents dumped to the printer. <option list> refers to the options described in the examples section above. Omitting the options list will result in a screen dump with the default settings of a normal dump centered on the page. If you dump a screen with CALL LINK("DUMP"), then dump another screen by pressing **FCTN 0**, the second dump will be done using the options set in the CALL LINK("DUMP") statement.

If you wish to exit the program before the screen dump is completed, press **FCTN 9**. The program only checks for this key at the beginning of each printed line, so you may have to hold it down for a moment. Pressing **FCTN 4** will also halt the program, but it results in an I/O error and leaves the printer in the graphics mode. If you halt the program with **FCTN 4**, you will most likely have to turn the printer off and back on in order for it to function properly.

USING THE PROGRAM WITH A "LOAD INTERRUPT" SWITCH

A load interrupt switch is a hardware modification that can be made to the TI-99/4A computer. This modification has been detailed in the November 1983 issue of *Enthusiast '99* magazine and in many users group newsletters. When this switch is pressed, the computer branches to an assembly language program pointed to by a vector at memory location >FFFE. The Screen Dump program sets up this location to point to itself. To use the program in conjunction with the switch, first load the program as usual, then press the switch whenever you want to force the computer to execute the screen dump program. There is a slight delay before the program starts to output. The setup of options for a screen dump that is to be initiated with the load interrupt switch is exactly the same as for the **FCTN 0** output. To set up the program for the options you desire, do a CALL LINK("SET",<option list>"). Again, <option list> refers to the options listed in the above section, i.e. CALL LINK("SET","DI",5).

A few notes on using this program with the load interrupt switch:

The program attempts, in software, to "debounce" the switch. This works most of the time, but occasionally the program will not return to Basic properly. For this reason, I suggest that you not use this feature while running a program that contains critical data without first saving the data. Also, I don't think it would be a good idea to use the switch while access is being made to an output device, especially a disk drive! I haven't tried it myself, but I think that would be a good way to screw up a good disk. Sometimes, when the switch has been pressed while the computer is in the command mode, when the program finishes the screen dump the computer appears to lock up. The cursor will not be blinking, or may not appear at all. If this happens, PRESS THE ENTER KEY! I don't know why this happens, but pressing **ENTER** is the only way I have been able to regain control of the computer. Everything seems to work OK after doing this. One more thing — I have had the best results with the load interrupt switch by pressing the switch firmly and then releasing, rather than jabbing it. REMEMBER — There is a delay between the time you press the switch and the beginning of output to the printer! Don't get impatient and press the switch again. This will only result in a longer delay.

SCREEN DUMPS OF CARTRIDGE PROGRAMS

To dump screens from solid state modules, you must have a load interrupt switch installed on your computer. Secondly, the module's program must use the regular graphics mode of the computer, and its' screen image table must begin at location >0800. Most TI modules use this format. (Don't try to dump a screen from Parsec. It uses bit image graphics mode.) Don't be disappointed if the screen dump doesn't turn out looking like the screen. A lot of module programs use sprites (the Screen Dump program does NOT print sprites), and a character that looks like a space on the screen may indeed have a pattern with the same color for foreground and background. (The Screen Dump program does not pay any attention to character colors.)

To dump cartridge screens, first load the program from Basic in the usual manner. You may use either Extended Basic or Console Basic, but I have had better results with Console Basic and the Editor/Assembler module. (You cannot dump cartridge screens if the program was loaded in Console Basic running with the Mini-Memory module!) After you have loaded the program do a CALL LINK("CART",<option list>) where <option list> is the options listed in the basic section above. This allows you to set the options, and also sets up the program for the screen format used by cartridge programs. Even if you want to use the default options, you *must* do a CALL LINK("CART") in order for the program to function correctly. After you have done this, exit Basic and insert the cartridge that you want to dump. When the desired screen appears, press the load interrupt switch, and after a short delay the program will output the screen to the printer. In most cases, the program will return to the cartridge program when it finishes. However, I recommend that you pay heed to the cautions mentioned in the last section whenever you use this feature. If you wish to change the options before you dump another cartridge screen, return to Basic and do a CALL LINK("CART",<option list>) again. If you wish to return to Basic and then use the program to dump a Basic screen, you must first do a CALL LINK("SET"). This will reset the program from the cartridge mode to the Basic mode.

NOTES FOR PROGRAM MODIFICATION

The Screen Dump programs are set up to print to an Epson/Gemini type printer. They have been fully tested on an Epson MX80, a Gemini 10X and SG10, a Panasonic KX-P1091, and a TI Impact printer. Other printers that use the same control codes and the same bit image graphics scheme as these printers will work. To modify the programs for use with other type printers will require a good understanding of the printers graphics modes, and a fair amount of assembly language programming experience. The source code sent with the programs is fully commented and the sections that must be modified for other type printers are clearly marked. The programs are set up to print to a parallel printer through an interface using the PIO file specification. If your printer uses another specification, (RS232 etc.) run the program DUMPDEFALT in Extended Basic and follow the instructions given on the screen. You will only need to run this program once, as it changes the object code in the screen dump files on your disk. You are limited to 28 characters in your printer specification. If you are using a printer connected to the RS232 card you *must* use the .CR option. If the default tabs do not center the print on your printer, you can change the tab defaults, then re-assemble the program. The tab settings for regular, rotated, and double size prints are stored at labels REGTAB, ROTTAB, and DUBTAB.

PERSONAL NOTES

I hope you enjoy using these programs. PLEASE — pass a copy on to your friends. They are public domain programs, and everyone that wants them should have them. My only request is that when you pass along the programs, please make sure you give the source code and this documentation file. Although the one you are giving the program to may not be interested in the source code, someone who gets it from him may be. I would like to thank D.C. Warren of Pomona, CA for his modifications to my original program to allow **FCTN 0** output. Please feel free to modify the programs to suit yourself. If you make a modification to the program that you feel enhances its' operation, please send me the details so that I can make the modifications known. The same goes for any hidden bugs you might find. You can reach me at the following address:

Danny Michael
Rt. 9, Box 460
Florence, AL 35630

Phone (205)764-7881

or you can reach me on CompuServe, ID# 75116,1225 . . . EMAIL only, please.

Disk 17. Contents of File DUMPXB/TXT

```
       TITL 'SCREEN DUMP FOR EPSON & GEMINI PRINTERS'
*            EXTENDED BASIC - VERSION 3
************************************
* THIS PROGRAM WRITTEN DEC. 1983   *
* MODIFIED DEC. 1984               *
* BY : DANNY MICHAEL               *
*      RT. 9, BOX 460              *
*      FLORENCE, AL.               *
*              35630               *
*                                  *
*    TELEPHONE:  (205) 764-7881    *
*                                  *
* COMPUSERVE ID#: 75116,1225       *
*                                  *
* PROGRAM RELEASED INTO THE PUBLIC *
* DOMAIN DEC. 1984.                *
*                                  *
* THIS PROGRAM NOT TO BE SOLD!!    *
*                                  *
* ANYONE WHO WISHES MAY HAVE IT    *
* FREE.                            *
*                                  *
* MODIFIED MAY 1985 BY D.C. WARREN *
* POMONA, CA FOR FCTN 0 OUTPUT     *
************************************
       DEF  DUMP,SET,CART   DEFINE ENTRY POINT
PABLOC EQU  >1000           PAB LOCATION IN VDP RAM
COUNT  EQU  PABLOC+5        POINTER TO BYTE COUNT BYTE
BUFFER EQU  PABLOC+72       READ/WRITE BUFFER IN VDP RAM
STATUS EQU  >837C           STATUS REGISTER
PNTR   EQU  >8356           POINTER BYTE FOR DSRLNK
GRMRA  EQU  >9802           GROM READ ADDRESS
GRMWA  EQU  >9C02           GROM WRITE ADDRESS
VMBR   EQU  >202C           VIDEO MULII BYTE READ
VMBW   EQU  >2024           VIDEO MULTI BYTE WRITE
VSBR   EQU  >2028           VIDEO SINGLE BYTE READ
VSBW   EQU  >2020           VIDEO SINGLE BYTE WRITE
KSCAN  EQU  >201C           KEY SCAN UTILITY
GPLWS  EQU  >83E0           GPL WORKSPACE REGISTER POINTER
FRSTHI EQU  >2002           POINTER TO FIRST FREE ADDRESS
       DATA 0,0             BUMP OBJECT CODE BY 2 WORDS
PTABLE DATA >0400           BEGINNING ADDRESS OF PATTERN TABLE
DSRLNK DATA REG2,DSRL1      DATA FOR BLWP @DSRLNK
F      DATA 15              MASK BITS
SXTEEN DATA 16              DITTO
INVERT DATA 0               FLAG FOR INVERTED PRINT
DOUBLE DATA 0               FLAG FOR DOUBLE SIZE PRINT
ROTATE DATA 0               FLAG FOR ROTATED PRINT
```

```
FLAG   DATA 0              FLAG WORD FOR VARIOUS ROUTINES
RESET  DATA >0D0A,>1B32    RESET DATA FOR PRINTER
REG1   BSS  >20            SPACE FOR REGISTER SET 1
REG2   BSS  >20              "    "      "     "  2
GPLBUF BSS  >B6            SPACE TO SAVE FAC THRU GPL REGISTERS
PABBUF BSS  212            SPACE TO SAVE VDP RAM USED IN PMG
GROMSV BSS  2              SPACE TO SAVE GROM ADDRESS
DSRBUF BSS  8              BUFFER FOR DSRLNK
BUF1   BSS  8              BUFFER FOR CHAR PATTERNS
BUF2   BSS  16             BUFFER FOR PRINT CODES
SVPNTR BSS  2              BUFFER USED IN DSRLNK CALLS
SPNTR  BSS  2              SCREEN LOCATION POINTER
SPNTR1 BSS  2              FOR ROTATE ROUTINE
BPNTR  BSS  2              BUFFER LOCATION POINTER
*
* DATA FOR PERIPHERAL ACCESS BLOCK
*
PDATA  DATA >0012,BUFFER,>FF00,0,>0020
****************************************
* TO CHANGE OUTPUT DEVICE, TYPE IN YOUR   *
* PRINTER DEVICE NAME BETWEEN THE ' MARKS *
* IN THE NEXT LINE, THEN SAVE THIS FILE   *
* AND RE-ASSEMBLE. BE SURE THAT THERE ARE *
* 32 POSITIONS BETWEEN THE ' MARKS. PAD   *
* YOUR PRINTER NAME WITH BLANKS ON THE    *
* RIGHT SIDE TO MAKE 32 POSITIONS.        *
****************************************
       TEXT 'PIO.CR                          '
       EVEN              DO NOT DELETE THIS STATEMENT!!!!
*
* DATA TO SET PRINTER UP FOR GRAPHICS
*
SETUP  DATA >0D0A,>1B41,>081B,>4F00
*
* BEGIN NEW LINE DATA
*
CRLF   DATA >0D0A,>1B4B
GRPNUM DATA 1              THE NUMBER OF DOTS PER LINE
*                          WILL BE PLACED HERE
ZERO   DATA 0,0,0,0        DATA FOR BLANK CHARACTER
*
* BYTE VALUES FOR VARIOUS ROUTINES
*
MASK   BYTE >20            MASK TO SET/RESET EQUAL BIT
MASK1  BYTE >80            MASK TO STRIP SCREEN OFFSET
PERIOD BYTE >2E            ASCII FOR PERIOD
VALID  BYTE >AA            DSR ROM VALIDATION BYTE
OFFSET BYTE >60            SCREEN OFFSET
OPEN   BYTE 0              FILE OPEN BYTE
CLOSE  BYTE 1              FILE CLOSE BYTE
WRITE  BYTE 3              FILE WRITE BYTE
```

```
LOW    BYTE >80             LOWEST PRINTABLE CHARACTER
HIGH   BYTE >EF             HIGHEST PRINTABLE CHAR
TAB    BYTE 19              TAB SPACES FOR PRINTING
REGTAB BYTE 19              LEFT TAB FOR REGULAR DUMP
ROTTAB BYTE 25                "   "   "  ROTATED   "
DUBTAB BYTE 9                 "   "   "  DOUBLE SIZE DUMP
*
MESG2  BYTE 10              DATA FOR ERROR MESSAGE
       TEXT 'I/O ERROR!'
       EVEN
************************
* SUBROUTINE DSRLNK     *
*                       *
* ENTER WITH POINTER TO *
* NAME LENGTH IN >8356  *
************************
DSRL1  MOV  *R14+,R5     GET DATA TO R5 & INC PMG CNTR
       SZCB @MASK,R15    RESET EQUAL BIT
       MOV  @>8356,R0    NAME LENGTH BYTE TO R0
       MOV  R0,R9        THEN TO R9
       AI   R9,>FFF8     R9 POINTS TO BEGINNING OF PAB
       BLWP @VSBR        GET LENGTH BYTE INTO R1
       MOVB R1,R3        MOVE TO R3
       SRL  R3,8         ALIGN TO RIGHT BYTE
       SETO R4           R4 = >FFFF
       LI   R2,DSRBUF    R2 POINTS TO BUFFER
DSRL2  INC  R0           R0=R0+1
       INC  R4           R4=R4+1
       C    R4,R3        DONE YET?
       JEQ  DSRL3        YES, GO ON
       BLWP @VSBR        NO, GET NEXT BYTE
       MOVB R1,*R2+      AND PUT IT IN BUFFER
       CB   R1,@PERIOD   WAS IT A PERIOD?
       JNE  DSRL2        NO, GET NEXT BYTE
DSRL3  MOV  R4,R4        R4=0?
       JEQ  DSRL10       YES, RETURN W/ERROR
       CI   R4,>0007     R4 GREATER THAN 7?
       JGT  DSRL10       YES, RETURN W/ERROR
       CLR  @>83D0       >83D0=0
       MOV  R4,@>8354    NAME LENGTH TO >8354
       INC  R4           R4=R4+1
       A    R4,@>8356    >8356 POINTS TO DATA AFTER NAME
       LWPI >83E0        USE GPL REGS
       CLR  R1           R1=0
       LI   R12,>0F00    R12 = BEGINNING OF DSR BLOCKS
DSRL4  MOV  R12,R12      CHECK FOR 0
       JEQ  DSRL04       YES, GO ON
       SBZ  >00          TURN CARD OFF
DSRL04 AI   R12,>0100    POINT TO NEXT DSR BLOCK
       CLR  @>83D0       >83D0=0
```

```
       CI   R12,>2000    DONE?
       JEQ  DSRL9        YES, INCORRECT DEVICE NAME
       MOV  R12,@>83D0   STORE BLOCK AT >83D0
       SBO  >00          TURN CARD ON
       LI   R2,>4000     R2 POINTS TO BEGINNING OF DSR ROM
       CB   *R2,@VALID   CHECK FOR VALID CARD
       JNE  DSRL4        NOT ONE HERE, TRY NEXT BLOCK
       AI   R2,8         R2 POINTS TO DSR ENTRY FIELD
       JMP  DSRL6        SKIP OVER THE FIRST TIME THROUGH
DSRL5  MOV  @>83D2,R2    NEXT DEVICE FIELD TO R2
       SBO  >00          TURN CARD ON
DSRL6  MOV  *R2,R2       ENTRY FIELD POINTER TO R2
       JEQ  DSRL4        TRY NEXT BLOCK IF THIS IS LAST ONE
       MOV  R2,@>83D2    SAVE POINTER TO NEXT ENTRY
       INCT R2           POINT TO ENTRY ADR FOR THIS DEVICE
       MOV  *R2+,R9      MOVE TO R9
       MOVB @>8355,R5    LENGTH BYTE TO R5
       JEQ  DSRL8        GO IF ZERO
       CB   R5,*R2+      LENGTH THE SAME?
       JNE  DSRL5        NO, TRY NEXT DEVICE
       SRL  R5,8         YES, ALIGN COUNT TO RIGHT BYTE
       LI   R6,DSRBUF    R6 POINTS TO NAME BUFFER
DSRL7  CB   *R6+,*R2+    CHECK FOR MATCH & INC POINTERS
       JNE  DSRL5        NO MATCH, TRY NEXT DEVICE
       DEC  R5           COUNT -1
       JNE  DSRL7        TRY NEXT BYTE
DSRL8  INC  R1           R1=R1+1
       BL   *R9          BRANCH TO DSR ROM
       JMP  DSRL5        THIS INSTRUCTION IS SKIPPED!
       SBZ  >00          TURN CARD OFF
       LWPI REG2         USE PROGRAM REGISTERS
       MOV  R9,R0        R0 POINTS TO FLAG/STATUS BYTE
       BLWP @VSBR        GET STATUS
       SRL  R1,13        ALIGN ERROR BITS
       JNE  DSRL11       GO IF ERROR
       RTWP              RETURN NO ERROR
DSRL9  LWPI REG2         USE PROGRAM REGISTERS
DSRL10 CLR  R1           R1=0
DSRL11 SWPB R1           ERROR CODE TO RIGHT BYTE
       MOVB R1,*R13      PUT IN R0 OF CALLING PMG
       SOCB @MASK,R15    SET ERROR BIT
       RTWP              RETURN W/ERROR
*
*BEGINNING OF PROGRAM
*
SET    LWPI REG1            LOAD WS REG
       SETO @FLAG           SET FLAG
       LI   R0,>80EF        FOR LOW & HIGH
       MOVB R0,@LOW         LOWEST CHAR
       SWPB R0              NEXT BYTE
       MOVB R0,@HIGH        HIGHEST BYTE
```

```
       LI   R0,>6080        SCREEN OFFSET
       MOVB R0,@OFFSET      SET OFFSET
       SWPB R0              NEXT BYTE
       MOVB R0,@MASK1       SET MASK FOR OFFSET CHARS
       LI   R0,>0400        PATTERN DESCRIPTOR TABLE
       MOV  R0,@PTABLE      IN PLACE
       JMP  DUMP01          CONTINUE
CART   LWPI REG1            LOAD WS REG
       SETO @FLAG           SET FLAG
       SETO R0              R0=>FFFF
       MOVB R0,@HIGH        SET HIGH BYTE
       INC  R0              R0=0
       MOVB R0,@LOW         SET LOW BYTE
       MOVB R0,@OFFSET      CLEAR OFFSET
       MOVB R0,@MASK1       CLEAR MASK FOR CHARS ABOVE >80
       LI   R0,>0800        PATTERN DESCRIPTOR TABLE
       MOV  R0,@PTABLE      IN PLACE
       JMP  DUMP01          CONTINUE
DUMP   LWPI REG1             LOAD WS REG
       CLR  @FLAG            CLEAR FLAG
DUMP01 SETO R0                RESET
       MOVB R0,@TAB           TAB
       INCT R0
       MOV  R0,@GRPNUM       GRAPHICS COUNT FOR NORMAL DUMP
       CLR  @INVERT          RESET
       CLR  @DOUBLE          PROGRAM
       CLR  @ROTATE          FLAGS
****************************
* GET PARAMETERS FROM LINK *
****************************
       MOVB @>8312,R3       # OF ARG. TO R3
       SRL  R3,8            ALIGN TO RIGHT
       MOV  R3,R3           TEST FOR ZERO
       JEQ  TABSET          NO ARGUMENTS
       LI   R7,>8300        ARGUMENT LIST
ARG01  LI   R6,16           OFFSET
       CI   R3,3            MAKE SURE ONLY
       JL   ARG02               TWO ARGUMENTS
       LI   R3,2
ARG02  MOV  @>8310,R0       STACK POINTER TO R0
       A    R6,R0           ADJUST
       LI   R1,DSRBUF       BUFFER POINTER TO R1
       LI   R2,8            BYTE COUNT
       BLWP @VMBR           READ STACK ENTRY
       MOVB *R7,R0          IDENT BYTE TO RO
       SRL  R0,8            ALIGN
       JEQ  NUMEXP          NUMERIC EXPRESSION
       AI   R1,4            POINT TO VALUE POINTER
       CI   R0,2            NUMERIC VARIABLE?
       JEQ  NUMVAR          YES
```

```
       CI   R0,4            CHECK FOR >4
       JLT  STRING          MUST BE STRING
       JMP  ARG09           BAD ARGUMENT
NUMEXP MOVB @1(R1),R4       FIRST WORD TO R4
       SWPB R4
       MOVB *R1,R4
       MOV  R4,R4           CHECK FOR ZERO
       JNE  ARG03           N0
       MOVB @ZERO,@TAB      SET TAB TO ZERO
       JMP  ARG09           THROUGH WITH THIS ONE
ARG03  SRL  R4,8            GET EXP
       CI   R4,>0040        EXP ONE?
       JNE  ARG09           NO, FORGET THIS ONE
       INC  R1              POINT TO NUMBER
       MOVB *R1,@TAB        MOVE TO TAB POINTER
       JMP  ARG09           GET NEXT ONE
NUMVAR MOV  *R1,R1          VALUE POINTER TO R1
       JMP  NUMEXP          GET VALUE
STRING MOV  *R1+,R0         STRING POINTER TO R0
       MOV  *R1,R2          BYTE COUNT TO R2
       CI   R2,9            CHECK FOR GREATER THAN 8
       JL   ARG04           IT'S OK
       LI   R2,8            MAKE IT OK
ARG04  LI   R1,DSRBUF       R1 POINTS TO BUFFER
       BLWP @VMBR           READ STRING TO BUFFER
ARG08  MOVB *R1,R9          BYTE TO R9
       SRL  R9,8            ALIGN
       CI   R9,73           IS IT "I"?
       JNE  ARG05           NO
       SETO @INVERT         YES, SET FLAG
       JMP  ARG07           GO ON
ARG05  CI   R9,82           IS IT "R"?
       JNE  ARG06           NO
       MOV  @ROTATE,@ROTATE ALREADY SET?
       JNE  ARG07           YES, SKIP IT
       SETO @ROTATE         YES, SET FLAG
       LI   R15,>C000       GRAPHICS COUNT
       MOV  R15,@GRPNUM     MOVE INTO POSITION
       JMP  ARG07           GO ON
ARG06  CI   R9,68           IS IT "D"?
       JNE  ARG07           NO
       SETO @DOUBLE         YES, SET FLAG
       SETO @ROTATE         ASSURE ROTATED IF DOUBLE
       LI   R15,>8001       GRAPHICS COUNT FOR DOUBLE
       MOV  R15,@GRPNUM     INTO POSITION
ARG07  INC  R1              INC POINTER
       DEC  R2              COUNT -1
       JNE  ARG08           DO AGAIN IF NOT FINISHED
ARG09  DEC  R3              DEC ARG. COUNTER
       JEQ  TABSET          THAT'S ALL
       AI   R6,8            POINT TO NEXT ENTRY
```

```
       INC  R7              IDENT POINTER +1
       JMP  ARG02           DO IT AGAIN
******************
* SET TAB SPACING *
******************
TABSET SETO R0              R0=>FFFF
       CB   @TAB,R0         HAS TAB BEEN SET?
       JNE  TAB02           YES, GO ON
       C    R0,@DOUBLE      DOUBLE FLAG SET?
       JNE  TAB00           NO, TRY NEXT ONE
       MOVB @DUBTAB,@TAB    SET TAB
       JMP  TAB02           CONTINUE
TAB00  C    R0,@ROTATE      ROTATED DUMP?
       JNE  TAB01           REGULAR DUMP
       MOVB @ROTTAB,@TAB    SET TAB
       JMP  TAB02           CONTINUE
TAB01  MOVB @REGTAB,@TAB    SET TAB
TAB02  MOV  @FLAG,@FLAG     SEE WHERE WE STARTED
       JEQ  BEGIN           CAME FROM CALL LINK("DUMP")
       CLR  @FLAG           RESET FLAG
       B    @EXIT01         CAME FROM CALL LINK("SET") OR ("CART")
       COPY "DSK1.DUMP_MAIN"  GET MAIN PROGRAM BODY
* ENTRY POINT FROM LOAD. ASSURES PROGRAM
* WILL SERVICE ANY OTHER INTERRUPT
* ROUTINE ALREADY IN MEMORY.
*
ISR    DATA 0            SPACE TO SAVE OTHER ROUTINES ADDRESS
ISR1   DATA 0            SPACE TO SAVE RUN TIME ISR
CKISR  MOV  @>83C4,@ISR  SAVE ADDRESS
       LI   R0,DELAY     DELAY FOR ONE SECOND
       MOV  R0,@>83C4    TO VECTOR
       B    @>0070       BACK TO BASIC
CNTR   DATA 60           60 COUNTS=1 SEC
DELAY  DEC  @CNTR        SUB ONE
       JNE  DELAY1       NOT DONE
       LI   R0,DIRENT    ISR ROUTINE
       MOV  R0,@>83C4    TO VECTOR
DELAY1 RT                RETURN
ENDPMG EQU  $            LABEL FOR PMG END ADDRESS
       AORG GPLWS+22     GPL REG 11
       DATA CKISR        MAKE THIS SUCKER AUTO EXECUTE!
       END
```

Disk 17. Contents of File DUMP_MAIN

```
* SCREEN DUMP VERSION 3 MAIN CODE
* A FREE, PUBLIC DOMAIN PROGRAM
* BY DANNY MICHAEL, FLORENCE, AL
*******************************
* SCREEN DUMP CODE BEGINS HERE *
*******************************
****************************************
* THIS SECTION OF CODE IS FOR FCTN 0     *
* OUTPUT, AND WAS WRITTEN BY D.C. WARREN *
* OF POMONA, CA. MY THANKS TO HIM FOR    *
* THIS MOST USEFUL MODIFICATION!         *
****************************************
DIRENT LWPI REG1         LOAD REGISTERS
       MOV  @FRSTHI,R0   GET FIRST AVAIL ADDRESS
       CI   R0,ENDPMG    COMPARE WITH END OF PROGRAM
       JL   QUIT         CALL INIT HAS BEEN DONE
       LI   R12,>24      LOAD 9901 KBD COL BASE
       CLR  R0           R0=0
       LDCR R0,3         SET KBD TO SCAN COL 0
       LI   R12,>6       LOAD 9901 KBD ROW BASE
       STCR R0,8         READ KBD ROW
       CI   R0,>EF00     IS FCTN KEY PRESSED?
       JNE  DIREN1       NO!
       LI   R12,>24      LOAD 9901 KBD COL BASE
       LI   R0,>0500
       LDCR R0,3         SET KBD TO SCAN COL
       LI   R12,>6       LOAD 9901 KBD ROW BASE
       STCR R0,8         READ KBD ROW
       CI   R0,>F700     IS ZERO KEY PRESSED?
       JEQ  BEGIN        YES!
DIREN1 LWPI GPLWS        GPL REGISTERS
       MOV  @ISR,R0      NO MATCH, CHECK FOR ANOTHER ISR
       JEQ  DIREN3       NO OTHER ISR
       B    *R0          GO TO OTHER ROUTINE
DIREN3 RT                RETURN TO BASIC
QUIT   MOV  @ISR,@>83C4  RESTORE ADDRESS
       LWPI GPLWS        LOAD GPL REGISTERS
       RT                AND RETURN
*************************
* INITIALIZATION ROUTINE *
*************************
BEGIN  MOV  @>83C4,@ISR1 SAVE VECTOR
       CLR  @>83C4       CLEAR ISR HOOK
       LWPI REG1         SET REGISTER POINTER
       LI   R1,>834A     R1 POINTS TO FAC
       LI   R2,GPLBUF    R2 POINTS TO BUFFER
       LI   R3,>5B       R3 = WORD COUNT
MOVE   MOV  *R1+,*R2+    MOVE WORD & INC POINTERS
```

```
       DEC  R3           COUNT -1
       JNE  MOVE         LOOP IF NOT DONE
       LI   R0,PABLOC    R0 POINTS TO PAB SPACE IN VDP RAM
       LI   R1,PABBUF    R1 POINTS TO RAM BUFFER
       LI   R2,212       R2 = BYTE COUNT
       BLWP @VMBR        MOVE TO BUFFER
       MOVB @GRMRA,@GROMSV    MOVE GROM
       NOP                    ADDRESS TO
       MOVB @GRMRA,@GROMSV+1  GROMSV
       DEC  @GROMSV         CORRECT ADDRESS
************************
* SET UP PAB IN VDP RAM *
************************
MAKPAB LI   R0,PABLOC    R0 POINTS TO PAB ADDRESS
       LI   R1,PDATA     R1 POINTS TO PAB DATA
       LI   R2,42        R2 IS BYTE COUNT. CHANGE IF YOU CHANGE DEVICE NAME.
       BLWP @VMBW        WRITE TO VDP
*
* OPEN FILE
*
       MOVB @OPEN,R1     OPEN BYTE TO R1
       LI   R0,PABLOC    R0 POINTS TO PAB
       BLWP @VSBW        WRITE OPCODE
       LI   R8,PABLOC+9  R8 POINTS TO LENGTH BYTE
       MOV  R8,@PNTR     PUT IT IN POSITION
       MOV  R8,@SVPNTR   AND SAVE IT
       BLWP @DSRLNK      SEND TO PERIPHERAL CARD
       DATA 8
       JNE  START        GO IF NO ERROR
       B    @ERROR1      GO IF ERROR
*
* SETUP DATA TO BUFFER
*
START  LI   R0,BUFFER    POINT TO PRINT BUFFER
       LI   R1,SETUP     SETUP CODE
       LI   R2,7         BYTE COUNT
       BLWP @VMBW        WRITE TO BUFFER
       A    R2,R0        BUFFER END TO R0
       MOV  R0,@BPNTR    AND TO POINTER
       BL   @NEW2        NEW LINE DATA TO BUFFER
*
* BEGIN ACTUAL DUMP CODE
*
       SETO R0
       C    @ROTATE,R0
       JNE  LOOP0
       B    @TURN
******************************
* THIS SECTION OF CODE IS FOR *
* A REGULAR SCREEN DUMP.      *
```

```
*****************************
LOOP0  SETO @SPNTR       SPNTR = >FFFF
LOOP4  MOV  @SPNTR,R0    POINTER TO R0
       INC  R0           POINTER +1
       CI   R0,>0300     ENOUGH?
       JEQ  DID          YES
       BL   @CHRPAT      GET PATTERN
*****************************************************
* AT THIS POINT, THE PATTERN FOR THE CHARACTER TO BE  *
* PRINTED IS IN BUF1. THE CODE BETWEEN LOOP5 AND DONE *
* CONVERTS THIS INTO GRAPHICS CODES FOR EPSON/GEMINI  *
* TYPE PRINTERS. (THIS CODE IS PUT INTO BUF2.)        *
* FOR OTHER TYPES OF PRINTERS, THE ROUTINE HERE WILL  *
* NEED TO BE MODIFIED.                                *
*****************************************************
LOOP5  LI   R3,>0080     MASK FOR LEFT BIT
       LI   R5,BUF2      BUFFER FOR CONVERTED CHAR
LOOP3  MOV  R3,R3        R3=0?
       JEQ  DONE         YES
       LI   R4,8         R4=8
       LI   R6,>8000     MASK
       LI   R1,BUF1      POINT TO PATTERN BUFFER
       CLR  R2           R2=0
LOOP2  MOVB *R1+,R8      SCREEN BYTE TO R8
       SRL  R8,8         ALIGN TO RIGHT BYTE
       COC  R3,R8        TEST BIT
       JNE LOOP1         NO MATCH
       SOCB R6,R2        MATCH, ADD COUNT
LOOP1  SRL  R6,1         MOVE MASK BIT TO RIGHT
       DEC  R4           COUNT -1
       JNE  LOOP2        DO AGAIN
       MOV  @INVERT,@INVERT CHECK INVERT FLAG
       JEQ  NOTINV          NO, GO ON
       INV  R2              YES, INVERT
NOTINV MOVB R2,*R5+      MOVE TO BUF2
       SRL  R3,1         MOVE MASK BIT
       JMP  LOOP3        AND DO AGAIN
DONE   LI   R2,8         FOR MOVBUF ROUTINE
       BL   @MOVBUF      MOVE TO PRINT BUFFER
       MOV  @SPNTR,R9    SCREEN LOCATION TO R9
       COC  @F,R9        MASK FOR MID OR END LINE
       JNE  LOOP4        NO, GET NEXT CHAR
       BL   @PRINT       YES, SEND TO PRINTER
       LI   R0,BUFFER    VDP PRINT BUFFER
       MOV  R0,@BPNTR    SAVE IT
       MOV  @SPNTR,R9    SCREEN LOCATION TO R9
       COC  @SXTEEN,R9   MASK FOR END OF LINE
       JNE  LOOP4        NO
       BL   @NEWLIN      SEND NEW LINE DATA
       JMP  LOOP4        DO AGAIN
DID    LI   R0,BUFFER    VDP PRINT BUFFER
```

```
       LI   R1,RESET     RESET DATA
       LI   R2,4         BYTE COUNT
       BLWP @VMBW        WRITE IT
       A    R2,R0        BUMP POINTER
       MOV  R0,@BPNTR    SAVE IT
       BL   @PRINT       SEND TO PRINTER
       B    @EXIT        RETURN
*******************************
* PRINT SUBROUTINE              *
*   SENDS PAB BUFFER TO PRINTER *
*******************************
PRINT  LI   R0,PABLOC    BEGINNING OF PAB
       MOVB @WRITE,R1    WRITE OPCODE TO R1
       BLWP @VSBW        ON TO PAB
       LI   R0,BUFFER    POINT TO VDP PRINT BUFFER
       S    R0,@BPNTR    GET BYTE COUNT
       MOVB @BPNTR+1,R1   INTO R1
       LI   R0,COUNT     POINT TO COUNT BYTE IN PAB
       BLWP @VSBW        WRITE IT
       MOV  @SVPNTR,@PNTR RESTORE POINTER TO NAME END
       BLWP @DSRLNK      PRINT IT
       DATA 8
       JEQ  ERROR1       GO ON ERROR
       B    *R11         RETURN
******************
* ERROR ROUTINE  *
******************
ERROR1 LI   R3,MESG2        POINT TO MESSAGE
       LI   R0,738          LAST LINE ON SCREEN
       MOVB *R3+,R2         BYTE COUNT TO R2
       SRL  R2,8            ALIGN
ERROR3 MOVB *R3+,R1         MOVE BYTE TO R1
       AB   @OFFSET,R1      ADD OFFSET
       BLWP @VSBW           WRITE IT!
       DEC  R2              COUNT -1
       JEQ  EXIT            DONE
       INC  R0              NEXT SCREEN LOCATION
       JMP  ERROR3          GET NEXT CHARACTER
****************************************
* EXIT ROUTINE, RESTORES ALL USED MEMORY *
*               AND CLOSES FILE          *
****************************************
EXIT   LI   R0,PABLOC       R0 POINTS TO OPCODE BYTE
       MOVB @CLOSE,R1       CLOSE OPCODE
       BLWP @VSBW           WRITE IT
       MOV  @SVPNTR,@PNTR   RESTORE LENGTH POINTER
       BLWP @DSRLNK         CLOSE FILE
       DATA 8
       MOVB @GROMSV,@GRMWA  RESTORE
       NOP
```

```
       MOVB @GROMSV+1,@GRMWA  GROM ADDRESS
       LI   R0,PABLOC       R0 POINTS TO PAB SPACE IN VDP RAM
       LI   R1,PABBUF       R1 POINTS TO RAM BUFFER
       LI   R2,212          R2 = BYTE COUNT
       BLWP @VMBW           MOVE FROM BUFFER
       LI   R1,GPLBUF       R1 POINTS TO BUFFER
       LI   R2,>834A        R2 POINTS TO FAC
       LI   R3,>5B          R3 = WORD COUNT
MOVE1  MOV  *R1+,*R2+       MOVE WORD & INC POINTERS
       DEC  R3              COUNT -1
       JNE  MOVE1           LOOP IF NOT DONE
       MOV  @ISR1,@>83C4    RESTORE VECTOR
EXIT01 CLR  R0              R0=0
       MOVB R0,@STATUS      CLEAR GPL STATUS BYTE
       MOV  @FLAG,@FLAG     FIND OUT WHERE PMG CALLED FROM
       JEQ  EXIT02          CALLED FROM BASIC
       CLR  @FLAG           CLEAR FLAG
       LIMI 2               RESTART INTERRUPTS
       RTWP                 CALLED WITH INTERRUPT SWITCH
EXIT02 LWPI GPLWS           RESTORE REGISTER POINTER
       RT                   RETURN TO BASIC
****************************************
* SUBROUTINE NEWLINE                   *
* SENDS NEW LINE DATA TO BUFFER        *
* ASSUMES BPNTR POINTS TO END OF BUFFER *
****************************************
NEWLIN MOV  @DOUBLE,@DOUBLE  CHECK FOR DOUBLE
       JEQ  NEW02        NO DOUBLE
       MOV  R10,R10      FIRST PASS?
       JEQ  NEW2         YES, CONTINUE
       JMP  NEW03        NO, BETTER CHECK LOCATION
NEW02  MOV  @ROTATE,@ROTATE  CHECK FOR TURN
       JEQ  NEW01
NEW03  MOV  @SPNTR,R0    LOCATION POINTER
       CI   R0,-1        END OF SCREEN?
       JNE  NEW2         NO
       B    *R11         YES, RETURN
NEW01  MOV  @SPNTR,R0    LOCATION POINTER TO R0
       CI   R0,>02FF     END?
       JNE  NEW2         NO
       B    *R11         YES,RETURN
NEW2   CLR  @>8374       SETUP FOR KEYSCAN
       BLWP @KSCAN       CALL KEY
       MOVB @>8375,R0    GET KEY #
       SRL  R0,8         ALIGN
       CI   R0,15        ESCAPE?
       JNE  NEW4         NO
       B    @DID         YES, RETURN
NEW4   MOV  @BPNTR,R0    END OF BUFFER TO R0
       LI   R1,CRLF      POINT TO DATA
       LI   R2,2         BYTE COUNT
```

```
       BLWP @VMBW        WRITE TO BUFFER
       INCT R0           BUMP POINTER
       MOVB @TAB,R2      BYTE COUNT
       SRL  R2,8         ALIGN
       DEC  R2           ADJUST
       CI   R2,1         CHECK FOR
       JLT  NEW3           LESS THAN 1
       LI   R1,>2000     SPACE IN LEFT BYTE
NEW1   BLWP @VSBW        WRITE SPACE TO BUFFER
       INC  R0           BUMP POINTER
       DEC  R2           COUNT -1
       JNE  NEW1         LOOP
NEW3   LI   R1,CRLF+2    POINT TO DATA
       LI   R2,4         BYTE COUNT
       BLWP @VMBW        GRAPHICS DATA TO BUFFER
       A    R2,R0        ADJUST POINTER
       MOV  R0,@BPNTR    SAVE IT
NEW5   MOV  R11,R9       SAVE RETURN ADR
       BL   @PRINT       PRINT BUFFER
       LI   R0,BUFFER    POINT TO BEGIN BUFFER
       MOV  R0,@BPNTR    RESTORE POINTER
       B    *R9          RETURN
*********************************
* CHARPAT ROUTINE -              *
* GETS THE PATTERN FOR THE CHAR- *
* ACTER POINTED TO BY R0, AND    *
* PUTS IT IN BUF1.               *
*********************************
CHRPAT MOV  R0,@SPNTR    SAVE POINTER
       CLR  R1           R1=0
       BLWP @VSBR        GET BYTE FROM SCREEN
       CB   R1,@LOW      LESS THAN SPACE CHAR?
       JL   SPACE        YES
       CB   R1,@HIGH     HIGHER THAN CHAR 143?
       JH   SPACE        YES, PRINT SPACE
       SZCB @MASK1,R1    STRIP OFFSET
       SRL  R1,5         MULTIPLY BY 8
       A    @PTABLE,R1   FIND PROPER TABLE ADDRESS
       MOV  R1,R0        PUT IT IN R0
       LI   R1,BUF1      POINT TO RAM BUFFER
       LI   R2,8         BYTE COUNT
       BLWP @VMBR        GET PATTERN FOR CHARACTER
       B    *R11         RETURN
SPACE  LI   R0,ZERO      POINT TO BUFFER OF ZEROS
       LI   R1,BUF1      R1 POINTS TO BUFFER
       LI   R2,4         WORD COUNT
SPACE1 MOV  *R0+,*R1+    MOVE & INC POINTERS
       DEC  R2           COUNT -1
       JNE  SPACE1       DO AGAIN IF NOT 0
       B    *R11         PRINT IT
```

```
**********************************
* THIS ROUTINE MOVES BUF2 TO VDP   *
* PRINT BUFFER AND ADJUSTS POINTER *
**********************************
MOVBUF MOV  @BPNTR,R0    POINTER TO R0
       LI   R1,BUF2      RAM BUFFER
       BLWP @VMBW        WRITE TO VDP PRINT BUFFER
       A    R2,R0        NEW POINTER
       MOV  R0,@BPNTR    SAVE IT
       B    *R11         RETURN
********************************
* THIS CODE IS FOR A ROTATED DUMP*
********************************
TURN   CLR  R10          FOR DOUBLE ROUTINE
       LI   R8,1          "    "       "
       LI   R0,736       LOWER LEFT CORNER
TURN99 MOV  R0,@SPNTR1   SAVE IT
TURN00 MOV  R0,@SPNTR    AGAIN
       JLT  TURN06       GO IF LESS THAN TOP OF SCREEN
       BL   @CHRPAT      GET PATTERN
*******************************************************
* AT THIS POINT, THE PATTERN FOR THE CHARACTER TO BE  *
* PRINTED IS IN BUF1. THE CODE IN THE NEXT 9 LINES    *
* CONVERTS THIS INTO GRAPHICS CODES FOR EPSON/GEMINI  *
* TYPE PRINTERS. (THIS CODE IS PUT INTO BUF2.)        *
* FOR OTHER TYPES OF PRINTERS, THE ROUTINE HERE WILL  *
* NEED TO BE MODIFIED.                                *
*******************************************************
       LI   R1,BUF1+7    POINT TO END OF BUFFER
       LI   R2,BUF2      CONVERTED BUFFER
TURN01 MOVB *R1,R3       GET BYTE IN R3
       MOV  @INVERT,@INVERT  CHECK FLAG
       JEQ  TURN02       NOT SET
       INV  R3           INVERT PATTERN
TURN02 MOVB R3,*R2+      MOVE
       DEC  R1           DEC POINTER
       CI   R1,BUF1      CHECK FOR END
       JHE  TURN01       LOOP
       LI   R2,8         BYTE COUNT FOR MOVBUF
       MOV  @DOUBLE,@DOUBLE  DOUBLE SIZE?
       JEQ  TURN2A       NO, GO ON
       BL   @MAKDUB      DO A DOUBLE SIZE CHAR
TURN2A BL   @MOVBUF      WRITE TO BUFFER
       MOV  @SPNTR,R0    SCREEN POS TO R0
       CI   R0,415       CHECK FOR MIDLINE
       JGT  TURN03       NOT THERE YET
       CI   R0,384       CHECK FOR LESS THAN MID
       JLT  TURN03       YES, GO ON
TURN06 BL   @NEW5        TIME TO PRINT 1/2 LINE
       MOV  @SPNTR,R0    GET POS INTO R0
       JLT  TURN04       NEW LINE IF LESS THAN 0
```

```
TURN03 AI   R0,-32       BUMP UP ONE LINE
       JMP  TURN00       LOOP
TURN04 BL   @NEWLIN      SEND NEW LINE DATA
       MOV  @SPNTR1,R0   LAST POS ON BOTTOM LINE
       MOV  @DOUBLE,@DOUBLE   CHECK FOR DOUBLE SIZE
       JEQ  TURN08       NO
       MOV  R10,R10      WHICH PASS?
       JEQ  TURN07       FIRST
TURN08 INC  R0           NEXT SPOT
TURN07 XOR  R8,R10       FLIP PASS CHECKER
       CI   R0,767       END OF PAGE?
       JGT  TURN05       YES
       JMP  TURN99       LOOP
TURN05 B    @DID         FINISH UP
****************************************
* THIS ROUTINE MAKES DOUBLE SIZE CHARS *
****************************************
MAKDUB LI   R1,BUF1      DESTINATION
       LI   R2,BUF2      SOURCE
       LI   R3,4         WORD COUNT
MAKD01 MOV  *R2+,*R1+      MOVE BUF2
       DEC  R3             BACK TO
       JNE  MAKD01         BUF1
       LI   R1,BUF1      SOURCE
       LI   R2,BUF2      DESTINATION
       LI   R3,8         BYTE COUNT
MAKD02 LI   R4,>8000     BIT SETTER
       MOV  R4,R5        BIT CHECKER
       LI   R9,4         BIT COUNT
       MOVB *R1+,R6      BYTE TO R6
       CLR  R7           DESTINATION BYTE
       MOV  R10,R10      CHECK FOR FIRST PASS
       JEQ  MAKD03       YEP, FIRST TIME FOR THIS CHAR
       SLA  R6,4         GET DATA FOR SECOND TIME
MAKD03 COC  R5,R6        THIS BIT SET?
       JNE  MAKD04       NO
       SOCB R4,R7        SET THIS BIT
       SRL  R4,1         SHIFT SETTER
       SOCB R4,R7        SET NEXT ONE
       SRL  R4,1         SHIFT AGAIN
       JMP  MAKD05       SKIP A LITTLE
MAKD04 SRL  R4,2         SHIFT SETTER
MAKD05 SRL  R5,1         SHIFT CHECKER
       DEC  R9           COUNT -1
       JNE  MAKD03       BIT LOOP
       MOVB R7,*R2+      MOVE BYTE TO BUFFER
       MOVB R7,*R2+      TWICE
       DEC  R3           BYTE COUNT -1
       JNE  MAKD02       BYTE LOOP
       LI   R2,16        FOR MOVBUF ROUTINE
```

```
       B    *R11          RETURN
****************************************
* THANKS TO C.J. DALY OF NEW CARROLLTON, *
* MD FOR THE FOLLOWING ROUTINE TO        *
* DEBOUNCE THE LOAD INTERRUPT SWITCH     *
****************************************
INTRPT CLR  @>FFFC        DISARM SWITCH
       LIMI 0             DISABLE INTERRUPTS
       LWPI REG1          LOAD WS REG
       CLR  R0            R0=0
INT01  DEC  R0            COUNT-1
       JNE  INT01         LOOP WHILE SWITCH SETTLES
       STWP R0            GET WS LOCATION
       MOV  R0,@>FFFC     REARM SWITCH
**********************
       SETO @FLAG         SET FLAG
       B    @BEGIN        START PROGRAM
       AORG >FFFC         SETUP FOR INT SWITCH
       DATA REG1,INTRPT
       RORG
```

Disk 18. Fast-Term

Version: 1.16 *Author*: Paul Charlton
Requires: XB or EA *Language*: AL *Updated*: 11/19/86

The definitive terminal emulator program. Supports Xmodem, TEII, and ASCII file transfers, print spooling, clock and more. A TI classic.

```
dskdir. v2.0. 12-dec-96
Disk name               = FAST-TERM
Sectors total           = 360
Sectors used            = 324
Sectors available       = 34
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BBS           2 DIS/VAR 80   >022 001
002 >003 CHARA1        9 PROGRAM      >023 008
003 >004 DEFAULT      24 PROGRAM      >02b 023
004 >005 FAST-TE2     25 DIS/FIX 80   >042 024
005 >006 FAST-TERM    70 DIS/FIX 80   >05a 069
006 >007 FAST-XMD     28 DIS/FIX 80   >09f 027
007 >008 LOAD          9 PROGRAM      >0ba 001 >153 007
008 >009 LOAD1         2 PROGRAM      >0bb 001
009 >00a READ*THIS!   84 DIS/VAR 80 Y >0bc 083
010 >00b READ-ME       5 DIS/VAR 80   >10f 004
011 >00c UTIL1        33 PROGRAM      >113 032
012 >00d UTIL2        33 PROGRAM      >133 032
```

Disk 18. Contents of File BBS

X<

Disk 18. Contents of File READ*THIS!

THE FAST TERMINAL PROGRAM

(C) 1985, Paul Charlton
1110 Pinehurst Ct.
Charlottesville, VA 22901

Program can be purchased by sending $10 check or money order to address above.

Persons wanting source code must have at least 720 sectors per disk (tell me what format) or send an extra disk to me.

Charges: $10.00 for program.
$ 3.50 for me to provide disk and mailer
$ 3.50 for updates (just send me a letter saying that you want update)

Instructions for use

1. You must have: Editor/Assembler, Mini-Memory, TI-Writer, or Extended Basic.
2. You must have a Disk Controller card.
3. You must have 32K of expansion memory.
4. To use the printer spooler feature you currently must have a PIO interface manufactured by TI, Axiom (Parallax), CorComp, or Myarc. The spooler has been tested with these devices and might not work with others.

To load from Extended Basic.

1. Turn computer off.
2. Place Extended Basic cartridge into the cartridge slot.
3. Turn computer on and go to Extended Basic.
4. Type: CALL INIT, then press **ENTER**.
5. Type: CALL LOAD("DSK1.FAST-LOAD")
6. Then: CALL LINK("TERM"). You may even do this from a program you have written (PHONE-DIALER, etc.)

To load with Editor/Assembler or TI-Writer menu.

1. Turn computer off.
2. Place Editor/Assembler or TI-Writer cartridge in the cartridge slot.
3. Turn computer on and go to the cartridge main menu.
4. Select "RUN PROGRAM FILE" (5) from E/A or "UTILITY" (3) from TI-Writer.
5. Press **ENTER** (No further typing is necessary.)

To load from Mini-Memory.

1. Turn computer off.
2. Place Mini-Memory cartridge into cartridge slot.
3. Turn computer on and go to Mini-Memory menu.
4. Press **3**, RE-INITIALIZE.
5. Press **FCTN 6 (PROC'D)**
6. Press **1**, "LOAD AND RUN".
7. Type: DSK1.FAST-LOAD **ENTER**.

Fast-Term now attempts to find the file DSK1.CHARA1, the TI-Writer replacement character set.

You are now in the program. The program has auto-repeat on all keys.

Enter the name of a parameter file you have created with the program DEFAULT or hit **ENTER** to choose the following defaults:

1200	Baud rate for the modem.
EVEN	Parity for the modem.
PIO/1	Printer spooler (PIO/1 on TI RS232 card! (not CorComp).
RS232/1	Serial port to be used by modem.
FULL	Duplex (auto-echo of keys you type is OFF).
OFF	Log file is not on when you enter Fast-Term.
OFF	Printer spooler is off when you enter FAST-TERM.
OFF	There is no linefeed sent after an **ENTER** in file send.
WHITE	Text color.
DK. BLUE	Screen color.
40	Screen width.
DSK1.SESSION	File to log to.

COMMUNICATIONS

There are several parameters that need to be configured to work with the system you are calling: baud rate, parity, and duplex.

The baud rate must be set to be the same as what the other system expects, it must also be set to a rate which your modem can handle. The default baud rate is 1200. This is not compatible with a large number of modems. You must set your baud rate by pressing **CTRL 1**. You will go through a series of baud rates that Fast-Term handles:

1200--> 2400--> 4800--> 9600--> 19200--> 110--> 300--> 600--> 1200

The baud rates faster than 1200 baud are not yet useful in tele-communications, they are most useful to someone who wishes to use the TI as a hard-wired terminal for another computer. (There are some CP/M cards for the TI that connect to the TI through the serial port, Fast-Term is very useful for those.)

The parity must be set also. Consult the service you will be using to determine what parity they expect. The service will also have a suggested number of data-bits to use. In Fast-Term, data with EVEN or ODD parity is automatically SEVEN bits; data with NONE for parity is automatically EIGHT bits. Use **CTRL 3** to change the parity. Here is the sequence:

EVEN--> ODD--> NONE--> EVEN.

There is also DUPLEX. This controls how characters you type will appear on your screen. FULL duplex means that any character you type must be sent back to you by the other system before you actually see it. HALF duplex means that the other system expects the terminal you are using (TI, with Fast-Term) to automatically display the characters as you type them. There are very, very few systems around which expect you to use HALF duplex, Fast-Term's default is FULL duplex. You may change the duplex by pressing **FCTN SHIFT D**, all at the same time.

HARDCOPY

SPOOLER : Fast-Term provides a printer spooler feature. This means that if you have a printer you may instruct Fast-Term to send everything you will see on your screen to the printer as it is displayed on your screen. The spooler allows data to be displayed on your screen faster than the printer can print it, the printer is allowed to be as much as 4,000 characters behind what you see on your screen. If the printer does manage to get 4,000 behind, the incoming data will no longer be sent to the printer and Fast-Term will give you a warning message and a bell to tell you that the printer is very far behind. The spooler enables you to have a complete print-out of your whole session with the other system. Spooler is enabled by pressing **CTRL 2**. You may disable the spooler by pressing **CTRL 2** again. When you disable the spooler no more incoming data will be sent to the printer; data that the printer had gotten behind on will still continue to print.

SCREEN-DUMP : Fast-Term also provides a screen-dump feature. If you have a printer you may print a copy of the data which is on the screen. Press **FCTN 0** to freeze the screen (explained later) then press **FCTN SHIFT P**, all at the same time. All of the characters on the screen will be placed into the spooler, to be printed when the printer catches up. You must leave this freeze mode before you will see the rest of what the other system was sending you. (press **FCTN 0** again)

HARDWARE

You must tell Fast-Term how you have your hardware attached.

MODEM: Tell Fast-Term which serial port your modem is attached to by pressing **CTRL 4**. Pressing this key takes you through a list of ports available for the modem. Stop when you get to the port to which your modem is attached.

PRINTER: Tell Fast-Term which port your printer is attached to by pressing **CTRL 6**, this takes you though a list of ports available for your printer. Stop when you get to the right one. The default is the PIO port on the TI RS232 interface. If you have a CorComp serial interface and you have a printer for which you send data to PIO, you must select CorComp PIO from the list, *not* PIO! If you have an Axiom Parallax interface, you must choose Axiom PIO for the printer to work. People with serial interfaces built by yet another manufacturer should try all of the available choices to find one which works. It is possible that your interface is *not* compatible with Fast-Term. For such people, I suggest that you LOG to your printer (LOGging is explained later).

Set your printer baud rate (serial printers only) by pressing **CTRL 7**. This works in the same way as setting the modem baud rate.

Set your printer parity (serial printers only) by pressing **CTRL 5**. This works in the same was as setting the modem parity.

SENDING AND RECEIVING FILES

RECEIVING ASCII files (text that you can read). You can receive an ASCII file by LOGging to a file. Press **FCTN B**, this closes any previous file you were logging to and asks you for a new filename to log to. To log data you must give a filename. Then, at the point at which you want data to start going into the file, you must press **FCTN .** (period). All data will now go into the log file. Press **FCTN .** again when you get to the end of the data you want to save. All of the data is now in memory. To write it to disk you must press **FCTN B** again. This writes the file to disk, closes it, then asks you for a new filename. Enter a blank line if you don't want another log. If at some point in the transfer the memory gets full, Fast-Term will empty the buffer to disk then continue where it left off, still logging to the file. With a proper setup, through the DEFAULT program written in TI Basic, no data will be lost as Fast-Term writes the log to the file. For people whose serial interfaces don't allow them to use a printer with Fast-Term, you should log to "PIO" to get a hardcopy of your session. The log file is in a DISPLAY, VARIABLE 80 format.

You may clear the LOG's memory buffer at any time by pressing **FCTN Y**.

SENDING ASCII files. To send an ASCII file you must first press **FCTN N** and give the name of the file which you wish to send. Press **FCTN ,** (comma) to start sending the file. You will be asked of you want to send the file line-by-line. If you decide to send the file line-by-line you must press the spacebar every time you want to send another line from the file. Press **FCTN 4** to stop sending from the file. If you choose not to send the file line-by-line the file will be sent as quickly as the other system can handle it. The other system may send Fast-Term a CTRL S when Fast-Term gets ahead of what the other system can handle, and send a CTRL Q to Fast-Term when it has caught up. (XOFF/XON handshaking) When you have sent enough of the file you may hit **FCTN 4** to stop. Files you send may be in either DISPLAY, FIXED 80 or DISPLAY, VARIABLE 80 format on your disk.

Another option you may use when sending the file is to add a line-feed after every carriage return you send to the other system. This is useful when you sending a file to someone else's terminal for them to read. This feature is enabled by pressing **FCTN J**. It is disabled by pressing **FCTN J** again.

TE2 protocol

Sending files using the TE2 protocol. Press **FCTN SHIFT T**, all at the same time. This enables transmission of files with the TE2 protocol. Pressing those keys again enables ASCII send. Press **FCTN N** and give the name of the file which you wish to send. Press **FCTN ,** (comma) to send the file. You will be kept informed of the progress of the file transfer by an on-screen report of current block number, record number, and number of retries. A bell will ring when the transfer is complete. You may abort the transfer at any time by pressing **FCTN 4**.

Receiving files using the TE2 protocol. *First* press **FCTN N** and give the name of the file which you want data to go into. *Then* tell the other system to start sending you the file. You are given a report of the transfer's progress. Press **FCTN 4** at any time to abort the transfer.

XMODEM protocol

Sending files using the XMODEM protocol. Press **FCTN N** and give the name of the file you wish to send. Tell the other system to prepare to receive a file. Once the other system is ready, press **FCTN SHIFT X**, all at the same time. Press **S** or **ENTER** to <S>end file. The rest of the transfer is automatic. The transfer will be aborted if at any time the number of retries for one record exceeds ten. You may abort the transfer at any time by pressing **FCTN 4**, note that the other system will become confused by your abort, and will take about 90 seconds or so to give you a prompt back. (Most other systems.)

Receiving files using the XMODEM protocol. *First* press **FCTN N** and give the name of the file you want data to go into. *Then* tell the other system to start sending you the file. Now press **FCTN SHIFT X**, all at the same time. Press **R** for <R>eceive. You now need to choose which type of error checking you wish to use, CRC or CHECKSUM. CRC is more accurate than CHECKSUM, but not all systems support it (ask the sysop of that system about availability of CRC error checking). Note that if you choose CRC and the other system doesn't support it that you will be delayed for about 60 seconds. This is the time it takes XMODEM to automatically switch to CHECKSUM mode. Press **FCTN 4** to abort. Note that the other systems aborts immediately as well.

XMODEM sends TI disk directory information along with the file. It expects to see this information on any file it receives so that it may place the data into a file easily usable on the TI. Reception of a file which does not include directory information results in data being placed into a DIS/FIX 128 file format. Also, any file with the characteristics of DIS/FIX 128 is sent *without* the directory information. This results in all-around compatibility with files from other systems.

OTHER FEATURES

SCREEN CONTROL: you may set the width of the screen to either 40 or 80 characters. Press **CTRL 0** to toggle between 40 and 80 columns. Note that this operation does not clear the screen like other terminal programs.

If you choose 80 column mode you will have to window from side-to-side to see all of the screen. Pressing **FCTN 5** jumps you to the right, **FCTN L** jumps you to the left. You can set how far you jump by using the DEFAULT program which is written in TI Basic.

COLORS : You may change the screen color by pressing **FCTN 8**, you may change the text color by pressing **FCTN 7**.

SCREEN FREEZE: You may pause display of incoming data by pressing **FCTN 0**. This allows you to read what's on the screen without worrying about it scrolling of the top before you can read it. You unpause by pressing **FCTN 0** again. All of the data that came in while you were paused is now printed on the screen.

WINDOW BACK: this feature allows you to look at data which has already scrolled off of your screen. First enter FREEZE mode by pressing **FCTN 0**. Press the spacebar to quickly scan through the data, press <S> to go slowly through the data. You will see a *solid bar* on the screen when you have caught up to where you were when you began. During window back print spooling and LOGging are suspended. You may re-enable them during window back mode in order to save "lost" data, data which had already scrolled off of the screen. You may also change things like screen width and color while you're in window back mode. When you leave window back mode by pressing **FCTN 0** everything that you did like change screen parameters and LOGging is returned to the state it was in when you entered window back mode.

TIMER: Fast-Term has a timer you can use to display your connect time. It is constantly display in the upper-left corner of the screen when you enable it by pressing **FCTN K**. You may turn the timer off by pressing **FCTN K** again. Enabling the timer always resets it to zero. The timer is disabled if you have Fast-Term running at greater than 1200 baud. Note that this is *not* a time-of-day clock, it is *only* an elapsed time counter.

Fast-Term knows all of the ADM3A control codes and escape sequences.

ADM3A: control code	*Function*
>07	bell
>08	backspace (non-destructive)
>0A	linefeed
>0B	move up one line
>0C	move forward one space
>0D	carriage return
>1A	clear screen
>1E	home cursor
>1B '=' R C	moves cursor to row (R-32) and column (C-32)

SPECIAL: (not ADM3A, Fast-Term only)	
>0E	turns off reverse video
>0F	turns on reverse video

To set up a parameter file for use with Fast-Term, go to BASIC and run the program DEFAULT. The program is self-explanatory and menu-driven. Use it to set up everything that you can set up with function keys within Fast-Term, as well as some features that can only be changed by the parameter file.

Have fun!

NOTES

CONTROL KEYS FOR TERMINAL PROGRAM

FCTN	**1**	Transmits >7F (DELETE)
	2	Transmits >14 (control-T)
	3	Transmits >0E (control-N)
	4	Transmits BREAK TONE. Also "break" hung-up I/O functions
	5	Window Right (like Editor/Assembler and TI-Writer)
	J	Toggle transmission of linefeed after "CR" in ASCII file send mode
	K	Set the timer to zero.
	L	Window Left (Like Window Right, but other direction)
	6	Transmits >05 (CONTROL-E)
	7	Change foreground text color
	8	Change background text color
	0	Toggle window back mode (on/off) --> Spacebar for fast display, "S" for slow display
CTRL	**0**	Toggle screen width between 80 and 40 chars — does *not* clear screen!
	1	Change communication speed (baud rate)
	2	Toggle print spooler on/off
	3	Change parity (modem)
	4	Change modem port
	5	Change printer parity (only useful for serial printer)
	6	Change printer port
	7	Change printer baud rate (only useful for serial printer)
FCTN	**.**	Toggle log on/off
	N	Name the disk file to use for file send and TE2 transfers.
	Y	Clear the log
	B	Writes log to disk, gets name for new log
	,	Send disk file to other system
FCTN SHIFT	**D**	Toggle duplex between full and half
	P	Print screen in freeze or window-back mode.
	T	Enables transmission of file by TE2 protocol.
	X	Enters XMODEM file transfer mode.

CURSOR CONTROL

Use arrow keys or **CTRL** (**K,S,I,J**)

Un-documented control keys are same as TI-99/4(A) in Pascal scan mode.

FCTN = QUIT Leave program
CTRL = QUIT Leave program

Program knows all standard ADM-3A control codes and escape sequences.

Printer buffer turns off and rings the bell when it get full.

Log automatically writes to disk when it gets full.

Disk 19. Sprite Builder

Version:
Author: John Taylor
Requires: XB
Language: XB, AL
Updated:

A sprite designer in XB with assembly sub-routines. Very fast. Saves characters in MERGE format so you can add them to your programs.

```
dskdir. v2.0. 12-dec-96
Disk name               = SPRITEBLR1
Sectors total           = 360
Sectors used            = 322
Sectors available       = 36
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 DISKINFO02   17 PROGRAM    Y >022 016
002 >003 LOAD         10 PROGRAM    Y >032 009
003 >004 SLIDESHOW    42 PROGRAM    Y >03b 041
004 >005 SPRITE2DOC   51 PROGRAM    Y >064 050
005 >006 SPRITE2XB    41 PROGRAM    Y >096 040
006 >007 SPRITE4DOC   82 INT/VAR254 Y >0be 081
007 >008 SPRITE4OBJ   16 DIS/FIX 80 Y >10f 015
008 >009 SPRITE4XB    63 INT/VAR254 Y >11e 062
```

Disk 20. Sprite Builder

Version: *Author*: John Taylor
Requires: *Language*: *Updated*:

Assembly source code and 100+ pre-drawn sprites for use and editing. For use with disk 19.

```
dskdir. v2.0. 12-dec-96
Disk name               = SPRITEBLR2
Sectors total           = 360
Sectors used            = 310
Sectors available       = 48
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 0             2 INT/FIX 65 Y >022 001
002 >003 1             2 INT/FIX 65 Y >023 001
003 >004 2             2 INT/FIX 65 Y >024 001
004 >005 3             2 INT/FIX 65 Y >025 001
005 >006 4             2 INT/FIX 65 Y >026 001
006 >007 5             2 INT/FIX 65 Y >027 001
007 >008 6             2 INT/FIX 65 Y >028 001
008 >009 7             2 INT/FIX 65 Y >029 001
009 >00a 8             2 INT/FIX 65 Y >02a 001
010 >00b 9             2 INT/FIX 65 Y >02b 001
011 >00c A             2 INT/FIX 65 Y >02c 001
012 >00d APPLE         2 INT/FIX 65 Y >02d 001
013 >00e ASTRONT       2 INT/FIX 65 Y >02e 001
014 >00f B             2 INT/FIX 65 Y >02f 001
015 >010 BALL          2 INT/FIX 65 Y >030 001
016 >011 BARN          2 INT/FIX 65 Y >031 001
017 >012 BFLY          2 INT/FIX 65 Y >032 001
018 >013 BIRD          2 INT/FIX 65 Y >033 001
019 >014 BIRDRUN       2 INT/FIX 65 Y >034 001
020 >015 BLOB          2 INT/FIX 65 Y >035 001
021 >016 C             2 INT/FIX 65 Y >036 001
022 >017 CAR/1         2 INT/FIX 65 Y >037 001
023 >018 CAR/2         2 INT/FIX 65 Y >038 001
024 >019 CAR/VW        2 INT/FIX 65 Y >039 001
025 >01a CART          2 INT/FIX 65 Y >03a 001
026 >01b CAT           2 INT/FIX 65 Y >03b 001
027 >01c CHICKEN       2 INT/FIX 65 Y >03c 001
028 >01d CLOUD/1       2 INT/FIX 65 Y >03d 001
```

```
029 >01e CLOUD/2       2 INT/FIX 65 Y >03e 001
030 >01f CLOVER        2 INT/FIX 65 Y >03f 001
031 >020 COMET         2 INT/FIX 65 Y >040 001
032 >021 COW           2 INT/FIX 65 Y >041 001
033 >042 D             2 INT/FIX 65 Y >043 001
034 >044 DIAMOND       2 INT/FIX 65 Y >045 001
035 >046 DOG           2 INT/FIX 65 Y >047 001
036 >048 DRAGON        2 INT/FIX 65 Y >049 001
037 >04a DRAGONF       2 INT/FIX 65 Y >04b 001
038 >04c DUCK          2 INT/FIX 65 Y >04d 001
039 >04e DUMPTRK       2 INT/FIX 65 Y >04f 001
040 >050 E             2 INT/FIX 65 Y >051 001
041 >052 ELEPHAN       2 INT/FIX 65 Y >053 001
042 >054 F             2 INT/FIX 65 Y >055 001
043 >056 FARMER        2 INT/FIX 65 Y >057 001
044 >058 FISH          2 INT/FIX 65 Y >059 001
045 >05a FROG          2 INT/FIX 65 Y >05b 001
046 >05c G             2 INT/FIX 65 Y >05d 001
047 >05e GIRL          2 INT/FIX 65 Y >05f 001
048 >060 GRIFFEN       2 INT/FIX 65 Y >061 001
049 >062 H             2 INT/FIX 65 Y >063 001
050 >064 HEART         2 INT/FIX 65 Y >065 001
051 >066 HELICOP       2 INT/FIX 65 Y >067 001
052 >068 HORSE/1       2 INT/FIX 65 Y >069 001
053 >06a HORSE/2       2 INT/FIX 65 Y >06b 001
054 >06c HOUSE/1       2 INT/FIX 65 Y >06d 001
055 >06e HOUSE/2       2 INT/FIX 65 Y >06f 001
056 >070 I             2 INT/FIX 65 Y >071 001
057 >072 INVADER       2 INT/FIX 65 Y >073 001
058 >074 J             2 INT/FIX 65 Y >075 001
059 >076 JET           2 INT/FIX 65 Y >077 001
060 >078 K             2 INT/FIX 65 Y >079 001
061 >07a KANGARO       2 INT/FIX 65 Y >07b 001
062 >07c L             2 INT/FIX 65 Y >07d 001
063 >07e LION          2 INT/FIX 65 Y >07f 001
064 >080 M             2 INT/FIX 65 Y >081 001
065 >082 MAN/GUN       2 INT/FIX 65 Y >083 001
066 >084 MAN/RUN       2 INT/FIX 65 Y >085 001
067 >086 METEOR        2 INT/FIX 65 Y >087 001
068 >088 MONKEY        2 INT/FIX 65 Y >089 001
069 >08a MONKEY2       2 INT/FIX 65 Y >08b 001
070 >08c MONSTER       2 INT/FIX 65 Y >08d 001
071 >08e MOON          2 INT/FIX 65 Y >08f 001
072 >090 MOTORCY       2 INT/FIX 65 Y >091 001
073 >092 MOUSE         2 INT/FIX 65 Y >093 001
074 >094 N             2 INT/FIX 65 Y >095 001
075 >096 O             2 INT/FIX 65 Y >097 001
076 >098 OCTAPUS       2 INT/FIX 65 Y >099 001
077 >09a P             2 INT/FIX 65 Y >09b 001
078 >09c PHONIX        2 INT/FIX 65 Y >09d 001
079 >09e PINE          2 INT/FIX 65 Y >09f 001
```

```
080 >0a0 PLANE          2 INT/FIX 65 Y >0a1 001
081 >0a2 PLANET         2 INT/FIX 65 Y >0a3 001
082 >0a4 PYRAMID        2 INT/FIX 65 Y >0a5 001
083 >0a6 Q              2 INT/FIX 65 Y >0a7 001
084 >0a8 R              2 INT/FIX 65 Y >0a9 001
085 >0aa RABBIT         2 INT/FIX 65 Y >0ab 001
086 >0ac ROBOT          2 INT/FIX 65 Y >0ad 001
087 >0ae ROCKET         2 INT/FIX 65 Y >0af 001
088 >0b0 ROCKET2        2 INT/FIX 65 Y >0b1 001
089 >0b2 S              2 INT/FIX 65 Y >0b3 001
090 >0b4 SAILBOT        2 INT/FIX 65 Y >0b5 001
091 >0b6 SATALIT        2 INT/FIX 65 Y >0b7 001
092 >0b8 SATURN         2 INT/FIX 65 Y >0b9 001
093 >0ba SCHOONR        2 INT/FIX 65 Y >0bb 001
094 >0bc SNOWFLK        2 INT/FIX 65 Y >0bd 001
095 >0be SPACESP        2 INT/FIX 65 Y >0bf 001
096 >0c0 SPADE          2 INT/FIX 65 Y >0c1 001
097 >0c2 SPIDER         2 INT/FIX 65 Y >0c3 001
098 >0c4 SPRITE4SRC    74 DIS/VAR 80 Y >0c5 073
099 >10e STAR/1         2 INT/FIX 65 Y >10f 001
100 >110 STAR/2         2 INT/FIX 65 Y >111 001
101 >112 SUN            2 INT/FIX 65 Y >113 001
102 >114 T              2 INT/FIX 65 Y >115 001
103 >116 TRACTOR        2 INT/FIX 65 Y >117 001
104 >118 TRAIN/1        2 INT/FIX 65 Y >119 001
105 >11a TRAIN/2        2 INT/FIX 65 Y >11b 001
106 >11c TRAIN/3        2 INT/FIX 65 Y >11d 001
107 >11e TRAIN/4        2 INT/FIX 65 Y >11f 001
108 >120 TRAIN/5        2 INT/FIX 65 Y >121 001
109 >122 TRICYCL        2 INT/FIX 65 Y >123 001
110 >124 TRUCK/1        2 INT/FIX 65 Y >125 001
111 >126 TRUCK/2        2 INT/FIX 65 Y >127 001
112 >128 TUGBOAT        2 INT/FIX 65 Y >129 001
113 >12a U              2 INT/FIX 65 Y >12b 001
114 >12c V              2 INT/FIX 65 Y >12d 001
115 >12e W              2 INT/FIX 65 Y >12f 001
116 >130 WARRIOR        2 INT/FIX 65 Y >131 001
117 >132 X              2 INT/FIX 65 Y >133 001
118 >134 Y              2 INT/FIX 65 Y >135 001
119 >136 Z              2 INT/FIX 65 Y >137 001
```

Disk 20. Contents of File SPRITE4SRC

```
*****************************************************************************
* SPRITE BUILDER - ASSEMBLY LANGUAGE ROUTINES                                *
* VERSION #4 - COMPLETE ON APRIL 6, 1985                                     *
* WRITTEN BY JOHN E. TAYLOR                                                  *
*            2170 ESTALINE DRIVE                                             *
*            FLORENCE AL 35630                                               *
*            1(205)764-5248                                                  *
* THIS IS MY FIRST ATTEMPT AT ASSEMBLY LANGUAGE. IT MAY NOT BE GREAT BUT     *
* IT WORKS. HOPE THAT YOU ENJOY IT!                                          *
*****************************************************************************
       DEF  KEEPIT,GETIT,INVERS,TOPBOT,RGTLFT,CRSWAY,CURSOR
*
* EQUATES SECTION
*
VSBW   EQU  >2020        VDP SINGLE BYTE WRITE ROUTINE ADDRESS
VMBW   EQU  >2024        VDP MULTIPLE BYTE WRITE ROUTINE ADDRESS
VSBR   EQU  >2028        VDP SINGLE BYTE READ ROUTINE ADDRESS
VMBR   EQU  >202C        VDP MULTIPLE BYTE READ ROUTINE ADDRESS
GPLWS  EQU  >83E0        ADDRESS FOR BASIC'S REGISTERS
STATUS EQU  >837C        ADDRESS OF THE BASIC STATUS REGISTER
*
* RESERVE SPACE FOR REGISTERS AND BUFFERS
*
MYREG  BSS  32           MEMORY FOR MY REGISTERS
GRDBUF BSS  256          BUFFER TO READ IN WHOLE GRID
*
* KEEPIT - READ PATTERN OFF GRID AND WRITE TO PATTERN TABLE.
*
KEEPIT LWPI MYREG        SET UP MY REGISTERS.
       BL   @GRIDR       READ PATTERN FROM GRID SUBROUTINE
       BL   @PATW        WRITE PATTERN TO THE PATTERN TABLE
       B    @END         RETURN TO BASIC
*
* GETIT - READ PATTERN FROM PATTERN TABLE AND DRAW ON GRID.
*
GETIT  LWPI MYREG        SET UP MY REGISTERS.
       BL   @PATR        READ PATTERN FROM TABLE.
       BL   @GRIDW       WRITE PATTERN TO THE GRID.
       B    @END         RETURN TO BASIC.
*
* INVERS - INVERSE THE GRID IMAGE
*
INVERS LWPI MYREG        SET UP MY REGISTERS
       BL   @I0          INVERT THE GRID IMAGE
       B    @END         RETURN TO BASIC.
*
* TOPBOT - FLIP GRID FROM TOP TO BOTTOM.
*
```

```
TOPBOT LWPI MYREG        SET UP MY REGISTERS.
       BL   @TB0         READ THE GRID FROM BOTTOM TO TOP.
       BL   @WBG0        WRITE THE NEW GRID BACK OUT.
       B    @END         RETURN TO BASIC.
*
* RGTLFT - FLIP GRID FROM LEFT TO RIGHT.
*
RGTLFT LWPI MYREG        SET UP MY REGISTERS.
       BL   @RL0         READ THE GRID FROM LEFT TO RIGHT.
       BL   @WBG0        WRITE THE NEW GRID BACK OUT.
       B    @END         RETURN TO BASIC.
*
* CRSWAY - TURN THE GRID CROSS WAYS.
*
CRSWAY LWPI MYREG        SET UP MY REGISTERS.
       BL   @CW0         READ THE GRID FROM TOP TO BOTTOM, LEFT TO RIGHT.
       BL   @WBG0        WRITE THE NEW GRID BACK OUT.
       B    @END         RETURN TO BASIC.
*
* CURSOR - MOVE THE CURSOR ON THE SCREEN AND DO THE LINE FILL.
*
CURSOR LWPI MYREG
       BL   @CURSR
       B    @KEEPIT
* GRIDR - GRID READ SUBROUTINE.
* READS THE PATTERN OFF OF THE SCREEN.
* CREATES AND STORES PATTERN CODE IN GRDBUF.
GRIDR  LI   R5,GRDBUF    ADDRESS TO BEGIN WRITING CHARACTER CODE TO
       LI   R3,>C100     DOT ON CHARACTER
       LI   R13,35       STARTING ADDRESS TO READ
GRPLOP CLR  R14          GROUP LOOP -  ROWS DOWN TO READ
ROWLOP CLR  R2           ROW LOOP - CLEAR REGISTER TO BUILD BYTE IN
       LI   R4,>8000     SET BIT MASK
       CLR  R15          COLUMNS ACROSS TO READ
COLLOP CLR  R0           COLUMN LOOP - CLEAR VDP READ ADDRESS
       A    R15,R0       \ SET UP VDP ADDRESS TO READ
       A    R14,R0       - IT IS R0=R13+R14+R15
       A    R13,R0       /
       BLWP @VSBR        READ THE SCREEN CHARACTER
       CB   R1,R3        IS THE DOT ON?
       JNE  DOTOFF       IF NOT JUMP TO DOT OFF
       SOCB R4,R2        TURN BIT ON IN THE CHARACTER TO BE BUILT
DOTOFF SRL  R4,1         SHIFT MASK TO TURN NEXT LOWER BIT ON
       AI   R15,1        ADD ONE TO READ THE NEXT COLUMN
       CI   R15,8        IS IT THE END OF THE COLUMN LOOP?
       JNE  COLLOP       IF NOT GO BACK AND READ THE NEXT COLUMN
       MOVB R2,*R5+      MOVE THE DATA TO GRDBUF AND INCREMENT TO NEXT ADD
       AI   R14,32       MOVE DOWN ONE ROW.
       CI   R14,512      IS IT THE END OF THE ROW LOOP?
       JNE  ROWLOP       CONTINUE IF THERE ARE MORE ROWS TO DO
```

```
       AI   R13,8        SHIFT OVER TO THE SECOND TWO CHARACTERS
       CI   R13,51       IS THE GROUP LOOP DONE?
       JNE  GRPLOP       IF NOT THEN CONTINUE WITH THE SECOND LOOP
       B    *R11         EXIT SUBROUTINE.
* GRIDW - GRID WRITE SUBROUTINE.
* READS THE PATTERN FROM THE GRDBUF AND
* WRITES IT TO THE GRID.
GRIDW  LI   R5,GRDBUF    ADDRESS TO BEGIN READING CHARACTER CODE TO
       LI   R13,35       STARTING ADDRESS TO WRITE
LOPGRP CLR  R14          GROUP LOOP -  ROWS DOWN TO READ
LOPROW CLR  R2           ROW LOOP - CLEAR REGISTER TO BUILD FROM
       LI   R4,>8000     SET BIT MASK
       CLR  R15          COLUMNS ACROSS TO READ
       MOVB *R5+,R2      READ BYTE FROM GRDBUF AND INC TO NEXT ADDRESS
LOPCOL CLR  R0           COLUMN LOOP - CLEAR VDP WRITE ADDRESS
       A    R15,R0       \ SET UP VDP ADDRESS TO WRITE
       A    R14,R0       - IT IS R0=R13+R14+R15
       A    R13,R0       /
       LI   R1,>C000     SET UP DOT OFF CHARACTER CODE.
       COC  R4,R2        COMPARE AGAINST MASK.
       JNE  A            IF BIT OFF, JUMP TO A.
       AI   R1,>100      SET UP DOT ON CHARACTER.
A      BLWP @VSBW        WRITE THE CHARACTER TO THE GRID.
       SRL  R4,1         SHIFT MASK TO THE NEXT BIT ON.
       AI   R15,1        ADD ONE TO COLUMN LOOP
       CI   R15,8        IS IT THE END OF THE COLUMN LOOP?
       JNE  LOPCOL       IF NOT GO BACK AND READ THE NEXT COLUMN
       AI   R14,32       MOVE DOWN ONE ROW.
       CI   R14,512      IS IT THE END OF THE ROW LOOP?
       JNE  LOPROW       CONTINUE IF THERE ARE MORE ROWS TO DO
       AI   R13,8        SHIFT OVER TO THE SECOND TWO CHARACTERS
       CI   R13,51       IS THE GROUP LOOP DONE?
       JNE  LOPGRP       IF NOT THEN CONTINUE WITH THE SECOND LOOP
       B    *R11         EXIT SUBROUTINE.
* PATW - PATTERN WRITE.
* WRITE DATA IN GRDBUF TO THE PATTERN DESCRIPTOR TABLE.
PATW   LI   R0,1600      CHARACTER h PATTERN BEGINNING ADDRESS
       LI   R1,GRDBUF    BEGINNING LOCATION OF THE CHARACTER BUFFER
       LI   R2,32        NUMBER OF BYTES TO WRITE
       BLWP @VMBW        WRITE CHARACTER PATTERN TO MEMORY
       B    *R11         EXIT SUBROUTINE.
* PATR - PATTERN READ.
* READ PATTERN FROM DESCRIPTOR TABLE AND PUT INTO GRDBUF.
PATR   LI   R0,1600      CHARACTER h PATTERN BEGINNING ADDRESS
       LI   R1,GRDBUF    BEGINNING LOCATION OF THE CHARACTER BUFFER
       LI   R2,32        NUMBER OF BYTES TO WRITE
       BLWP @VMBR        READ CHARACTER PATTERN TO MEMORY
       B    *R11         EXIT SUBROUTINE.
* INVERS - TURN ON DOTS OFF AND OFF DOTS ON.
I0     LI   R13,35       ADDRESS TO BEGIN READING AT.
       CLR  R1           CLEAR SPACE TO READ THE CHARACTER INTO.
```

```
I4      CLR  R14          ROWS TO READ AND WRITE.
I3      CLR  R15          COLUMNS TO READ AND WRITE.
I2      CLR  R0           CLEAR ADDRESS TO READ AND WRITE.
        A    R15,R0       \
        A    R14,R0       - BUILD ADDRESS TO READ.
        A    R13,R0       /
        BLWP @VSBR        READ GRID DOT.
        CI   R1,>C100     IS THE DOT ON?
        JEQ  I5           IF SO JUMP TO I5.
        LI   R1,>C100     DOT WAS OFF SO TURN IT ON.
        JMP  I1           JUMP TO I1.
I5      LI   R1,>C000     DOT WAS ON SO TURN IT OFF.
I1      BLWP @VSBW        WRITE THE DATA BACK TO THE GRID.
        INC  R15          ADD 1 TO COLUMN.
        CI   R15,8        IS THE LOOP DONE?
        JNE  I2           IF NOT THEN JUMP BACK TO I2.
        AI   R14,32       MOVE DOWN ONE ROW.
        CI   R14,512      IS IT THE LAST ROW?
        JNE  I3           IF NOT THEN JUMP BACK TO I3.
        AI   R13,8        SHIFT TO NEXT GROUP.
        CI   R13,51       IS IT THE LAST GROUP?
        JNE  I4           IF NOT JUMP TO I4
        B    *R11         END THE SUBROUTINE.
* TOPBOT - TOP TO BOTTOM SUBROUTINE.
TB0     LI   R0,515       BOTTOM GRID ROW.
        LI   R1,GRDBUF    ADDRESS OF THE GRID BUFFER.
        LI   R2,16        NUMBER OF BYTES TO READ.
TB1     BLWP @VMBR        READ THE DATA FROM THE GRID INTO MEMORY.
        AI   R0,-32       MOVE UP ONE ROW.
        AI   R1,16        MOVE TO NEXT ADDRESS IN BUFFER TO WRITE TO.
        CI   R0,3         IS IT THE LAST ROW TO READ?
        JNE  TB1          IF NOT JUMP TO TB1
        B    *R11         END THE SUBROUTINE.
* RGTLFT - RIGHT TO LEFT SUBROUTINE.
RL0     LI   R0,50        FIRST ROW ON THE RIGHT MOST COLUMN.
        LI   R5,GRDBUF    ADDRESS OF THE GRID BUFFER.
        LI   R14,16       NUMBER OF ROWS TO READ.
RL2     LI   R15,16       NUMBER OF COLUMNS TO READ.
RL1     BLWP @VSBR        READ DOT FROM THE GRID.
        MOVB R1,*R5+      MOVE THE BYTE INTO THE GRID BUFFER.
        DEC  R0           LOOK BACK TO PRIOR DOT.
        DEC  R15          DROP 1 FROM NUMBER OF COLUMNS TO READ.
        JNE  RL1          IF NOT LAST COLUMN GOTO RL1
        AI   R0,48        GO BACK TO NEXT ROW, RIGHT MOST COLUMN.
        DEC  R14          ONE LSEE ROW TO READ.
        JNE  RL2          IF MORE TO READ GO TO RL2
        B    *R11         END SUBROUTINE.
* CRSWAY - CROSS WAYS SUBROUTINE.
CW0     LI   R0,35        BEGINNING ADDRESS TO READ.
        LI   R5,GRDBUF    ADDRESS OF GRID BUFFER.
```

```
       LI   R14,16       NUMBER OF COLUMNS TO READ.
CW2    LI   R15,16       NUMBER OF ROWS TO READ.
CW1    BLWP @VSBR        READ THE GRID DOT.
       MOVB R1,*R5+      MOVE BYTE TO GRID BUFFER AND INCREMENT ADDRESS
       AI   R0,32        MOVE DOWN TO NEXT ROW.
       DEC  R15          ONE LESS ROW TO READ
       JNE  CW1          IF MORE ROWS TO GO THE GOTO CW1
       AI   R0,-511      MOVE BACK TO FIRST ROW, NEXT COLUMN.
       DEC  R14          ONE LESS COLUMN TO READ.
       JNE  CW2          IF MORE COLUMNS TO GO, JUMP TO CW2
       B    *R11         END SUBROUTINE.
* WRITE GRID BUFFER BACK TO GRID.
WBG0   LI   R0,35        BEGINNING ADDRESS TO WRITE TO.
       LI   R1,GRDBUF    ADDRESS TO READ DATA FROM
       LI   R2,16        NUMBER OF BYTES TO WRITE
WBG1   BLWP @VMBW        WRITE DATA TO THE SCREEN
       AI   R0,32        DROP DOWN TO THE NEXT ROW.
       AI   R1,16        LOOK AT NEXT SET OF DATA TO WRITE.
       CI   R0,547       IS IT THE LAST ROW?
       JNE  WBG1         IF MORE TO DO JUMP TO WBG1
       B    *R11
* CURSOR - MOVES THE CURSOR.
CURSR  MOV  R11,R15      STORE THE RETURN ADDRESS IN R15
       MOV  @>8310,R0    MOVE LINK VARIABLE LIST ADDRESS INTO R0
       AI   R0,20        POINT TO FIRST VARIABLE ADDRESS
       BL   @GETPRM      BRANCH TO GET PARAMETER
       MOV  R1,R5        MOVE LINE FILL FLAG INTO R5
       BL   @GETPRM      BRANCH TO GET PARAMETER
       MOV  R1,R6        MOVE KEY PRESSED INTO R6
       BL   @GETPRM      BRANCH TO GET PARAMETER
       MOV  R1,R7        MOVE AUTO GO SETTING INTO R7
       BL   @GETPRM      BRANCH TO GET PARAMETER
       MOV  R1,R8        MOVE CURSOR COLUMN INTO R8
       BL   @GETPRM      BRANCH TO GET PARAMETER
       MOV  R1,R9        MOVE CURSOR ROW INTO R9
CURSR0 MOV  R6,R14       SAVE KEY PRESS IN R14
       CI   R6,48        WAS THE KEY PRESSED A ZERO?
       JNE  AX           IF NOT JUMP TO AX
       BL   @CURPOS      CALCULATE THE CURSOR POSITION IN R0
       LI   R1,>C000     DOT OFF CHARACTER.
       BLWP @VSBW        WRITE A DOT OFF TO THE SCREEN
       MOV  R7,R6        MOVE THE AUTO GO KEY INTO KEY PRESSED
       JMP  BX           JUMP TO BX
AX     CI   R6,49        IS THE KEY PRESSED A ONE?
       JNE  BX           IF NOT JUMP TO BX
       BL   @CURPOS      CALCULATE THE CURSOR POSITION IN R0
       LI   R1,>C100     DOT ON CHARACTER.
       BLWP @VSBW        WRITE A DOT ON CHARACTER TO THE SCREEN
       MOV  R7,R6        MOVE THE AUTO GO KEY INTO KEY PRESSED
BX     CLR  R12          CLEAR THE ROW FLAG
       CLR  R13          CLEAR THE COLUMN FLAG
```

```
        CI   R6,68        RIGHT ARROW PRESSED?
        JNE  AA           IF NOT JUMP TO A
        INC  R8           ADD ONE TO THE COLUMN
        JMP  CCHCK        JUMP TO COLUMN CHECK
AA      CI   R6,69        UP ARROW PRESSED?
        JNE  B            IF NOT JUMP TO B
        DEC  R9           SUBTRACT ONE FROM THE ROW
        JMP  RCHCK        JUMP TO ROW CHECK
B       CI   R6,83        LEFT ARROW PRESSED?
        JNE  C            IF NOT THEN JUMP TO C
        DEC  R8           SUBTRACT ONE FROM THE COLUMN
        JMP  CCHCK        JUMP TO COLUMN CHECK
C       CI   R6,88        DOWN ARROW PRESSED?
        JEQ  CC           IF SO JUMP TO CC
        CI   R6,79        IS THE AUTO GO SETTING TURNED OFF?
        JEQ  G            IF SO JUMP TO G
        B    @END         BAD KEY PRESS - RETURN TO BASIC
CC      INC  R9           ADD ONE TO THE ROW
        JMP  RCHCK        JUMP TO ROW CHECK
CCHCK   CI   R8,3         CURSOR PAST LEFT COLUMN
        JNE  D            IF NOT THEN JUMP TO D
        LI   R8,19        SET COLUMN TO RIGHT SIDE
        INC  R13          TURN THE COLUMN FLAG ON
        CI   R12,1        IS THE ROW FLAG ON?
        JEQ  G            IF SO THEN LEAVE
        DEC  R9           DECREASE THE ROW BY ONE
        JMP  RCHCK        JUMP TO ROW CHECK
D       CI   R8,20        IS THE CURSOR PAST THE RIGHT COLUMN?
        JNE  G            IF NOT JUMP TO G
        LI   R8,4         CHANGE COLUMN TO FAR RIGHT POSITION
        INC  R13          TURN THE COLUMN FLAG ON
        CI   R12,1        IS THE ROW FLAG ON?
        JEQ  G            IF SO JUMP TO G
        INC  R9           ADD ONE TO THE ROW
RCHCK   CI   R9,1         CURSOR PAST THE FIRST ROW?
        JNE  E            IF NOT JUMP TO E
        LI   R9,17        CHANGE TO LAST ROW
        INC  R12          TURN THE ROW FLAG ON
        CI   R13,1        IS THE COLUMN FLAG ON?
        JEQ  G            IF SO JUMP TO G
        DEC  R8           SUBTRACT ONE FROM THE COLUMN
        JMP  CCHCK        JUMP TO COLUMN CHECK
E       CI   R9,18        CURSOR PAST THE LAST ROW?
        JNE  G            IF NOT THEN JUMP TO G
        LI   R9,2         CHANGE THE ROW TO THE FIRST ROW
        INC  R12          TURN THE ROW FLAG ON
        CI   R13,1        IS THE COLUMN FLAG ON?
        JEQ  G            IF SO JUMP TO G
        INC  R8           ADD ONE TO THE COLUMN
        JMP  CCHCK        JUMP TO COLUMN CHECK
```

```
G       MOV  R14,R6       MOVE KEYPRESS BACK INTO R6
        DEC  R5           IS IT A LINE FILL?
        JNE  CURSR0       IF SO JUMP BACK AND DO IT AGAIN.
        BL   @CURPOS      WHAT IS THE NEW CURSOR POSITION?
        BLWP @VSBR        WHAT IS THE CHARACTER AT THAT POSITION
        SWPB R1           SWAP BYTES
        AI   R1,-96       REMOVE OFFSET
        MOV  R1,R10       STORE THE RESULT IN R10
        MOV  @>8310,R0    POINT TO LINK LIST LOCATION
        AI   R0,44        POINT TO CURSOR COLUMN
        MOV  R8,R6        GET READY TO RE-WRITE THE NEW COLUMN
        BL   @GIVPRM      WRITE THE PARAMETER
        MOV  R9,R6        GET READY TO RE-WRITE THE NEW ROW
        BL   @GIVPRM      WRITE THE PARAMETER
        MOV  R10,R6       GET READY TO WRITE THE CHARACTER
        BL   @GIVPRM      WRITE THE PARAMETER
        B    *R15         RETURN
* GETPRM - GET THE PARAMETER
GETPRM LI   R1,GRDBUF     LOCATION TO WRITE INTO
        LI   R2,2         BYTES TO READ
        BLWP @VMBR        READ THE DATA
        MOV  R0,R14       SAVE CONTENTS OF R0
        MOV  @GRDBUF,R0   MOVE ADDRESS TO READ INTO R0
        INC  R0           POINT TO NUMERIC PART
        CLR  R1           CLEAR R1
        MOVB *R0,R1       READ THE NUMBER
        MOV  R14,R0       RESTORE CONTENTS INTO R0
        AI   R0,8         POINT TO THE NEXT PARAMETER
        SWPB R1           SWAP BYTES
        B    *R11         RETURN
* GIVPRM - GIVE THE PARAMETER
GIVPRM LI   R1,GRDBUF     LOCATION TO WRITE INTO
        LI   R2,2         BYTES TO READ
        BLWP @VMBR        READ THE DATA
        MOV  R0,R14       SAVE CONTENTS OF R0
        MOV  @GRDBUF,R0   MOVE ADDRESS TO READ INTO R0
        INC  R0           POINT TO NUMERIC PART
        SWPB R6           SWAP BYTES
        MOVB R6,*R0       WRITE THE NUMBER
        MOV  R14,R0       RESTORE CONTENTS INTO R0
        AI   R0,8         POINT TO THE NEXT PARAMETER
        B    *R11         RETURN
* CURPOS - CALCULATE THE CURSOR POSITION
CURPOS MOV  R9,R0         MOVE THE ROW TO R0
        DEC  R0           SUBTRACT ONE
        SLA  R0,5         MULTIPLY BY 32
        A    R8,R0        ADD THE COLUMN TO R0
        DEC  R0           SUBTRACT 1
        B    *R11         RETURN
* END - RETURN TO BASIC.
END     LWPI GPLWS        RESTORE BASIC'S WORKSPACE REGISTERS
```

```
CLR  @STATUS      CLEAR BASIC'S STATUS REGISTER
B    @>0070       BRANCH BACK TO BASIC
END
```

Disk 21. Pilot 99

Version:
Author: Thomas Weithofer
Requires: XB, EA
Language: Forth
Updated:

Excellent implementation of the Pilot programming language. Includes full support for the TI's sound and graphics. Two disks. Complete documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = PILOT
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 PILOT         6 DIS/FIX 80   >022 005
002 >003 PILOTDATA    39 PROGRAM      >027 038
003 >004 ZZZZZZZZZZ  313 DIS/FIX128   >04d 283 >005 029
```

Disk 22. Pilot 99

Version: *Author*:
Requires: *Language*: *Updated*:

See disk 21.

```
dskdir. v2.0. 12-dec-96
Disk name               = P-MANUAL
Sectors total           = 360
Sectors used            = 336
Sectors available       = 22
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 LOAD          2 PROGRAM      >022 001
002 >003 P-MANUAL    334 DIS/VAR 80   >023 325 >004 008
```

Disk 22. Contents of File P-MANUAL

```
PPPPPPPPPPP   IIIIIIIIIII   LLL           OOOOOOOOOOO   TTTTTTTTTTT
PPPPPPPPPPP   IIIIIIIIIII   LLL           OOOOOOOOOOO   TTTTTTTTTTT
PPP     PPP       III       LLL           OOO     OOO       TTT
PPP     PPP       III       LLL           OOO     OOO       TTT
PPPPPPPPPPP       III       LLL           OOO     OOO       TTT
PPPPPPPPPPP       III       LLL           OOO     OOO       TTT
PPP               III       LLL           OOO     OOO       TTT
PPP               III       LLL           OOO     OOO       TTT
PPP               III       LLL           OOO     OOO       TTT
PPP           IIIIIIIIIII   LLLLLLLLLLL   OOOOOOOOOOO       TTT
PPP           IIIIIIIIIII   LLLLLLLLLLL   OOOOOOOOOOO       TTT
```

```
      99999999999   99999999999
      99999999999   99999999999
      999     999   999     999
      999     999   999     999
      99999999999   99999999999
      99999999999   99999999999
             9999          9999
             9999          9999
         9999          9999
         9999          9999
      9999          9999
      9999          9999
```

I would like to thank the following people for their invaluable help in creating PILOT 99: Dr. James Delaney, Xavier University; Dr. David Berry, Xavier University; Dr. Alam, Xavier University, Dr. Brewer, Xavier University; and Mr. Roy Thompson, Cin-Day Users Group. I would also like to thank the following individuals at Texas Instruments: Mr. Bill Gosett, Ms. Shirley Mathews, Mr. David Horn, and all those at the Technical Assistance Center and the TI-CARES Information Center.

Finally, I would like to thank my family who has put up with my devotion to PILOT 99 for the last few months.

The computer language known as PILOT, which stands for Programmed Inquiry Learning Or Teaching, was developed by Professor John A. Starkweather of the University of California in San Francisco in the late 1960s and early 1970s. PILOT was developed to be a language for developing computer-aided-instruction (CAI) lessons. The developer of the language insisted that it be easy to learn and use.

PILOT 99 is an advanced implementation of PILOT for the Texas Instruments TI-99/4A. The TI-99/4A has advanced features which would not be available for use if the standard PILOT statements were not supplemented. PILOT 99 adds statements for using sprites, bit-map graphics, character graphics, sound, and files. The concept of ease of learning the language and use are maintained by PILOT 99.

With the advanced statements available with PILOT 99 the PILOT language has been transformed from a language designed solely for CAI lessons to a language that is easy to learn and use for writing home management programs as well as CAI lessons .

The manual you have before you is not a textbook, is explains the PILOT 99 statements and how to write and run PILOT 99 programs. It is beyond the scope of this manual to teach one the concepts of programming a computer. PILOT The manual you have before you is not a textbook, is explains the PILOT 99 statements and how to write and run PILOT 99 programs. It is beyond the scope of this manual to teach one the concepts of programming a computer. PILOT however is an easy to learn language so these limitations should not impose a burden. Tutorials on PILOT 99 and possible other languages available on the TI-99/4A will be available to those that chose to purchase the PILOT 99 program through the PILOT 99 Program Exchange.

Please remember that PILOT 99 is freeware. Freeware doesn't mean that you receive the program free of charge, it is simply that I am allowing you to review the PILOT 99 system before purchasing it. I can not make you pay for PILOT 99 if you use it but you must remember that your are aiding the downfall of the freeware system, and that the next nice piece of software you want you may have to pay up front to find out it isn't for you! Another words don't bite the had that feeds you!. Also since this is being distributed under the Freeware concept feel free to pass it along to others but with the following restrictions.

1. Both the system disk and the manual disk must be given.
2. You may not charge in anyway shape or form for giving another the PILOT 99 system.

Please note that TI Users Groups are given the right to distribute the program as long as the preceding rules are followed.

I would appreciate any comments and/or suggestions you have concerning the PILOT 99 programming language.

Thank you,

Thomas P. Weithofer
1000 Harbury Dr.
Cincinnati, OH 45224

WRITING A PILOT 99 PROGRAM

When writing or creating a PILOT 99 program you use an editor to create a text file that contains you program. If you have the Editor/Assembler version of PILOT 99 then you should use the editor accompanying the module and follow the directs as given in the Editor/Assembler manual. If you are using the Extended BASIC version of PILOT 99 then you should use either an editor that you have or the one that accompanied the PILOT 99 system if you asked for it. If you are using the editor that accompanied PILOT 99 then please see the summary of it printed by the manual disk.

After you have created your program with the editor you must save it to disk in order for PILOT 99 to be able to run it (please note: those using TI-Writer should Print File, PF command, to disk not Save File). The filename used in saving the file to disk is the filename you should give PILOT 99 when running the program.

RUNNING PILOT 99 PROGRAMS

The first step in running a PILOT 99 program is loading the PILOT 99 language. The method used in loading the PILOT 99 language depends on the version in use, thus follow the instructions for your version.

EXTENDED BASIC VERSION

1. Turn on the computer in the normal manner.
2. Place the PILOT 99 System Disk in drive 1.
3. Press any key to make the master selection screen appear. Then press the key corresponding to the number beside TI EXTENDED BASIC.
4. PILOT 99 will load by itself from this point.

EDITOR/ASSEMBLER VERSION

1. Turn on the computer in the normal manner.
2. Place the PILOT 99 System Disk in drive 1.
3. Press any key to make the master selection screen appear. Then press the key corresponding to the number beside EDITOR/ASSEMBLER.
4. Select option 3, LOAD AND RUN, of the Editor/Assembler option list.
5. At the FILE NAME? prompt type DSK1.PILOT and press **ENTER**.
6. PILOT 99 will load and automatically run.

After the PILOT 99 language is loaded you will be asked for the program name. Enter the filename of the PILOT 99 program you wish to run. PILOT 99 will check to see that the name is valid by accessing the file. You will then be asked for the device name of you printer, if you wish to use a printer enter the device name else simply press enter. Your PILOT 99 program will now be run.

If any errors occur you will be given an error message and the statement number that caused the error, and told to press any key to continue. Make sure to copy all information off the screen before pressing a key because once a key is pressed you are returned to the power up title screen.

FUNDAMENTAL PILOT

The following terms and conventions are used in PILOT.

Note: PILOT 99 requires that all reserved words of the language be in uppercase.

ANSWER BUFFER

This is a section of memory set off by the PILOT language for inputting and outputting information to the screen, keyboard, and files. The answer buffer is also used in comparing for a correct answer by the M: and MJ: statements. As a whole the programmer need not worry about the actual workings of the answer buffer.

YES FLAG

The yes flag is controller of all conditional execution of PILOT 99. What this means is that PILOT 99 has no if statements put allows the programmer to decide if a statement should be executed depending on the YES FLAG.

Various PILOT statements effect the YES FLAG, for example the M: statement. For more information on using the YES FLAG for conditional execution see the STATEMENT MODIFIERS section.

STATEMENT FORMAT

PILOT 99 is relatively free form this means that in most cases multiple spaces are ignored. The general form of a statement is as follows:

```
STATEMENT-IDENTIFIER MODIFIER : STATEMENT-DATA
```

The first character of any PILOT 99 statement must be the start of a statement-identifier, imbedded spaces are not allowed. If a modifier is being used it must be immediately following the statement-identifier. The colon must immediately follow the modifier or statement-identifier if no modifier is being used and can not be separated by a space from either one.

The format of statement-data is governed by the type of statement and can be found in that statements explanation.

The only other time that spaces are important is when printing to the screen, printer, or writing to a file. Leading spaces after the colon are assumed to be characters to be printed, the same exists for any spaces imbedded with in statement-data.

STATEMENT MODIFIERS

The PILOT language allows any statement to be modified in the three ways by the use of modifiers. The "Y" or YES modifier causes a statement to be executed only when the YES FLAG is in a true state, defined as having a non-zero value. The "N" or NO modifier causes a statement to be executed only if the YES FLAG is in false state. The third type of modifier allows the user to specify that the result of a string or numeric expression should be used, if the expression is true, non-zero result, the statement will be executed and if the result is false or zero the statement will not be executed. There are however a few restrictions currently imposed on the use of this modifier: 1) the expression mut be enclosed within parentheses. 2) The second item of the expression must be a valid representation of a variable.

NUMERIC VARIABLES

PILOT 99 allows use of up to 26 numeric variables. The capital letters of the alphabet are used as there names, A,B,C,D...Z. A numeric variable's name must always be preceded by a pound sign (#), no spaces may be imbedded between the pound sign and the numeric variable's name.

STRING VARIABLES

PILOT 99 allows the use of 13 string variables, A,B,C,D. . .M. The reason behind the few number of string variables is due to memory limitations. Each string variable can hold a maximum of eighty characters. A string variable must be immediately preceded by a dollar sign ($).

NUMERIC CONSTANTS

Numeric constants are permitted by PILOT 99 in numeric expressions. They can be placed in the answer buffer and can also be placed in a variable . There is a limit if eighty characters that can be inputted. To check for a correct answer use this statement and a M: or MJ: statement.

STRING CONSTANTS

String constants are not permitted by PILOT 99 at this time.

The following numeric operators and functions are supported by PILOT 99. The standard TI format for numeric expressions apply to PILOT 99, also the standard priority of operations apply.

+	addition
*	multiplication
^	exponentiation
-	subtraction
/	division

=	equal to
>	greater than
<	less than
<>	not equal to
>=	greater than or equal to
=<	less than or equal to

ATN	arctangent
COS	cosine
SQR	square root
EXP	exponential
TAN	tangent
SIN	sine
LOG	logarithm
NEG	negative of

Parentheses can be used to alter the normal order of operations.

STRING OPERATORS

PILOT 99 supports only the concatenation of strings. In the third type of modifier you can compare strings with the normal relational operators.

LABELS

Labels are statements used for separating parts of a program or being the object of a branch statement. Label statements must start with an asterisk which is followed by up to ten uppercase alphabetic characters.

AS:

The AS: statement inputs a single character from the keyboard and places it in the answer buffer. This statement is useful for multiple choice questions. Like with the A: statement use the M: or MJ: statement to determine if an answer is correct.

C:

The C: statement is used to assign a numeric variable a value. The operators and functions that are permitted by PILOT were listed earlier. This statement allows you to keep track of the number of correct and/or incorrect answers. It also allows you to mathematics those giving you the ability to do problems as you ask them.

CH:

This one statement makes it easy to clear the text screen and home the cursor.

CS:

This statement make sit possible to combine or concatenate strings and assign the result to a string.

E:

The E: statement is two statements in one. If there is no subroutine call in effect then the E: statement causes the PILOT program to end. If a subroutine call is in effect then the E: statement transfers control back to the statement following the last U: statement.

J:

The J: statement is very powerful because it allows you the programmer to control the order that the program is executed. The J: statement causes execution of the program to be transferred to the label given in the statement. This statement with a modifier are like the IF..THEN statement in BASIC. This statement and the YES or NO modifier and the M: statement make it extremely easy to perform branches depending on input from the keyboard.

JM:

This modified version of the J: statement makes it extremely easy to do things depending on the choices entered from the keyboard. This statement should be used along with the M: statement, somewhat like a team.

M:

This statement has been said to be the most powerful of all PILOT statements. The M: statement make s it easy to check for a correct answer out of a number of choices. For example, this statement will allow you to answer GEORGE, WASHINGTON, GEORGE WASHINGTON, G.W. ; and so on and still receive credit for a correct answer. This statement and the power of the three kinds of modifiers make a very useful combination.

MJ:

This statement is a slightly modified M: statement, if no match is found the next M: or NJ: statement is performed, skipping all statement in between the two statements. This gives you the ability to make long lists of correct answers or to perform special tasks depending on which of a series of correct answers where given.

PR:

This statement makes it easy to set problems apart within a PILOT program.

R:

The R: statement is totally ignored by PILOT but that doesn't mean you should ignore it! This statement allows you to add comments within your program to make it easy for you and others to understand.

T:

The T: statement allows you to place a line of text on the display screen, in either text mode or the additional graphics mode. With the statement you can also type the values of numeric and string variables. This statement allows one to place problems or questions on the screen.

TH:

This statement is a slight modification to the T: statement in that the cursor remains in the position following the displayed text. This can make it easy to have an answer inputted next to a question.

TP:

If you have ever done any programming in other languages on the TI-99/4A you know that it is not to easy to print things to your printer. But PILOT's TP: statement makes it easy to do just that. With this statement you don't have to worry about opening and closing the printer the PILOT system does all that for you. You can also use this statement to print the values of numeric and string variables to the printer.
U:

Here is another powerful PILOT statement. This statement in combination with the E: statement allow you to create subroutines. Subroutines should be used for often need tasks. The use of subroutines make it easy to create and understand programs. Subroutines can also be used to expand the PILOT 99 language, for example writing a subroutine to draw smaller circle inside each other gives you a modified DC: statement that fills in a circle.

CHARACTER GRAPHICS STATEMENTS

The TI-99/4A allows you to change the pattern of characters to suit the needs of a program. There are 256 characters available to the user, some of these are used for the standard display characters. The 256 characters are divided into 32 groups, each group can have a different foreground and background color. The foreground color determines the color given to the part of the character that is visible while the background color determines the invisible part of the character's color. Also provided with PILOT 99 are statements allowing rapid displaying of a character vertically or horizontally along the screen.

CC:

This statement allows you to change the foreground and background colors of a character set. This statement gives you the power to emphasis text, also in combination with the CP: statement to easily make color graphics.

CP:

This statement allows you to define the pattern of a character. The instructions for defining a character can be found in the explanation of this statement. This statement can be used for defining large letters, numbers, symbols such as logos, etc. Another use of this statement could be for defining special symbols for mathematics, foreign languages, chemistry, etc.

HC:

The HC: statement operates almost identically to the TI BASIC HCHAR subprogram. It allows one to place a character anywhere on the screen and to repeat a number of times if wanted horizontally across the screen.

IT:

The IT: statement places PILOT in the text mode. The screen is divided into 24 rows of 32 columns.

SN:

This statement permits the programmer to change the screen color.

TC:

The TC: statement gives the programmer the ability to locate the text cursor anywhere on the screen. This in combination with the T: or TH: statements make it possible to place graphics and/or text anywhere on the screen.

VC:

The VC: statement serves the same function as the HC: statement puts will repeat the character vertically.

SPRITES

The TI-99/4A allows the use of up to 32 sprites. Sprites have additional features not found in character graphics. Sprites can be located at any of the over 4,000 pixels of the screen, they can also be four different sizes, and can be placed in auto-motion. Once a sprite is placed in auto-motion the program doesn't need to control the sprite.

GP:

The GP: statement performs the same function as the CP: statement but for sprite-characters. Sprite-characters are used in defining the patterns of sprites. There are 42 sprite-characters available for use.

SA:

The SA: statement gives the programmer the ability to detect if any two sprites are touching. This statement makes it possible to perform a certain action if two sprites touch each other.

SC:

Sprites have color just as do characters (the background color of sprites is always transparent). The SC: statement is used to change or set the color of a sprite:

SD:

The ability to delete a given sprite is handled by the SD: statement.

SG:

The SG: statement deletes all sprites.

SH:

The SH: statement is used to detect if two sprites are in coincidence or touch each other. Once again this allows actions to be taken on the coincidence of two sprites.

SL:

One of the powers of sprites is their ability to be located anywhere on the screen, not be limited by character cells. The SL: statement is used for locating a sprite on the screen.

SM:

Probably the most powerful feature of sprites is their ability to be placed in motion once and the left to continue that motion on their own. The SM: statement allows the programmer to specify the motion of a sprite. This statement can not be used in graphics mode.

SP:

The SP: statement is used to set the pattern of a sprite.

SS:

Sprites can be four different sizes which a set by this statement.

The attributes of color, pattern, location, size, and motion of a sprite can be changed at anytime.

BITMAP GRAPHICS

The TI-99/4A's advanced graphics chip allows the programmer to set individual pixels of the screen on or off and also to specify there color. For specifying color in the graphics mode remember that each group of eight horizontal pixels have the same colors. In graphics mode the screen is divided into two parts. The top two thirds of the screen is in bit-map mode this is where the following statements perform their operations. The bottom third of the screen is in text mode. The two different parts of the screen allow you to draw detailed illustrations and to ask questions about them.

DC:

This statement makes it easy to draw a circle on the graphics screen.

DL:

This statement will draw a line on the graphics screen

DR:

This statement draws a rectangle on the graphics screen.

GC:

This statement allows the programmer to specify the foreground and background colors for drawing graphics.

IG:

This statement causes PILOT 99 to go into the graphics mode.

TG:

This statement permits the standard ASCII characters to be displayed on the graphic part of the screen.

FILE STATEMENTS

PILOT 99 allows the use of a data file. The data fill format is preset to INTERNAL, FIXED 80, RELATIVE, UPDATE, However the average use need not worry about this. What all this means is that you have a maximum of eighty characters that can be read in or written out at one time, plus you can read and write to the file at the same time, plus you can access records in any order.

CF:

This statement is use to close the data file. You should always close the data file before ending the program. If your program ends with a error the data file will be closed by the PILOT 99 system.

OF:

The OF: statement is used to open a data file.

RE:

The RE: statement reads a record from the data file into the answer buffer. The RE: statement can also place the record into a variable.

RF:

The RF: statement moves the file pointer (the control for which record is next to be read or written) to a given record. This statement allows the programmer to access records of a data file in any order.

WA:

The WA: statement writes the contents of the answer buffer out to the data file. This statement makes it easy to keep the answers of session in a data file.

WR:

The WR: statement allows the programmer to put any characters out to the data file, this statement is also used to save the values of variables.

MISCELLANEOUS STATEMENTS

The following statements are additional statements that fit in to no special category.

FB:

The FB: statement checks the fire button of one of the joystick units.

JS:

The JS: statement returns the position of the joysticks unit specified. The joysticks are duplicated on the keyboard, for more information on this see the JS: statement .

BW:

The BW: statement makes the beginning of a While loop. This statement and the WH: statement control a while loop.

WH:

The WH: statement is the controller of the while loop. A while loop allows repeated execution of a group of PILOT 99 statements while a condition is true.

LP:

The LP: statement is the first statement of a LOOP structure, this is very similar to BASIC's FOR..NEXT. The LOOP structure allows a group of PILOT 99 statements to be executed a set number of times.

EL:

The EL: statement marks the end of the LOOP structure.

S:

The TI-99/4A has the ability to create music using three independent voices. The S: statement gives the programmer access to this power. This statement makes it possible to use music as a reward for a correct answer or to teach the concepts of music theory.

ERROR MESSAGES

If your PILOT 99 program generates a error you will informed with an error message the statement that caused the error. After you are finished copying the error message and statement number down press a any key and you will be returned to the title screen.

`syntax error`

The statement contains invalid syntax for that statement.

`invalid variable`

The name of a numeric or string variable is invalid.

`bad value`

The value supplied is bad or out of range. This usually will happen for incorrect sprite numbers, color codes, argument of a numeric function out of range, row or column out of range, dot-row or dot-column out of range, bad ASCII code, etc.

`nesting error`

The means that you have attempted to nest a WHILE LOOP, LOOP, or SUBROUTINE to many times.

`FILE I/O ERROR`

This can be caused if you fail to open a file and try to read, write, or restore it. Also caused if a invalid filename is given in a OP: statement. If the last statement in a program (usually a E: but any that don't have to do with files) causes the error then you have jumped to a label that does not appear in the program.

`floating point error`

Caused by a number of things: dividing by zero, result too large or too small, bad argument for a numeric function, etc.

`incorrect screen mode`

The statement that caused the error can not be executed in the screen mode that PILOT 99 is currently in.

Reference Section

A: Accept

Format: A:[variable]

The A: statement accepts up to a maximum of eighty characters into the ANSWER BUFFER. Optionally a variable can be set equal to the inputted string. If a numeric variable is specified the inputted string is converted to its numeric equivalent.

Examples:

```
A:
```

The string inputted is placed in the ANSWER BUFFER.

```
A: $A
T: $A
```

Inputs a string from the keyboard and places it into the string variable A. The ANSWER BUFFER also contains the string. The string is then printed.

```
A: #A
T: #A
```

Inputs a string from the keyboard and places the numeric equivalent in the numeric variable A. Then the value of A is printed.

AS: Accept Single Character

Format: AS:

The AS: statement accepts a single character into the ANSWER BUFFER. The ANSWER BUFFER. then can be used in the normal manner.

Example:

```
AS:
```

Inputs a single character into the ANSWER BUFFER.

BW: Begin While

Format: BW:

The BW: statement signifies the beginning of a WHILE LOOP. The body of the WHILE LOOP consists of all the statements between the BW: statement and the next WH: statement. WHILE LOOPS maybe nested three deep.

Example:

```
OF: DSK1.TEMP
BW:
RE: #N
WH: #N<>0
```

Opens a data file and reads in numbers until a 0 is encountered.

C: Compute

Format: C: N_Variable <- Numeric-expression

The C: statement is used to assign a value to a numeric variable. Numeric expression may consist of the following elements: constants, variables, operators, and functions.

A constant is any legal representation of a floating-point number. Exponential notation for a floating-point number is not allowed by PILOT 99. For more information on constants please see Page II-9 of the User's Reference Guide.

PILOT 99 allows the use of the following numeric operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
=	Equal to
<>	Not Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
OR	Logical OR
AND	Logical AND

PILOT 99 allows the use of the following numeric functions:

ATN	Arctangent
COS	Cosine
EXP	Exponential
INT	Integer
LOG	Natural Logarithm
NEG	Negative
SIN	Sine
SQR	Square Root Function
TAN	Tangent

The normal order of operations can be altered by using parenthesis. Items inclosed with parenthesis are performed first.

Example:

```
C: #N<-1
LP:
C: #S<-#N^2
C: #N<-#N+1
T: #S
EL:
```

This PILOT 99 code will print out the squares of the 10 numbers 1 through 10.

CC: Character Color TEXT MODE ONLY

Format: CC: character-set#,foreground-color,background-color

CC: is used to specify the foreground and background colors of a character-set. Each character has two colors. The color of the dots that make up the character itself is called the foreground -color. The color that occupies the rest of the character position on the screen is called the background-color. Foreground-color and background-color must have a value from 1 through 16. The color codes are shown below.

COLOR	*CODE*	*COLOR*	*CODE*
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

When Using CC: you must specify to which character set the colors are to be The character-set numbers are given below.

Character Set	*ASCII Codes*	*Character Set*	*ASCII Codes*	*Character Set*	*ASCII Codes*	*Character Set*	*ASCII Codes*
0	0-7	8	64-71	16	128-135	24	192-199
2	8-15	9	72-79	17	136-143	25	200-207
3	16-23	10	80-89	18	144-151	26	208-215
4	24-31	11	88-95	19	152-159	27	216-223
5	32-39	12	96-103	20	160-167	28	224-231
6	40-47	13	104-111	21	168-175	29	232-239
7	48-55	14	112-119	22	176-183	30	240-247
7	56-63	15	120-127	23	184-191	31	248-255

Examples:

```
IT:
CC: 8,13,16
```

The eighth character set (@ -G) is made green on white.

```
IT:
C: #N<-1
BW:
CC: #N,13,12
C: #N<-#N+1
WH:#N<32
```

This PILOT 99 code will change all the standard characters to Dark Green on Light Yellow.

CF: Close File

Format: CF:

The CF: statement closes the data file opened by a previous OF: statement. If a previous OF: statement was not executed this statement will result in a file error.

Example:

```
OF: DSK1.TEMP
WR: HI THERE
WR: WRITTEN BY PILOT 99
CF:
```

This code will open a data file and print two records then close the data file.

CH: Clear Home TEXT MODE ONLY

Format: CH:

The CH: statement fills the text mode screen with spaces and places the text cursor in the upper left hand corner.

Example:

```
IT:
T: Watch me
T: clear the screen
T: and go home
CH:
T: See that!!!!
```

Writes three lines to the screen and then clears the screen and homes the cursor. The last line written appears at the top of the screen.

CP: Character Pattern TEXT MODE ONLY

Format: CC: character-code,pattern-id

The CP: statement allows you to define the pattern of a character. The character with the ASCII code of character-code is given the pattern defined by pattern-id. To determine the value for pattern-id follow the instructions below:

The characters in pattern-id define the pattern of the character. Each character can be represented by an 8-by-8 grid. Each one of the 8 lines of the grid consist of eight dots, which are divided in two groups of four dots for the purpose of defining the character. First you must decide which dots are to be on (visible) and which are to be off (invisible). After this is done the pattern you created must be translated in to hexadecimal numbers so that it can be understood by the computer. The table below gives all the possible arrangements of the groups of four dots. First find the arrangement in the table that matches the first half of the first line of dots and write done the number or letter next to it. Repeat this for the second half of the first line.

Again do the second line like the first. You should now have a group of four numbers or letters, place a comma after this group. You have now translated the first two lines of the character pattern. Repeat the above procedure for the remaining 3 groups of two lines. When you are finished you should have 4 groups of numbers and/or letters. You need not place a comma after the last group. This is what you should place in the CC: statement for pattern-id.

Note: @ means dot on and a space means dot off.

1234	Code	1234	Code	1234	Code	1234	Code
	0	@	4	@	8	@@	C
@	1	@ @	5	@ @	9	@@ @	D
@	2	@@	6	@ @	A	@@@	E
@@	3	@@@	7	@ @@	B	@@@@	F

Example:

```
IT:
CP: 65,FFFF,FFFF,FFFF,FFFF
CP: 66,1898,FF3D,3C3C,E404
```

This code changes the letter A into a solid block, and the letter B into a little man.

CS: Compute String

Format: CS: $-variable <- $-expression

The CS: statement allows you to concatenate strings together and assign the result to a string variable. PILOT 99 currently only supports concatenation of string variables, string constants are not allowed and no string functions are supported. The preceding limitations are imposed by the size of the program, all available memory is used. If an improved version is developed all those that purchased the program will be notified.

The plus sign (+) is used as the concatenation operator. Parentheses can be used to alter the normal addition order, however they slow down the execution spend and are not really necessary since addition can be order [sic] easily.

Example:

```
A: $A
A: $B
CS: $C<-$A+$B
T: $C
```

Inputs two strings, concatenates them, and prints out the resulting string.

DC: Draw Circle GRAPHIC MODE ONLY

Format: DC: row,column,radius

The DC: statement draws a circle in the graphic part of the graphics mode screen. The circle is centered at the point given by row,column, and has a radius in pixels given by radius. Care must be taken to insure that no part of the circle extends beyond the graphic part of the screen.

Example:

```
IG:
DC: 100,100,10
```

Draws a circle with center at 100,100 and a radius of 10.

```
IG:
C: #N<-10
LP: 10
DC: 100,100,#N
C: #N<-#N-1
EL:
```

Like above but the continues to draw smaller circles inside each other, effectively filling in the outer circle.

DL: Draw Line GRAPHIC MODE ONLY

Format: DL: row1,column1,row2,column2

The DL: statement draws a line between the point given by row1 and column1 to and the point given by row2 and column2.

Example:

```
IG:
DL: 50,50,100,50
DL: 100,50,100,100
DL: 100,100,50,50
```

Draws a triangle on the screen.

DR: Draw Rectangle GRAPHIC MODE ONLY

Format: DR: row1,column1,row2,column2

The DR: statement draws a rectangular with the point given by row1 and column1 being the upper left hand corner and the point given by row2 and column2 being the lower right hand corner.

Example:

```
IG:
DR: 50,50,100,100
DR: 55,55,95,95
```

Draws two boxes, one inside the other.

E: End

Format: E:

The E: statement is used to end a subroutine or the entire program. If a E: statement is encountered and no subroutine call is in effect the program is halted and a message to that effect is displayed, otherwise program control returns to the statement immediately following the last U: statement executed.

Example:

```
U: *SUB
E:
*SUB
T: THIS WAS TYPED FROM A
T: SUBROUTINE
E:
```

Executes the subroutine *SUB and the ends the program.

EL: End Loop

Format: EL:

The EL: statement serves as the end of a LOOP structure in PILOT. The body of the LOOP is defined as all the statements between the last LP: statement and the current EL: statement. LOOPS maybe nested up to three deep.

Example:

```
C: #N<-1
LP: 10
T: #N
C: #N<-#N+1
EL:
```

This code uses a loop to print the numbers 1 through 10.

FB: Fire Button

The FB: statement checks the status of the specified joystick's fire button. If the fire button is pressed the YES-FLAG will be set to a true state and if the fire button is not pressed the YES-FLAG will be set to a false state.

PILOT 99 also maps out the keyboard to simulate the two joysticks. Joystick number one's fire button is the "Q" key, and joystick number two's fire button is the "Y" key.

Example:

```
FB: 1
TY: YES
```

Checks Joystick one fire button and prints YES if the fire button was pressed.

GP: Graphic Pattern

Format: GP: sprite-char-code,pattern-id

The GP: statement is used in defining the pattern of a sprite character, the characters used for sprites. The specified sprite character is given the pattern defined by pattern-id. For instructions on determining the value of pattern-id. please see the CP: statement. Sprite-char-code can range from 1 to 42. Please not that sprite characters are totally different from characters and can not be interchanged.

Example:

```
IT:
GP: 1,1898,FF3D,3C3C,E404
SP: 1,1
SC: 1,16
SL: 1,10,10
SM: 1,10,10
```

Creates a sprite in the shape of a man and sets it in motion. The sprite is white in color.

GC: Graphic Color

Format: GC: foreground-color,background-color

In graphics mode pixels must be given a color when they are drawn. Those pixels that are turned on or are visible are given the foreground-color while those that are turned off are given the background-color.

The foreground and background color codes and there corresponding colors are given below :

COLOR	*CODE*	*COLOR*	*CODE*
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

Example:

```
IG:
GC: 4,8
DB: 50,50,100,100
E:
```

Draws a Light Green box on the graphics screen.

HC: HChar TEXT MODE ONLY

Format: HC: row,column,character-code,repetition

The HC: statement performs the same function as TI BASIC's CALL HCHAR statement. The character with the ASCII value of character-code is placed at the character position given by row and column and repeated repetition number of times.

Row can range from 1 for the top of the screen to 24 for the bottom of the screen. Column can range from 1 for the left side of the screen to 32 for the right side of the screen. Repetition may range from 1 to 32767.

Example:

```
IT:
HC: 1,1,65,384
HC: 12,1,66,384
```

This short program fills the top half of the screen with "A"s and the bottom half with "B"s.

IG: Initialize Graphics

Format: IG:

The IG: statement is used to initialize graphics mode. This must be done before any graphic mode statements can be executed. In graphics mode the screen is divided into two parts. The top two-thirds of the screen (light color) is in bitmap mode, this allows very detailed graphics to be displayed. This area is the area effected by the graphic commands such as DC: The bottom third of the screen is in text or 32-column mode. This area is used for displaying any characters you print or have inputted. The dual nature of the graphic mode allows you to draw detailed graphics and ask questions about them in the normal size characters. The bottom half of the screen will scroll automatically without interfering with the graphic section.

Note: If you wish to clear the graphic screen of a drawing the only way to do so is to use the IG: statement, note however that the bottom section of the screen is cleared also.

Example:

```
IG:
DL: 50,50,100,100
T: Graphics Mode!
```

Initializes graphic mode and draws a line then types on the bottom half of the screen.

IT: Initialize Text Mode

Format: IT:

The IT: statement initializes the text mode screen. In this mode the screen is 24 rows by 32 columns, and can only display characters (standard or user defined). When the IT: statement is executed the screen is filled with blanks and the cursor is positioned in the upper left hand corner.

Example:

```
IT:
T: TEXT MODE !
```

Initializes text mode and prints a line.

J: Jump

Format: J: label

The J: statement allows you to control the execution of you PILOT 99 program. When a J: statement is performed the program file is searched for the label (for an explanation of labels see the LABEL section of chapter 2) and execution continues with the statement immediately following the label. This permits you to selectively execute statements. Please note the following: 1) if the label does not exist in the program an error results, 2) Duplicate labels should not be used in a program, although they will not cause an error message they will not allow the program to operate correctly.

There are three special labels that are predefined by the PILOT 99 system: @A, @M, @P. @A branches back to the last executed A: or AS: statement, remember that it is the last executed. The @M label branches to the next M: or MJ: statement. The @P label branches to the next PR: statement.

Example:

```
T: What is 2 + 2?
A:
M: 4, FOUR
JN: @A
```

Asks for an answer to the problem and then checks for a correct answer. If the answer is incorrect the last accept statement is branched to.

JM: Jump on Match

Format: JM: label[,label...]

The JM: statement is like TI BASIC's ON...GOTO.. statement. When a M: or MJ: statement is executed the PILOT 99 system keeps track of which MATCH-STRING was found to be identical to the ANSWER-BUFFER. When a JM: is executed a branch is executed to the LABEL in the same position of the list as the MATCH-STRING that was matched in the M: statement. For, example if the third MATCH-STRING was the one matched then the third label in the JM: statement will be branched to.

Note: IF NO MATCH WAS FOUND IN THE PRECEDING M: OR MJ: STATEMENT THEN THE JM: STATEMENT HAS NO EFFECT. THE NEXT STATEMENT WILL BE EXECUTED.

A MJ: statement should only be executed immediately after a M: or MJ: statement. Other statements can effect the label number used by the JM: statement, and therefore can cause unpredictable results. In addition there should be a LABEL for every MATCH-STRING.

Example:

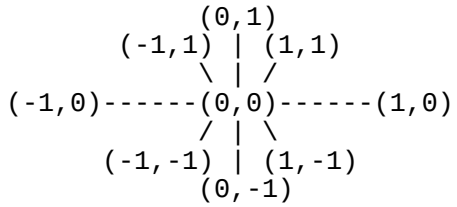
```
T:
A:
M: 4,FOUR
JM: *A<*B
E:
*A
T: YOU SAID 4
E:
*B
T: YOU SAID FOUR
E:
```

What is 2+2? Inputs an answer then jumps according to the answer given. A related message is printed depending on the answer.

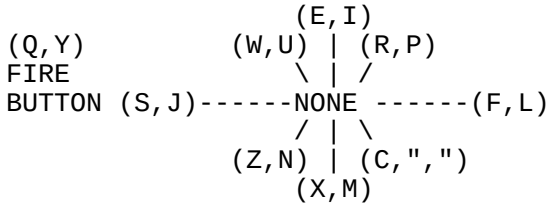
JS: JoyStick

Format: JS: joystick#,x-return,y-return

To sense the position of one of the joysticks use the JY: statement. The position of the joystick is returned in the x-return and y-return variables corresponding to the diagram below. The first value in the parenthesis is placed in x-return and the second in y-return.



A unique feature of PILOT 99 is that they joysticks are duplicated on the keyboard of the TI-99/4A, thus permitting use of joystick type input with out any additional equipment. The figure below gives the keys used in simulating the joysticks. The first key in the group is for joystick number one and the second for joystick number two.



Example:

```
IT:
C: #N<-12
C: #J<-16
HC: #N,#J,65,1
*JOY
JS: 1,#A,#B
J(#A=0 AND #B=0): *JOY
HC: #N+#A,#J+#B,65,1
E:
```

This short program displays an "A" in the middle of the screen and allows you to move it once by using the joystick.

LP: LooP

Format: LP: count

The LP: statement allows repeated execution of a section of code for a given number of times. The loop must always start with a LP: statement and end with a EL: statement, the body of the loop is consist of the statements between the LP: statement and the EL: statement. Loops may be nested three deep. The body of the loop is executed COUNT number of times. Your program should not branch out of the loop, if this happens unpredictable errors will result.

Example:

```
IT:
C: #R<-12
C: #C<-16
HC: #R,#C,65,1
*JOY
JS: 1,#A,#B
J(#A=0 AND #B=0): *JOY
HC: #R,#C,32,1
C: #R<-#R+#A
C: #C<-#C+#B
HC: #R,#C,65,1
EL:
E:
```

This program allows you to move the letter a around the screen up to ten times using joystick number one.

M: Match

Format: M: character-string[,character-string...]

The M: statement compares each character-string against the ANSWER BUFFER to see if the two match. If an exact match (including case) is found the YES FLAG is given a true state, else it is given a false state. This allows you with the use of the "Y" and "N" modifiers to perform actions depending on if a desired input was received.

Example:

```
T: INPUT A DIGIT?
AS:
M: 1,2,3,4,5,6,7,8,9,0
TN: YOU DIDN'T INPUT A DIGIT!
TY: THANK YOU!
E:
```

Asks for a digit and prints a message depending on if a digit was entered or not.

MJ: Match or Jump

Format: MJ: match-string[,match-string...]

The MJ: statement is a modified version of the M: statement. The statement works the same as the M: statement except that if no match is found it performs a branch to the next M: or MJ: statement. The MJ: statement is equivalent to the following PILOT 99 code: M: match -string. . . JN: @M

Example:

```
IT:
T: PRESS A KEY
AS:
MJ: 1,2,3,4,5,6,7,8,9,0
T: YOU PRESSED A DIGIT
J: *END
MJ: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
T: YOU PRESSED A LETTER
*END
E:
```

A short program that inputs a character and acts depending on the character.

OF: Open File

OF: file-name

The OF: statement is used to open the data file. File-name maybe any valid file-name for your system, opening a illegal file name will result in a file error and end the program. Any file you open should be able to take a RELATIVE, FIXED 80, INTERNAL, UPDATE type file. Only one data file can be opened at a time.

Example:

```
OF: DSK1.TEST
WR: TEST IT OUT
CF:
E:
```

This short program opens a data file and writes a record to it and then closes the data file.

PP: Plot Point GRAPHICS MODE ONLY

Format: PP: dot-row,dot-col

The PP: statement is used to turn a single pixel of the graphic screen on.

Example:

```
IG:
PP: 50,50
PP: 50,100
PP: 100,50
PP: 100,100
PP: 75,75
E:
```

This short program plots five points on the graphics screen.

PR: PRoblem

Format: PR:

The PR: statement performs no action, however it is used to easily mark the beginning of problem. The statement makes it easy to branch to the next problem in a program.

Example:

```
IT:
PR: PILOT
T: 2 + 2 IS?
A:
M: 4,FOUR
TY: CORRECT
JY: @P
T: IN CORRECT
J: @A
PR:
T: IT JUMPED TO THE NEXT
T: PROBLEM IF CORRECT
E:
```

Another short example shows the unique features of allowing the ease of programming problems.

R: Remark

Format: R: characters

The R: statement is totally ignored by the PILOT 99 system. This statement allows the programmer to place comments within his program. It is always a good idea to place comments within a program for documentation purposes.

Example:

```
R: THIS IS A COMMENT
```

RE: REad

Format: RE: [variable]

The RE: statement reads the next record from the data file and places it in the ANSWER BUFFER, the record can also be optionally placed in a variable. If a numeric variable is specified the record is converted. If a invalid representation of a numeric value is attempted to be converted it will cause an error.

Example:

```
IT:
OF: DSK1.DATA
WR: HELLO THERE
RF:
RE: A$
T: A$
CF:
E:
```

A simple program to open a file, write a record, and then read it back in. The record is printed to the screen.

RF: Restore File

Format: RF: [record#]

The RF: statement restores the data file to a specified record. If record# is omitted the data file is restored to record 0. The RF: statement allows you to access the data file in any order you wish. The maximum record# allowed is 32767.

Example:

```
IT:
OF: DSK1.DATA
WR: RECORD 1
WR: RECORD 2
WR: RECORD 3
RF: 2
RE: A$
T: A$
CF:
E:
```

First the program opens a data file and writes three records out to it. The file is then restored to the second record. The second record is read in and then typed on the screen.

S: Sound TEXT MODE ONLY

Format: S: duration,frequencey,volume,voice

The S: statement is used to create sound effects and/or music for PILOT 99 programs. The TI-99/4A has three voices all of which can be used in PILOT 99. Duration can be from 1 to 255, with the high numbers being of longer duration. Frequency can range from 110 to 32767. Voice can be either 1,2, or 3. Volume can be from 0 to 28, with 0 the loudest and 28 the softest.

Example:

```
IT:
S: 50,110,5,1
S: 50,220,5,1
E:
```

This short program creates two sounds.

SA: Sprites Atouch

Format: SA:

The SA: statement checks to see if any two sprites are in coincidence or touching each other. Unlike the SH: statement the SA: statement will only detect a coincidence when the visible part of two sprites are touching. If to sprites are touching then YES FLAG is set to a true state, else YES FLAG is set to a false state.

Whether or not a coincidence is detected depends on several variables. If the sprites are moving very quickly, the SA: statement may not be able to detect their coincidence. Also, SA: checks for a coincidence only when it is executed therefore it may miss a coincidence when another statement is being executed.

Example:

```
IT:
GP: 0,FFFF,FFFF,FFFF,FFFF
SC: 1,16
SC: 2,3
SC: 3,6
SP: 1,0
SP: 2,0
SP: 3,0
SL: 1,100,100
SL: 2,110,110
SL: 3,90,90
SM: 1,10,10
SM: 2,15,15
SM: 3,30,30
LP: 25
```

```
SA:
TY: SPRITE HIT
EL:
E:
```

This short example sets three sprites in motion and checks twenty-five times for a coincidence. If a coincidence occurs a message to that affect will be printed.

SC: Sprite Color

Format: SC: sprite#,color

The SP: statement is used to assign a color to a sprite. There are 32 sprites available, they are number 1 through 32. The color given to the sprite controls those pixels that are on or visible, a sprite always has a transparent as its background color. The following table shows the color codes for the sixteen available colors.

COLOR	*CODE*	*COLOR*	*CODE*
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

Example:

```
IT:
GP: 1,FFFF,FFFF,FFFF,FFFF
SL: 1,100,100
SM: 1,0,0
SP: 1,1
SS: 4
SC: 1,3
SC: 1,4
E:
```

This short program creates a sprite in the shape of a large block. The sprite is placed on the screen and with a color of medium green which is then changed to light green.

SD: Sprite Delete

Format: SD: Sprite#

The SD: statement is used to delete a sprite. The specified sprite is deleted, no other sprites are effected.

Example:

```
IT:
GP: 1,FFFF,FFFF,FFFF,FFFF
SP: 1,1
SC: 1,16
SL: 1,100,100
SM: 1,10,10
LP: 25
EL:
SD: 1
E:
```

Once again a sprite is created in the form of a block add set in motion. After a short while the sprite is deleted.

SG: Sprites Gone

Format: SG:

The SG: statement deletes all sprites.

Example:

```
IT:
GP: 1,FFFF,FFFF,FFFF,FFFF
SP: 1,1
SP: 2,1
SP: 3,1
SC: 1,3
SC: 2,6
SC: 3,16
SL: 1,100,100
SL: 2,150,50
SL: 3,50,200
SM: 1,10,10
SM: 2,10,0
LP: 25
EL:
SG:
E:
```

This longer program creates three sprites that are in the shape of blocks, and set the first two in motion. After a while all the sprites are deleted with the SG: statement.

SH: Sprite Hit

Format: SH: sprite#,sprite#

The SH: statement is used to detect a coincidence between two sprites. When the SH: statement is executed the two sprites are checked to see if their upper left hand corners are within 5 pixels. If a coincidence is detected the YES FLAG is given a true state, else it is given a false state. A coincidence can be detected even though the visible parts of the sprite are not visibly touching.

Whether or not a coincidence is detected depends on several variables. If the sprites are moving very quickly, the SH: statement may not be able to detect their coincidence. Also, the SH: statement only checks for a coincidence when it is being executed, so a program may miss a coincidence that occurs when another statement is being executed.

Example:

```
IT:
GP: 0,FFFF,FFFF,FFFF,FFFF
SC: 1,16
SC: 2,8
SP: 1,0
SP: 2,0
SL: 1,100,100
SL: 1,90,90
SM: 1,10,10
SM: 2,15,15
LP: 25
SH: 1,2
TY: SPRITES HIT
EL:
E:
```

This example sets two sprites in motion and checks twenty-five times for a coincidence between them. If the hit a message will be printed.

SL: Sprite Location

Format: SL: Sprite#,dot-row,dot-col

To specify the location of a sprite use the SL: statement. The specified sprite is moved so that its upper left hand corner is at the pixel given by dot-row and dot-col. Dot-row can range from 1 to 255, however the rows between 192 and 255 are off the bottom of the screen. Dot-col likewise can range from 1 to 255, are columns are visible on the screen.

Example:

```
IT:
GP: 1,FFFF,FFFF,FFFF,FFFF
SC: 1,16
SP: 1,1
SM: 1,0,0
SL: 1,100,100
LP: 25
EL:
SL: 1,50,50
E:
```

This short program creates a sprite in the shape of a white block and places it at pixel 100,100. After a short pause the sprite is moved to 50,50.

SM: Sprite Motion TEXT MODE ONLY

Format: SM: Sprite#,row-velocity,col-velocity

The SM: statement is used to set sprites in motion. Row-velocity and col-velocity may be from -128 to 127. For row-velocity a positive value is downward and a negative value is upward, for col-velocity a positive value is towards the right and a negative value is towards the left. There are 32 sprites available therefore sprite# must be from 1 to 32.

If both row-velocity and col-velocity are zero the sprite will be stationary at its present location. Row- and col-velocities near zero are slow, and like was those near -128 or 127 are fast.

Remember that the faster a sprite is moving the harder it is to detect a coincidence, and the more it appears to jump across the screen.

Example:

```
IT:
GP: 0,FFFF,FF00,00FF,FFFF
SC: 1,16
SP: 1,0
SL: 1,100,100
SM: 1,10,10
LP: 25
EL:
SM: 1,20,NEG(10)
LP: 30
EL:
E:
```

Sets a sprite in motion and after a short while changes its motion.

SN: ScreeN color

Format: SN: color

The SN: statement is used to change the color of the screen. Color must be have a value between 1 and 16, inclusive. For the corresponding color codes see either the CC:,GC:, or SC: statements.

Example:

```
IT:
SN: 15
LP: 25
EL:
E:
```

This short program changes the screen color to gray.

SP: Sprite Pattern

Format: SP: sprite#,sprite-character-code

The SP: statement is used to assign a pattern to a sprite. The sprite of number sprite# is given the pattern of sprite-character-code. The pattern for the sprite-character can be defined through the GP: statement.

If the current sprite size is 1 or 2 then the sprite pattern is defined by pattern of the sprite-character specified. If the sprite size is either 3 or 4 then the pattern of the sprite is defined by four sprite-characters. For a complete explanation please see the SS: statement.

Example:

```
IT:
GP: 0,FFFF,FFFF,FFFF,FFFF
GP: 1,F0F0,F0F0,F0F0,F0F0
GP: 2,00FF,00FF,00FF,00FF
GP: 3,AAAA,AAAA,AAAA,AAAA
SC: 1,16
SL: 1,100,100
LP: 5
SP: 1,0
SP: 1,1
SP: 1,2
SP: 1,3
EL:
E:
```

Creates a sprite and changes its pattern a number of times.

SS: Sprite Size

Format: SS: size

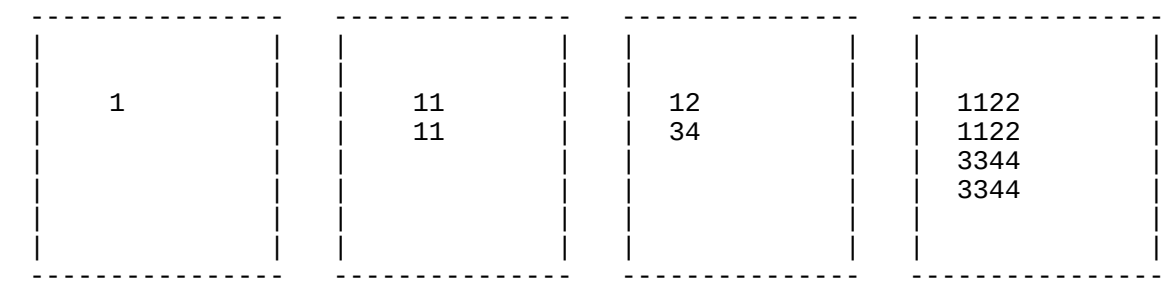
The SS: statement is used in controlling the size of all sprites. There are four sizes that sprites can take 1,2,3,4. The default size is 1. A complete explanation of each size follows.

A size of 1 causes sprites to be single size and unmagnified. This means that each sprite is defined only by the sprite-character specified with the SP: statement and take sub just one character position on the screen. (see figure 1)

A size of 2 causes sprites to be single size and magnified. This means that each sprite is defined only be the sprite-character specified with the SP: statement, but takes up four positions on the screen. (see figure 2)

A size of 3 causes sprites to be double size and unmagnified . Each sprite is defined by four character positions that include the sprite-character specified in the SP: statement. The first sprite-character used is the one specified in the SP: statement if it is evenly divisible by four, or the next smallest sprite character evenly divisible by four. That sprite-character is the upper left quarter, the next sprite-character is the upper right quarter, the next the lower left quarter, and the next the lower right quarter. The sprite takes up four character positions on the screen. (see figure 3)

A size of 4 causes all sprites to be double size and magnified. Each sprite is defined by four sprite-character positions as explained for size 3, and is magnified. The sprite takes up sixteen character positions. (see figure 4)



Example:

```
IT:
GP: 0,FFFF,FFFF,FFFF,FFFF
GP: 1,FF00,FF00,FF00,FF00
GP: 2,F0F0,F0F0,F0F0,F0F0
GP: 3,0000,FFFF,FFFF,0000
SP: 1,0
SC: 1,16
SL: 1,100,100
T: SPRITE SIZE=1
U: *PAUSE
T: SPRITE SIZE=2
SS: 2
U: *PAUSE
T: SPRITE SIZE=3
SS: 3
U: *PAUSE
T: SPRITE SIZE=4
SS: 4
E:
*PAUSE
LP: 5
EL:
E:
```

This program defines four sprite-characters. A sprite is defined giving it the pattern of sprite-character 0. The SS: command is then executed once for each of the four valid sizes. A short pause is provided between each size change.

T: Type

Format: T: characters

The T: statement types the characters onto the system monitor then performs a carriage return. After the T: statement the text cursor is on the first character of the next line. If the screen is full it is rolled up one line, and the top line of the screen is lost.

To print the value of a variable include the variable name in the normal manner within characters. To include a dollar sign ($) or a pound sign (#) place two of the symbols next to each other within characters. To include a character that can not be entered directly from the keyboard simply place the ASCII value of the character (can be an expression) between colons, for example :128: would print the ASCII character 128. To print a colon place two colons together within characters.

Example:

```
IT:
T: TYPING IN TEXT
T: MODE IS EASY!!
LP: 25
EL:
IG:
T: TYPING IN GRAPHICS
T: MODE IS JUST AS
T: EASY!!!!!!!!!!!!!
LP: 25
EL:
E:
```

A short example showing typing in the two screen modes.

TC: Text Cursor TEXT MODE ONLY

Format: TC: row,column

This statement moves the text cursor to the character position given by row and column. Row may be from 1 for the top of the screen to 24 for the bottom, and column can be from 1 for the left side of the screen to 32 for the right side of the screen. The TC: statement can be used in combination with the type statements to produce output anywhere on the screen. Please note that the TC: statement can be used in graphics mode put that the upper limit to row is changed to 5.

Example:

```
IT:
T: TYPE HERE
TC: 12,4
T: THEN HERE
TC: 4,2
T: NOW HERE
LP: 25
EL:
E:
```

A very short example showing the use of TC: with T:.

TG: Type Graphic

Format: TG: graph-row,graph-col,characters

The TG: statement makes it possible to place alphanumeric characters on the graphic part of the graphic mode screen. Only standard ASCII characters can be used in this statement, the value of variables can not be printed. For the purpose of the TG: statement the graphic part of the screen is divided into 16 rows of 64 characters.

Example:

```
IG:
T: REGULAR TYPING DOWN
T: HERE
TG: 1,5,GRAPHICS TYPING
TG: 2,5,UP HERE.
LP: 25
EL:
E:
```

A short example showing the difference between graphic typing and regular typing.

TH: Type and Hang

Format: TH: characters

The TH: statement operates very similar to the T: statement except it does not perform a carriage return. The text cursor is located in the next character position following the text printed. If necessary a screen scroll will be performed, thus losing the top line of text.

To print the value of a variable include the variable name in the normal manner within characters. To include a character that can not be directly inputted from the keyboard, simply places the numeric ASCII value of the character between colons. For example, : 128 : will print the character having a ASCII value of 128. To include a dollar sign ($), a pound sign (#), or a colon (:) place two of the symbols together within characters.

Example:

```
IT:
T: SEE THE CARRIAGE RETURN
TH: NO CARRIAGE RETURN
T:RUNS TOGETHER
LP: 25
EL:
E:
```

This short program shows the difference between the T: and TH: statements.

TP: Type to Printer

Format: TP: characters

The TP: statement is identical to the T: statement put instead of the output going to the system monitor it goes to the system printer. If your system does not have a system printer, the TP: statement will not perform any action.

To print the value of a variable include the name of the variable within characters in the normal manner. To print a character that can not be inputted directly from the keyboard, place the ASCII value of the character between colons. For example : 128 : would print the character with a ASCII value of 128. To print a dollar sign ($), a pound sign (#), or a colon (:) place two of the symbols together within characters.

Example:

```
TP: THIS IS PILOT
TP: TO PRINTER DO
TP: COPY???
E:
```

This PILOT program shows the ease of typing to the system printer.

U: User Subroutine

Format: U: label

The U: statement is like TI BASIC's GOSUB statement. The U: statement executes a user subroutine starting with the given label. The subroutine must end with a E: statement. Subroutines maybe nested up to three deep, that is subroutine A may call subroutine B which in turn can call subroutine C. After the subroutine is finished the control passes back to the statement after the subroutine call.

Example:

```
IT:
C: #N<-5
T: SUBROUTINE TO DO
T: SOME MATH WORK
T: AND PRINT IT!
U: *SUB
E:
*SUB
C: #A<-SQR(#N)
C: #B<-#N^2
T: N IS #N
T: SQR(N) IS #A
T: N^2 IS #B
E:
```

A simple use of a subroutine.

UP: Unplot Point GRAPHICS MODE ONLY

Format: UP: dot-row,dot-column

The UP: statement makes the pixel given by dot-row and dot-column to be invisible. The UP: statement allows you to selectively turn off pixels. Dot-row can range from 1 for the top of the screen to 127 for the bottom of the screen, and Dot-column can range from 1 for the left side to 255 for the right side of the screen. The pixel specified will be given the background color of the graphic colors in use.

VC: VChar TEXT MODE ONLY

Format: VC: row,column,character-code,repetition

VC: Displays the character with ASCII value of character-code in the position given by row and column and repeats it vertically repetition number of times.

A value of 1 for row is the top of the screen and 24 for row is the bottom of the screen. A value of 1 for column indicates the left side of the screen, and a value of 32 is the right side of the screen. Repetition can range from 1 to 32767.

Example:

```
IT:
VC: 1,1,65,24
VC: 12,2,66,43
E:
```

Just like HC: but the other way and this program shows it.

WA: Write Answer Buffer

Format: WA:

The WA: statement writes the ANSWER BUFFER out to the data file. This statement makes it easy to save to a data file the answers given.

Example:

```
IT:
T: WHAT IS YOUR NAME?
A:
OF: DSK1.TEST
WA:
CF:
E:
```

Inputs an answer and then writes it to a data file.

WH: WHile

Format: WH: expression

The WH: statement signifies the end of a WHILE loop. A while loop is executed as long as the expression in the WH: statement is in a true state. When the expression assumes a false state control is passed to the statement following the WH: statement. WHILE loops maybe nested up to three deep.

Example:

```
IT:
BW:
T: INPUT A NUMBER
AS: #A
T: NOT IT
WH: #A<>0
T: THAT'S IT
E:
```

Inputs a number until the number zero is inputted.

WR: WRite

Format: WR: characters

The WR: statement is similar in nature to the TP: statement except that it works with the data file instead of the system printer. The maximum number of characters that can be sent by a WR: statement is eighty (80).

To print the value of a variable include the name of the variable within characters in the normal manner. To include a dollar sign ($) or a pound sign (#) place two of the symbols together within characters.

Please note: That if a variable is not printed by itself, that is the only item for that WR: statement, it can not be retrieved as a variable. This is especially true for numeric variables.

Example:

```
OF: DSK1.TEST
WR: THIS WAS CREATED
WR: USING PILOT
C: #N<-5
WR: #N
CF:
```

This short example opens a file and writes two records then writes a record saving a numeric variable.

Disk 23. Games-n-Graphics

Version: *Author*: Jim Peterson, others
Requires: XB, EA *Language*: XB, AL *Updated*:

An arcade style assembly game King's Castle, a very complete XB version of Star Trek and several sprite and graphics demos from Jim Peterson of Tigercub.

```
dskdir. v2.0. 12-dec-96
Disk name               = GAMES
Sectors total           = 360
Sectors used            = 315
Sectors available       = 43
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CASTLEOBJ    39 DIS/FIX 80   >022 038
002 >003 COLUMBIA     14 PROGRAM      >048 013
003 >004 KINGCASTLE    2 PROGRAM      >055 001
004 >005 KINGOBJ      55 DIS/FIX 80   >056 054
005 >006 LOAD         14 PROGRAM      >08c 013
006 >007 RANSYMCHAR   11 PROGRAM      >099 010
007 >008 READ-ME      10 DIS/VAR 80   >0a3 009
008 >009 SPRITEDEMO   40 PROGRAM      >0ac 039
009 >00a TREK         89 INT/VAR254   >0d3 088
010 >00b TREKINS      41 PROGRAM      >12b 040
```

Disk 23. Contents of File READ-ME

Notes on the Boston Computer Society "Games and Graphics" Disk

All programs on this disk may be run by simply putting the disk in drive one and selecting Extended Basic. The LOAD program will present a menu of programs. There is no program on the disk that can not be accessed via the LOAD program.

The Programs:

LOAD

> This program comes from Jim Peterson of Tigercub. It allows you to print a catalog of the disk, delete files, and run Extended Basic programs at a key press.

COLUMBIA
RANSYMCHAR
SPRITEDEMO

> These three programs are also from Jim Peterson. They are excellent graphics demos. Columbia has full sound with graphics, Spritedemo has a demonstrates the vast array of things one can do with sprites, and Ransymchar generates random symmetric characters. All of Mr. Peterson's programs are unprotected and are well worth looking at as they are excellent examples of programming. Mr. Peterson has over 130 programs available for $3 each. You can obtain (and I recommend that you do so, — jph) a complete catalog by sending $1 (refundable on your first purchase) to Tigercub Software, 156 Collingwood Ave., Columbus, OH 43213.

TREK
TREKINS

> Trek is a fantastically complete version of the game Star Trek written in Extended Basic. For instructions run the program TREKINS which will present complete instructions in a very clear format by showing you graphics and displays from the actual game.

KINGCASTLE

> This program is the loader for the two assembly files KINGOBJ and CASTLEOBJ. It is a fast paced assembly game called King of the Castle and requires the use of a joystick. It was released into the public domain by Cydex and comes to us through multiple sources.

Disk 24. Disk Utilities

Version:
Author: Todd Kaplan, others
Requires: EA
Language: AL
Updated:

DISK02 — now includes disk sector editor with printer support and new features.
CATLIB — keeps catalog of all programs on your disks. Disk Master — another disk manager program with some unique features and exceptionally clear menus, actually early version of a commercial release.

```
dskdir. v2.0. 12-dec-96
Disk name               = DISK-UTILS
Sectors total           = 360
Sectors used            = 269
Sectors available       = 89
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >006 CATLIB      110 DIS/FIX 80 Y >091 063 >0f9 046
002 >005 CATLIB/DOC   42 DIS/VAR 80 Y >068 041
003 >002 D-UTIL1      33 PROGRAM    Y >022 032
004 >003 D-UTIL2      33 PROGRAM    Y >042 032
005 >004 D-UTIL3       7 PROGRAM    Y >062 006
006 >007 DISKO2       25 PROGRAM    Y >0d0 024
007 >008 DISKO2DOC    15 DIS/VAR 80 Y >0e8 014
008 >009 READ-ME       4 DIS/VAR 80   >0f6 003
```

Disk 24. Contents of File CATLIB/DOC

CATALOGING LIBRARY. Version 1.4.0

A Fairware program by Marty Kroll Jr.

FEATURES

1. Catalogs up to 123 disks and 900 files on each set of data files.

2. Allows storage of multiple data files on one data disk. In fact, one bare DSSD disk holds more than 5 full sets of data, enabling you to store more than 5000 programs and 600 disks on one data disk.

3. Saves data for later listings, additions, or deletions.

4. Load and/or reload data files without rebooting program.

4. Works on single or multiple disk systems.

5. For single disk systems there is no need to continually switch disks until all deletions and additions are made.

6. For multiple disk systems you can catalog a disk from any drive.

7. When adding disks, catalog is listed on the screen. You have the option of adding the disk or not.

8. When adding disks, you are informed if the diskname is already on file. If it is, you can either replace the old listing with the new (in the case of an update, etc.), or you can give the new listing a temporary name (as in the case of a backup disk, etc).

9. Catalogs these "funny sectored" disks:
   Those that appear not initialized
   Those that appear empty because sector 1 has been altered

10. Eliminates all non-printable characters from file and disk names, replacing them with a period, since no legal name contains a period. This eliminates sending unwanted control codes to your printer.

11. Print a standard format catalog of any disk on file in your library, including the funny sectored disks.

12. Outputs the following to screen or printer:
    Summary of disks
    Complete listing of files
    Conventional catalog listing of any disk on file

13. Output complete catalog of all disks, disk by disk, either for all disks or selectively, to the printer.

14. Chose 1,2, or 3 columns for printer outputs.

15. 100% assembly language = fast sorts.

CHANGING DEFAULTS

Using a disk sector editor, such as Diskfixer, Disko, or Disk+aid, read the first sector of the object code for the Cataloging Library program.

The first bytes of the program should contain these values:

Bytes 0000-000F
0100 004D 2E20 4B52 4F4C 4C39 2676 4207

Bytes 0010-001F
1742 07F6 4200 0242 1D20 4220 2042 1E20

Bytes 0020-002F
4220 2042 0012 4210 6042 5050 4200 0042

Bytes 0030-003F
0003 4250 4942 4F20 4220 2042 2020 4220

Bytes 0040-004F
2042 2020 4220 2042 2020 4220 2046 2020

Bytes 00B0-00BF
0032 4204 E042 3654 4204 E042 3656 4202

These bytes on this sector contain the following information, which can be changed with the sector editor:

FOREGROUND/BACKGROUND COLORS:
Byte 0010
Default = 17 = black on Cyan

FOREGROUND/BACKGROUND COLORS OF ERROR SUBROUTINE:
Byte 0013
Default = F6 = white on dark red

NUMBER OF COLUMNS FOR PRINTOUT
Byte 0016 = 03
Default = 3

CONTROL CODES FOR CONDENSED PRINT:
Bytes 0018,0019,001B,001C
Epson = 0F 20 20 20 = CHR$(15) & 3 spaces
Okidata = 1D 20 20 20 = CHR$(29) & 3 spaces

Bytes 79,7B,7F,81,85,87,8B,8D
Epson = 30 46 32 30 32 30 32 30
Okidata = 31 44 32 30 32 30 32 30

CONTROL CODES FOR NORMAL PRINT:
Bytes 001E,001F,0021,0022
Epson = 12 20 20 20 = CHR$(18) & 3 spaces
Okidata = 1E 20 20 20 = CHR$(30) & 3 spaces

Bytes 91,93,97,99,A5,A7,AB,AD
Epson = 31 32 32 30 32 30 32 30
Okidata = 31 45 32 30 32 30 32 30

FLAG FOR CONDENSED PRINT:
Byte 0075,0076
Default = FF FF = condensed print
00 00 = normal print

OUTPUT DEVICE NAME
Bytes 0033,0034,0036,0037,0039,003A
003C,003D,003F,0040,0042,0043
and continue thru byte 0073
Default = 50 49 4F 20 20 20 20
= 'PIO '

OF PROGRAMS PRINTED PER COLUMN PER PAGE
Byte 00B0,00B1
Default = 0032 = >32 or decimal 50
actual files printed per page (not including heading and title)

LOADING

Select Load and Run from Editor/Assembler or Mini-Memory. Enter object code name DSK1.CATLIB/EPS (or DSK1.CATLIB/OKI.). Program Auto Starts.

After viewing title screen, press **ENTER**. You will be asked to enter the name for the files to read/store the file and disk data. Enter a 2-digit decimal number in the range of 00 to 99. You will enter the number for the filedata name, and the same number will be included in the diskdata name.

The first time you select the 2 filenames for data, 2 files will be opened on DSK1. Thereafter, each time you select the same file names for data, the 2 files on DSK1 will be read into memory.

Note: This selection is made upon entry into the program, and can also be made thru menu selection anytime while running the program.

Note: To get the maximum number of data files on a disk, either use a blank data disk, or make a duplicate master disk and delete all versions of this program you will not need.

HINT: Divide your disk library into categories, such as games, utilities, etc. Use one set of data files for each category, such as 01 for games, 02 for utilities. These numbers and categories can then be included when entering the date, giving a small meaningful title to your printout, such as: 01*GAMES 12-15-85.

FUNCTIONS

ADDING DISKS

After listing disk and contents on screen choose to add the disk or not, Y or N. If you pick Y, program searches for identical diskname already on catalog. If match is found you can: Replace old listing with new updated version. Give new listing temporary name. A temporary name replaces the 6th character with a small "t", making it easy to recognize which disk it corresponds to. In addition, this allows it to be near the original disk when alphabetized.

DELETE DISKS

You can delete a disk from the listing. You will be asked for diskname. Type filename only, NOT DSK1. If disk is in catalog you will be notified of its deletion. If it is not in catalog, you will be told this also.

ADDITIONS AND DELETIONS

If you have made additions and/or deletions, have not sorted the data, and choose another menu choice, the program will sort data and save on disk in drive #1. Thereafter, you menu choice will completed.

If you attempt to add a disk that will take the totals over the maximum limit you will be given the catalog filled message. If you have deleted some disks from the catalog and have not yet sorted them, select option "M" to sort. This may free some room for more additions.

PRINTING CATALOG DISK BY DISK

After picking this menu choice ("G"), you have the option of printing all disks or selective disks. If you choose selective, the disknames are displayed on the screen, and you are prompted as to where you want to print it or not.

Whether you choose printing all disks, or selective disks, the date and page number are centered on each page. The diskname, sectors used and free are printed next. The programs on the disk are printed, in the number of columns you selected with the printing option menu. If you previously selected 3 columns, the printout will actually be made in four columns, to balance out the paper.

Before another disk is printed, the program determines whether the listing will fit on the page. If not, the paper is advanced, then the disk and its contents are printed. These eliminates a disk and contents being split between 2 pages.

CHANGE PRINTER OPTIONS

Use this routine to print control codes (ex. condensed print for 3 output columns). If you want to send control codes, enter "Y" when asked if using condensed print. Then enter up to 4 HEX control codes. If you don't need to use all 4 values, just enter >20 for any entries not required.

Next enter up to 4 hex codes to return the printer to normal printing, again entering >20 for values that are not required on your printer.

This option also allows you to choose your output file name (PIO, DSK1.TEST, etc.)

This option also lets you select the number of columns for your output.

In addition, you can also enter the current date with this option.

HALTING OUTPUT

FCTN 4 halts all output to screen or printer.

OBTAINING THE PROGRAM

You may obtain a copy by sending $3.50 for disk and postage to:

Marty Kroll
218 Kaplan Ave.
Pittsburgh, PA 15227

Reminder: If you use this program, please send $10.00 to the author named above. Any comments are appreciated.

Disk 24. Contents of File DISKO2DOC

DISK UTILITITES (version 1.1)

by JOHN BIRDWELL

PURPOSE

These utilities are intended to allow you to learn more about the structure of the data on your diskettes and to enable you to make whatever changes to the data that you may need.

UTILITIES

Compare Disks
: This utility allows you to compare 2 diskettes to see if the contents of both are identical. Sectors which do not match are printer on you printer.

Print Sectors
: This utility allows you to print single or multiple sectors to your printer.

Sector Editor
: This utility allows you to edit the contents of a sector, in ASCII or hex, and write it back to the same or a different disk/sector.

Find String
: This utility will find a string of up to 10 bytes, ASCII or hex, on your diskette. If the string is found it will be displayed and you will enter the Sector Editor. *Note*: Spaces are not allowed in the string.

Disk Report
: This utility will printout the disk name, total number sectors, the files, their size, what sector they start and end at, and if they are protected.

Edit Keys (All modes except Sector Editor)

FCTN 8	Returns to the Menu
FCTN 4	Stop printing and return to Menu.
FCTN +	Quit Utilities
FCTN D	Moves cursor left
FCTN S	Moves cursor right
ENTER	Terminates input

Edit Keys — Sector Editor

CTRL N	Reads (N)ext sector
CTRL B	Reads (B)ack a sector
CTRL W	(W)rites a sector
CTRL R	(R)eads a new drive/sector
CTRL P	(P)rints a copy of the screen on your printer.
CTRL D	(D)one with Editor, returns to Menu
CTRL A	Edit in (A)SCII
CTRL H	Edit in (H)ex
FCTN D	Forward space
FCTN S	Backspace
FCTN E	Up line
FCTN X	Down line

STARTUP

Load this program via option 5 of the Editor/Assembler. Upon completion of the load you will be asked your printer type, PIO, is the default. This is the only time this is required. From this point on it should be self-explanatory.

POSSIBLE CHANGES

If you do not have a PIO type printer you can use the FIND STRING utility to locate where this string is at in the program. After finding it you can change the default to reflect your printer type (I have left room for up to 28 characters for this purpose).

An additional change most of you will need to make is for the control characters which place your printer into condensed mode and into normal mode. I have an Okidata printer and it uses a Hex '1D' to go into compressed mode and a Hex '1E' to go back into normal mode. I have allowed for 2 characters for this control sequence. These words are located after the spaces for the PIO description. Their format is 1D00 followed by 1E00. You can use the FIND STRING to locate them.

This should give you enough information to get started. Use caution with the Editor as you can really mess yourself up with it. Please notify me through any of the Chicago TI-BBSs of any bugs or changes you think might be good.

— JOHN BIRDWELL

Disk 25. PR Base — Disk 1

Version: 2.0 *Author*: William Warren
Requires: EA, XB *Language*: AL *Updated: 08/20/86*

Complete data base written in assembly. Allows design of both input and output format. Widely praised. Full disk of documentation. Certainly the best fairware database and among the best available for the 99/4A. A TI Classic.

```
dskdir. v2.0. 12-dec-96
Disk name               = PRBASE
Sectors total           = 360
Sectors used            = 338
Sectors available       = 20
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHAR          6 PROGRAM    Y >020 005
002 >003 COPY         11 DIS/FIX 80 Y >025 010
003 >004 CRT:1        33 PROGRAM    Y >02f 032
004 >005 CRT:2        33 PROGRAM    Y >04f 032
005 >006 CRT:3         7 PROGRAM    Y >06f 006
006 >007 LABEL:USR    17 PROGRAM    Y >075 016
007 >008 LOAD         15 PROGRAM    Y >085 014
008 >009 MC3          29 PROGRAM    Y >093 028
009 >00a PRB:1        33 PROGRAM    Y >0af 032
010 >00b PRB:2        33 PROGRAM    Y >0cf 032
011 >00c PRB:3        16 PROGRAM    Y >0ef 015
012 >00d PRBASE       23 DIS/FIX 80 Y >0fe 022
013 >00e PRBUTL/BAS   46 PROGRAM    Y >114 045
014 >00f PRBUTL/DOC   24 DIS/VAR 80 Y >141 023
015 >010 PRINTFILE     4 PROGRAM    Y >158 003
016 >011 UTIL1         8 PROGRAM    Y >15b 007
```

Disk 25. Contents of file PRBUTL/DOC

[See The Cyc, Appendix, *Genial TRAVelER* for the contents of this file — Ed.]

Disk 26. PR Base — Disk 2

Version: 2.0 *Author*:
Requires: *Language*: *Updated*: 08/20/86

See disk 25.

```
dskdir. v2.0. 12-dec-96
Disk name               = PRBASE2/0B
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 PRB:DOC     357 DIS/VAR 80 Y >020 328 >003 028
```

Disk 26. Contents of file PRB:DOC

[See The Cyc, Appendix, *Genial TRAVelER* for the contents of this file — Ed.]

Disk 27. C99

Version: 2.1 *Author*: Clint Pulley
Requires: EA *Language*: *Updated*: 12/12/86

Implementation of the C programming language for the TI. Very usable. While this is not a full Implementation of C, it contains most C constructs except records, unions, and floating point. The first real compiled language for the 99/4A. Also requires #45.

```
dskdir. v2.0. 12-dec-96
Disk name               = C99REL4A
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README1      9 DIS/VAR 80   >022 008
002 >003 C99C         33 PROGRAM      >02a 032
003 >004 C99D         33 PROGRAM      >04a 032
004 >005 C99E         33 PROGRAM      >06a 032
005 >006 C99MAN1      45 DIS/VAR 80   >08a 044
006 >007 C99MAN2      48 DIS/VAR 80   >0b6 047
007 >008 C99MAN3      36 DIS/VAR 80   >0e5 035
008 >009 C99SPECS     39 DIS/VAR 80   >108 038
009 >00d CFIO         11 DIS/FIX 80   >155 010
010 >00e CONIO         2 DIS/VAR 80   >15f 001
011 >00a CSUP         14 DIS/FIX 80   >12e 013
012 >00f FPRINTF       5 DIS/FIX 80   >160 004
013 >010 FSCANF        6 DIS/FIX 80   >164 004 >011 001
014 >012 PRINTDOC      2 DIS/VAR 80   >013 001
015 >00b PRINTF       13 DIS/FIX 80   >13b 012
016 >00c SCANF        15 DIS/FIX 80   >147 014
017 >014 SPRINTF       5 DIS/FIX 80   >015 004
018 >019 SSCANF        5 DIS/FIX 80   >01a 004
019 >01e STDIO         3 DIS/VAR 80   >01f 002
```

Disk 27. Contents of file -README1

C99REL4A Release Notes 88/01/07

by Clint Pulley

This is the fourth release of c99 with improvements and new features. If you have an earlier version of c99, please check the documentation carefully. This disk, C99REL4A, includes version 4.0 of the c99 compiler, console and file I/O libraries and functions, and documentation.

The compiler is based on Ron Cain's original small-c as revised by Jeff Lomicka. Considerable insight and many improvements resulted from analyzing Jim Hendrix's small-c version 2.

Contents of the C99REL4A diskette:

C99C/D/E	The c99 compiler pgm files
C99MAN1/2/3	The c99 user's manual
C99SPECS	c99 specifications
CFIO	Obj file I/O library
CONIO	Include file (cf manual)
CSUP	Obj support and console I/O library
PRINTDOC	Batch file to print docs
STDIO	Include file (cf manual)
PRINTF	Object files for the
FPRINTF	Standard C formatted
SPRINTF	Output functions
SCANF	Object files for the
FSCANF	Standard C formatted
SSCANF	Input functions

All documentation files were written and saved with TI-Writer. They are set up for printing with RUNOFF1. To use this program, select the E/A Load Program File option. Place disk C99REL4B in drive 1 and type DSK1.RUNOFF1. The program will load and start. It will ask for an output filename. Enter the device name for your printer (i.e. PIO). Reply **N** to the keyboard input prompt. It will then ask for an input filename. Place this disk (C99REL4A) in the drive. Enter DSK1.PRINTDOC. When printing is complete, it will again ask for an input file. Press **ENTER** and respond with **N** to the rerun query.

The file FMTIODOC on C99REL2B provides documentation for printf and scanf.

Clint

Disk 27. Contents of file C99MAN1

c99 User's Manual Edition 4.0: 88/01/01

Software and Documentation written and adapted by Clint Pulley

User-supported software (FAIRWARE)

The c99 compiler, libraries, associated software and documentation are provided for your use and that of your friends and colleagues.

You are encouraged to distribute c99 freely provided you charge no more than media and reasonable distribution costs. All copies of the release diskettes must include this disclaimer.

If you are using c99 and find it of value, your donation ($20.00 suggested) to the author will be appreciated and will help support further development of this product. Contributing users will be placed on a mailing list and advised of new releases.

If you develop useful applications using c99 you may market them provided that the software and documentation acknowledge the use of c99. The author would appreciate receiving a complimentary copy of any such programs.

Please address all correspondence to:

Clint Pulley
38 Townsend Avenue
Burlington, Ontario
Canada L7T 1Y6
(416) 639-0583 (home)
Source TI7395
CompuServe 73247,3245
GEnie C.PULLEY

Requests for copies of the c99 release diskettes should include two formatted diskettes (SSSD please) in a mailer and $1.00 for return postage. If you already have a version of c99, please indicate the version number in your letter.

This software carries no warranty , either written or implied, regarding its performance or suitability. Neither the author nor any subsequent distributor accepts any responsibility for losses which might derive from its use.

I. INTRODUCTION

c99 is based on small-c version 1 which was published by Ron Cain in *Dr. Dobb's Journal* No. 45, May 1980. Many enhancements were adapted from Jim Hendrix's small-c version 2. Small-c is a useful subset of the C programming language. The compiler produces assembler source code as its output. This code is assembled to produce an object file which is loaded together with all required libraries and run.

c99 was designed to run in the Editor/Assembler (cartridge) environment on the TI-99/4A computer. Since a number of Extended Basic loadable Editor/Assembler "simulators" are now available, this version of c99 has been modified to be useable with these products. The compiler and generated programs have been run successfully with Funlwriter V 3.0 and BEAXS.

c99 has these features:

- It supports a subset of the C language.
- Most of the compiler is coded in c99.
- It is syntactically identical to standard C.
- It produces assembler source code rather than an object file.
- It is a stand-alone single pass compiler. It does its own syntax checking and parsing.
- It can compile itself.

Although c99 lacks many of the features of standard C systems and produces code which is less than optimal, it is, in the opinion of the author, a worthwhile addition to the software repertoire of the TI-99/4A computer. For the first time a structured language with a true compiler is available to TI users. c99 is sufficiently powerful for the development of utilities, text processors, database systems and games. Since the introduction of c99 over two years ago, many useful programs have been written by users of this language.

Extensive testing of version 4.0 by the author and several advanced c99 users has not revealed any glaring errors. If you find bugs or wish to suggest improvements, please drop the author a line or leave a message on The Source (TI7395), CompuServe (73247,3245), or GEnie (C.Pulley).

c99 was developed on a 1983 (black) TI-99/4A with 32K memory expansion, TI (later Myarc) disk controller, two SSSD drives, RS232 interface and a Roland printer. The final changes for version 4 were developed on a Myarc 9640 computer in 99/4A emulation mode. The Editor/Assembler software environment was used for all software development. All code was written or adapted by Clint Pulley.

This manual assumes a knowledge of standard C or the availability of a suitable reference. The file C99SPECS, found on the release diskette, identifies the features of C which are available in c99.

II. USING c99

A. Entering the source program

Input to the compiler must be a Display/ Variable 80 disk file or (for test purposes) the console keyboard. The standard TI editor is normally used for this purpose. If TI-Writer is preferred, be sure to save the source program with the Print File option to avoid getting a line of tab data at the end of the file.

B. Compiling the source program

From the Editor/Assembler menu, select the Load Program File option. Enter DSKn.C99C, where n is the diskette drive containing the compiler disk. When the compiler has been loaded it will identify itself and ask about a number of options. The questions asked are:

```
Include c-text? [n]
```

This provides the option of including the c99 source code as comments in the output file. If your response is **y** or **Y** each line of c source will appear in the output file preceded by an asterisk. This option should not be selected for large programs as it will result in a very large output file. The default response (**n**) will result from pressing the **ENTER** key.

```
Inline push code? [n]
```

The c99 compiler normally generates a subroutine branch to push a value onto its stack. This results in fewer instructions being generated, but execution speed suffers slightly. If maximum performance is required, reply with **y** or **Y**. This will cause the compiler to generate the two instruction push sequence inline. The compiler will then prompt for the input and output filenames. In each case, respond with the full filename (in upper or lower case). If c99 is unable to open a file it will display "Bad filename-try again" and prompt for another name. If the response is a null name (pressing enter only) that file will map to the keyboard or screen. This may be useful for providing a quick check of a short program. Screen output will pause when a key is pressed.

The compiler will now proceed to process the source program. As each function header is encountered, the first six characters of the function name are displayed on the screen. If it is desired to abort execution at any time, press **CLEAR (FCTN 4)**. This will terminate the compiler and close all files.

If the compiler encounters an error in your program, it will display an error message on the screen and pause. After noting the error, press enter to resume compilation.

When the compiler has finished, it will display the number of source lines processed, symbol table usage, and the number of errors encountered. It will then ask if more compilations are to be done. A response of **y** or **Y** will restart the compiler, a response of **n** or **N** will exit. If the Editor/ Assembler cartridge is being used, exit will be to the E/A screen. Otherwise, exit is to the power-up screen.

C. Assembling the compiler output

The output file should be assembled using the TI Assembler (option 2 from the E/A menu). The R option is not required since c99 generates numeric register references. This reduces the size of the output file and speeds the assembly slightly.

D. Loading the program and libraries

The object file from the assembly may be loaded using the Load and Run option from the E/A menu. After loading the object file the CSUP library must always be loaded. If the program uses file I/O and contains the statement:

```
#include dsk1.stdio
```

then the CFIO library must also be loaded. If other external functions or libraries have been referenced by your program they must also be loaded at this time.

E. Running the loaded program

When the last file has been loaded, press enter to display the program name prompt. All c99 programs begin execution at the entry point START. When this has been entered, your program will (hopefully) execute correctly. At program exit the message "c99exit-Rerun? (Y/N)" may be displayed. This ensures that the screen will not be cleared at the instant the program stops and provides the option to rerun the program. Press **y** or **Y** to rerun the program, **n** or **N** to exit to E/A.

III. ERROR MESSAGES

When c99 detects an error in the source program it displays the error on screen and pauses. The error display is of the form:

```
ERROR : description
line of source code
^_ (pointer to approximate location of error)
```

In most cases the description, although brief, is self-explanatory. Among the more cryptic are:

`NOT AN LVAL`
In C jargon, an lvalue is an expression which may be assigned a value. If a and b are variables, a is an lvalue, a+b is not.

`TABLE OVERFLOW`
The c99 compiler contains a number of tables. The capacity of these tables is specified in C99SPECS.

`OUT OF CONTEXT`
Some keywords such as break and case can only be used within loops or switches.

`LINE TOO LONG`
A line in your program may not exceed 80 characters after macro substitution.

After noting the error, press enter to resume compilation. If 40 errors are encountered, compilation is terminated.

Notes:

a. The source line displayed has been pre-processed, so all multiple spaces have been removed, all names have been truncated to six characters, and all macro substitutions have been performed.

b. The error handling in c99 has been designed to minimize the number of spurious error messages generated. This has resulted in one shortcoming — if a statement contains more than one real error, only the first is reported.

Disk 27. Contents of file C99MAN2

IV. LIBRARIES AND INCLUDE FILES

The object libraries provided with this release are:

a. CSUP
The compiler support library. This contains the initialization, exit, and direct support (C$) functions required by all c99 programs as well as console I/O functions.

b. CFIO
The file input/output library. This contains the file tables and all file I/O functions.

The include files provided with this release are:

a.CONIO
I/O definitions for console functions only.

b.STDIO
I/O definitions for console and file functions as well as extern specifiers for all functions in CFIO. If STDIO is included in a program, CONIO must not be included or duplicate definitions will result.

V. LIBRARY FUNCTIONS

Each function is introduced by a sample call. If a function returns a value, an assignment is shown. You may, of course, discard the function result. Arguments must be of the same type as the sample.

The following declarations specify the type of all variables and arguments used in the sample calls.

```
int b,c,f,row,col,key,unit,len,recno;
char buff[81];
char *filename,*mode,*name,*string;
```

Functions in CSUP:

- Read one character from the keyboard.

```
c=getchar();
```

Waits for a key to be pressed and returns the character value. The character is echoed to the screen. If the character is **ENTER**, the screen spaces to the start of a new line and a value of 10 (EOL) is returned. If the character is **CTRL Z**, -1 (EOF) is returned.

- Write one character to the screen.

```
c=putchar(c);
```

Writes the character whose ASCII value is c to the screen. If c== 10 (EOL), the screen spaces to the start of a new line. If c==8 (BS), the cursor is backspaced. If c==12 (FF),the screen is cleared and the cursor is homed. If c represents a non-printing character, a \ is echoed. The value of c is returned.

- Read a line from the keyboard.

```
c=gets(buff);
```

Reads one line from the keyboard into a character array. The line is terminated with **ENTER**, **CTRL Z** or the 80th character. The array is assumed to be 81 characters long and a null (0) character is appended to the end of the string. A value of buff is returned unless **CTRL Z** is pressed. In that case, 0 (NULL) is returned. Use of the backspace key (**FCTN S**) for inline editing is supported.

- Write a string to the screen.

```
puts(string);
```

Writes a string to the screen, stopping when it finds a null character. The null character is not written. The cursor is not spaced to the start of a new line unless newline (\n) is encountered.

- Exit the program.

```
exit(c);    or     abort(c);
```

Branches to the c99 exit function which closes any open files. The value of c may be between 0 and 7. If c==0, the normal exit message is displayed. If c==7, the program terminates immediately. Otherwise the value of c is displayed in an error message. The exit(0) function is also executed when function main terminates normally.

- Locate the cursor on the screen.

```
locate(row,col);
```

Places the cursor at the screen location specified by row and col. Subsequent console I/O will start at the new cursor location. Row and column numbering start at 1 as in TI Basic. The validity of row and col is not checked.

- Check keyboard status.

```
key=poll(c);
```

Scans the keyboard and returns the key value (if one is pressed) or 0. If c != 0, the program will pause while a key is down. If c != 0 and **CLEAR (FCTN 4)** is pressed, the program will branch to the c99 exit function.

- Change screen color.

```
tscrn(f,b);
```

Changes the text mode screen colors to f (foreground) and b (background). The Basic color number convention is used.

Functions in CFIO:

Note: In most of the file I/O functions which reference the argument "unit", the operation will default to the corresponding console I/O function if the value of unit is <= 0. For this reason, the values for stdin, stdout, and stderr as defined in STDIO are -1, -2 and -3.

- Open a file.

```
unit=fopen(name,mode);
```

Opens the named file in the specified mode. Both name and mode must be strings or pointers to strings. Currently supported modes are:

display/variable	*display/fixed*	*display/relative*
"r" - read	"R" - read	"I" - read
"w" - write	"W" - write	"O" - read/write
"u" - update	"U" - update	
"a" - append		

In the mode parameter string, the mode character may be followed by a 1-3 digit record length. If this is omitted, a default length of 80 is assigned. If the file is opened for input and a record length of zero is specified, the actual record length of the existing file is utilized. The function ferrc(unit) can be used to obtain the record length.

A unit number is returned for use with the file I/O functions. This unit number must not be altered. If the open fails, NULL (0) is returned. No more than four files may be open simultaneously and no more than three may be disk files. Filenames may be upper or lower case and must not exceed 26 characters in length.

- Close a file.

```
c=fclose(unit);
```

Performs the appropriate file closing action and makes the unit available for another file. In the case of output files being written with putc, an incomplete line is lost. This function returns NULL if the close fails and non-null if it succeeds. All open files are closed automatically if a program terminates normally.

- Read one character from a file.

```
c=getc(unit);
```

Reads and returns the next character from the file corresponding to unit. If the end-of-line is reached a value of 10 (EOL) is returned. If the end-of-file is reached, a value of -1 (EOF) is returned. If an error occurs, -2 (ERR) is returned.

- Write one character to a file.

```
c=putc(c,unit);
```

Writes the character whose ASCII value is c to the file. If c==10 (EOL), the actual write operation occurs. The value of c is returned. If an error occurs, -2 (err) is returned.

- Read a string from a file.

```
c=fgets(buff,col,unit);
```

Reads one line from the file into a character array. At most, col-1 characters will be transferred. A null character is appended to the end of the line. If a partial line is transferred, the remainder of the line is discarded. If unit<=0, gets is called and the value of col is ignored. This could result in buffer overflow. A value of buff is returned. If an end-of-file or error condition occurs, NULL is returned.

- Write a string to a file.

```
c=fputs(string,unit);
```

Writes a string to a file, stopping when it finds a null character. Imbedded EOL characters act as record terminators, so multiple records can be generated by a single call to fputs. A value of buff is returned. On end-of-file or error, NULL is returned.

- Read a record from a file.

```
c=fread(buff,len,unit);
```

Reads the next record from the file into the buffer area starting at buff. At most, len bytes will be transferred. A null byte is NOT appended. If a partial record is transferred, the remainder is discarded. The actual number of bytes transferred is returned. If an end-of-file or error condition occurs, -2 (ERR) is returned. fread does not default to the console.

- Write a record to a file.

```
c=fwrite(buff,len,unit);
```

Writes a record of len bytes from the buffer area starting at buff. If len is greater than the maximum record length for the file the record is truncated. No special action occurs for null or EOL bytes. The actual number of bytes transferred is returned. If an error condition occurs, -2 (ERR) is returned. fwrite does not default to the console.

Note: fread/fwrite provide a binary I/O capability for applications such as printer graphics since all bytes are transferred regardless of value. If the buffer area is comprised of consecutive global variables and arrays then transfers can be made directly from/to the variables and arrays. If the first element of the buffer area is a scalar variable then its address (&var) must be used in the function call.

- Set record number.

```
fseek(unit,recno);
```

Sets the record number for the next I/O operation (fread or fwrite) on unit. This function provides a random access capability for files opened as relative. If a single fseek is followed by multiple fread or fwrite operations access becomes sequential starting with recno.

- Delete a file.

```
fdelete(filename);
```

Deletes the file specified by filename, which must be a string or pointer. No error conditions are returned.

- Test for end-of file.

```
c=feof(unit);
```

Returns a true value if the next read from unit would return an end-of-file error condition. Returns false otherwise.

- Get error code.

```
c=ferrc(unit);
```

If the previous I/O operation resulted in an error, returns the error code. This function should be used only when an error has occurred. The returned value is meaningless otherwise. Error codes are listed in the TI-99/4A reference manual. This function cannot be used after fopen errors since unit is not valid at that time. If used immediately after a successful fopen, ferrc returns the actual record length of the file.

- Rewind a file.

```
rewind(unit);
```

If a disk file is open for read or append it is rewound. All other cases are ignored.

VI. ASSEMBLY LANGUAGE INTERFACE

Interfacing to assembly language is relatively straightforward. The "#asm ... #endasm" form allows the placing of assembly source code directly into the program. Since the compiler considers it to be a single statement, it may be used as:

```
while(1) #asm ... #endasm
```

or

```
if(expression) #asm ... #endasm else ...
```

In actual program coding, the #asm directive must be the last item on a line and the #endasm directive must appear on a line by itself. Since the compiler is free-format otherwise, the expected format is:

```
if(expression) #asm
---
---
#endasm
else statement;
```

A semicolon is not required after #endasm.

Assembly code within the "#asm ... #endasm" form has access to all global symbols and functions by name. It is the programmer's responsibility to know the data type of the symbol (whether "int" or "char" implies word or byte access). Stack locals and arguments may be retrieved by offset.

The push-down stack used by c99 is located in the upper part of the low (8k) bank of the TI-99/4A memory expansion. Register 14 in the c99 workspace is reserved as the stack pointer. The stack begins at >3FFE and grows towards >2000.

External assembly language routines accessed by function calls from c code may use registers R0 thru R7. They may push items on the stack, but must pop them off before exit. It is the responsibility of the calling program to remove arguments from the stack after a function call. Since arguments are passed by value, the arguments on the stack may be modified by the called program.

Disk 27. Contents of file C99MAN2

VII. MEMORY UTILIZATION

PAD Usage.

c99 reserves PAD locations >8300 thru >832F for its workspace and support code. Other PAD locations not used by console routines are available to the programmer. In particular, locations >8330 thru >8348 can be used to store frequently accessed global variables by use of AORG and DORG directives. The file SIEVE;C, found on the release diskette, uses this technique.

The c99 workspace is at >8300. Register utilization is:

R0-R7	temporary storage
R8	primary computation register
R9	local address register
R10	address of the least-significant byte of R8
R11	return address for BL instruction
R12	address of recursive subroutine call routine
R13	address of recursive subroutine return routine
R14	the stack pointer
R15	first word of the PUSH routine (hence BL 15)

Memory Expansion Usage.

The entire 24K bank of memory is available for program usage. The 8K bank contains the standard Editor/Assembler utilities (>2000 thru >2676). As previously mentioned, the c99 stack grows down from >3FFE. Since typical stack usage is a few hundred bytes, the intervening space is available.

Since no indication of stack/program overlap is provided in this version of c99, very large programs could crash. However, the c99 compiler which uses all of the 24K bank and much of the 8K bank is able to compile itself successfully, so this problem may never arise for most users.

VDP Ram Usage.

Aside from the areas normally used in the E/A environment, c99 reserves VDP memory locations >1B70 (>1B5D for the delete function) thru 1FFF for file I/O requirements. This area was chosen to permit implementation of bitmap graphics.

VIII. STACK USAGE

c99 makes extensive use of the previously-mentioned push-down stack for temporary storage. Function arguments are pushed onto the stack as they are encountered between parentheses, so the last argument is at the "top" of the stack. This inverse order is somewhat unconventional. After all arguments have been pushed on the stack, the return address is pushed on by the recursive call code accessed via R12. Since the stack grows downwards in memory, the last argument value is located 2 bytes above the stack pointer's current contents at function entry.

As specified in the C language definition, parameter passing is "call by value". If X and Y are global variables, the compiler generates the following code for this C statement:

```
X=function1(X,Y,zf());
MOV  @X,8         value of X to primary register
BL   15           push onto stack
MOV  @Y,8         ditto for Y
BL   15
BL   *12          recursive call to zf
DATA ZF           function value is returned in primary reg
BL   15           push value onto stack
BL   *12          call function1
DATA FUNCTI       note 6 characters used
AI 14,6           restore stack pointer
MOV 8,@X          primary reg to X
```

As function1 is entered, the stack contents are:

```
      .
      .
      .
-----------
value of X
-----------
value of Y
-----------
zf fcn value
-----------
return addr     <=== stack pointer (R14)
-----------
```

In this case, the value of Y could be accessed by "MOV @4(14),8". Local variables allocate as much stack space as needed and are assigned the current value of the stack pointer (after allocation) as their address. The compiler ensures that each variable is located on a word boundary. The declarations:

```
int z;
char array[5];
```

generate:

```
        AI   14,-8
```

which allocates space on the stack for 8 bytes (not initialized). References to z will be made to stack pointer+6. Note that the stack pointer changes by 8 (not 7) bytes, ensuring that the following instruction falls on a word boundary.

Until the stack pointer is altered again, , array[0] is at *14, array[1] is at @1(14), array[2] is at @2(14), etc. For this reason, imbedded assembly language code using "#asm ... #endasm" cannot access local variables by name, but must know their location relative to the stack pointer's current contents.

IX. PROGRAMMING INFORMATION.

- When a c99 program begins execution, the screen is in text mode (40 characters/line) displaying white characters on a dark blue background. The function tscrn provides a means of changing the default color. It is possible to access VDP registers and memory from c99 by using the appropriate values with pointers.

- The logical operators "&&" and "||" (with left-to-right evaluation and early dropout) are available in c99. The bitwise operators "&" and "|" will usually yield the correct results in logical expressions, but forms such as "if(i&j)" should be avoided as erroneous results may be produced (e.g. if i=1, j=2 then i&j=0). c99 follows the usual convention in using a non-zero value to represent a true condition and a zero value to represent a false condition.

- Global variables result in more efficient code than local variables but they cannot be used in recursive situations. In addition, global variables (especially arrays) increase the size of a program module.

- Functions may be passed the names of other functions as arguments for indirect calling. The dummy argument for a function name must appear in the argument list as a simple name and must be declared as an integer pointer. Calls should be coded as "fcn()" and not as "(*fcn)()".

- Since c99 programs are self-contained, they may be saved as program files. Two object files, C99PFI and C99PFF, have been placed on the release diskette to facilitate program file creation. C99PFI must be loaded before user programs and libraries since it defines the entry points SFIRST and SLOAD and contains code to check for the presence of the Editor/Assembler cartridge and load the standard utilities from the Ed/Asm GROM. C99PFF must be loaded after all programs and libraries as it defines the entry point SLAST. If the E/A program SAVE is then loaded and run, a useable program file should result. The c99 compiler and utility program files were created in this manner.

- If program files are run using an Editor/Assembler simulator, they will only run if the standard utilities are available. The two simulators tested both satisfied this requirement. Since the possibility of over-writing the simulator code exists, exit is to the power-up title screen.

- If the rerun option is chosen at program termination, global variables are NOT reset to their initial values. Since this may cause undesirable results in some programs, the ability to bypass the rerun option is provided by exit(7).

- c99 has consistent handling of GROM addressing and GPL subprogram linkage. The GROM address is saved at program start and restored at program exit. C$GPLL, a special GPL link function which will function in any run mode, is included in CSUP. This function is used as follows:

```
#asm
 REF C$GPLL
#endasm
 . . .
#asm
 BL @C$GPLL
 DATA n   (n is the address of the required routine, as specified
#endasm    for GPLLNK in the E/A manual)
```

A modest degree of incompatibility exists between this version of c99 and the versions 2.0/2.1. Indirect function calls are handled differently, so any library functions using this feature must be recompiled. The new CSUP library must be used with all programs compiled by c99 v3.0, but this library is compatible with functions compiled by v2.0/2.1. Since the object code produced by v3.0 is more efficient and compact, consideration should be given to recompiling all frequently used programs and functions.

X. REFERENCES.

The C Programming Language by Brian Kernighan and Dennis Ritchie, Prentice-Hall 1978.
The C Primer by Les Hancock and Morris Kreiger, McGraw-Hill 1982.
Learning to Program in C by Thomas Plum, Plum Hall Inc. 1983.

A number of c99 users have found *The C Primer Plus* to be quite useful.

Disk 28. TI-Writer Help Disk

Version: *Author*:
Requires: TIW *Language*: *Updated*:

A set of text files on TI-Writer including a reference chart and a unique tutorial (over 100 sectors long) with commented formatting commands. Also includes a program to run TI-Writer from Mini Memory with a working SD (Show Directory) command.

```
dskdir. v2.0. 12-dec-96
Disk name               = TIW-HELP
Sectors total           = 360
Sectors used            = 247
Sectors available       = 111
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >008 -README       5 DIS/VAR 80   >10e 004
002 >007 MMTIW        17 DIS/FIX 80 Y >0fe 016
003 >002 TI-REWRITE  101 DIS/VAR 80 Y >022 100
004 >006 TIW-BUGS     13 DIS/VAR 80 Y >0f2 012
005 >003 TIWRIT/PT1   57 DIS/VAR 80 Y >086 056
006 >004 TIWRIT/PT2   36 DIS/VAR 80 Y >0be 035
007 >005 TIWRIT/REF   18 DIS/VAR 80 Y >0e1 017
```

Disk 28. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

TI-Writer Help Disk

Notes by J. Peter Hoddie

MMTIW: TI-Writer loader by Clint Pulley. Runs ONLY from Mini Memory Module. Load with "Load and Run" option. Program name is START. Allows use of SD (Show Directory) command.

TI-REWRITE: Complete reworking of TI-Writer manual by Dick Altman. In a unique format. The formatting commands are all commented completely.

TIW-BUGS: Description of some of the bugs in TI-Writer by Jim Peterson of Tigercub Software. Some odd but useful information.

TIWRIT/PT1 and TIWRIT/PT2: Lots of notes and hints on using the Editor.

TIWRIT/REF: A one page reference chart of information on using TI-Writer.

Disk 28. Contents of file TI-REWRITE

INSTRUCTIONS AND HINTS FOR TI-WRITER WORD PROCESSOR

by Dick Altman, July 27, 1985

IT *CAN BE* MASTERED! It just takes perseverance and determination and a desire. I have been using it since January 1985 and I don't have it all yet, but I can use it to my immense satisfaction. This came from months of sitting with the large manual in my lap flipping pages back and forth until I had practically *memorized* the #$% thing! I was at the point where when I had a problem I could say "Oh that is on page 146" or whatever. For instance: this article was done on the TI-Writer and I now do ALL of my correspondence with it also.

If you received the disk with this article, load it up in TI-Writer and call it up on the screen so that you can see which commands-and where they were used-to cause the different effects shown in this article. If you received the *disk* only, then you aren't reading this unless you have already booted it up. It is suggested that you run off a printed copy then reboot this back up so that you can see the commands in use as you read the article. There are comments in the program just below or above the commands that *don't* show in the printout! This is another "freeware" item. There is no price set for it.

Feel free to pass a copy on to whomever wants it. If it will help only one or two people that are struggling to learn TI-Writer I will be pleased. If you learn anything from it, and are inclined to fairness, send a few bucks when you can afford it to Dick Altman, 1053 Shrader St., San Francisco, CA 94117. There's no big deal if you don't — only *your* conscience will know. At least drop me a note and let me know it helped someone.

This is gonna be *loo-o-ng*, but still much shorter than the 175-page instruction manual!

FIRST RULE: Read the TI-Writer Quick Reference card and reread it. Of course this means *after* you read this article. Do *all* of the operations shown on the card-at least once-even though you might think you will never need that particular one. You will find you *have* to open up the big manual probably, to accomplish some of the operations. After you have almost "memorized" the card (literally!) then you will find yourself using it almost exclusively and very seldom having to refer to the cumbersome manual. Personally I think the manual is poorly written.

You will find 3 "windows" — from left to right — to obtain the 80 columns (80 normal characters) width. Each window is 40 columns wide. The first one is from 0 to 40, second one is from 20 to 60, and the third is from 40 to 80. The first thing I do upon booting up TI-Writer is to set my limits to 37 characters wide. If I take a whole window of 40 characters, it seems to crowd my screen, and I don't like to window back and forth to read my work. I do this by pressing **T** (for TABS), then press **ENTER**, then placing an "L" on the second dot, and an "R" on the 39th dot, then pressing **ENTER** again. Now I find my cursor blinking at me from line #0001. Here is where I tell the printer what margins I want it to print my work within. It's also at this point that I select condensed type because I like it better than the normal size type, and I can get 132 characters per line if I wish. It just looks better in my opinion. I normally do this on line 0002 because I used 0001 to set up the formatting (margins, etc.) commands to the printer.

.LM 25; RM 105; FI; AD (changes the margins)

So, on line 0001 I put in the following "dot" command (a dot command is merely starting with a period):

```
.LM 20;RM 120;FI;AD
```

(AND END *ALL* DOT COMMANDS WITH A "carriage return").

The semicolons are necessary, and the spaces, just as I listed it here. I'll do it again:

```
.LM 20;RM 120;FI;AD(c/r)
```

You of course don't put in the line number 0001. That is already there.

That tells the printer to set the Left Margin at 20, the Right Margin at 120, then Fill each line, and Adust (justify) the right margin. The "FILL" command tells the program to put in as many *whole* words on a line, within your predetermined margins, as possible. The "ADJUST" tells it to add extra blanks between words to cause the even right margin as this article has.

.LM 5;RM 130;FI;AD;IN 10 (returns margins)

I changed the margin settings on the last two paragraphs just to show you that you can enter your "commands" just about anywhere within your work!

.The page number was with the FO command

Just pressing **ENTER** will normally automatically put in the "carriage return" symbol, but sometimes it doesn't. It depends on what you were doing last. In that case, use **CTRL** and **8** to put in a carriage return.

On line 0002 I put in a "Control" command thusly: **CTRL U SHIFT O CTRL U**. Neither a "dot" at the beginning, nor a "carriage return" at the end is necessary. This command throws the printer into "condensed" type. Neither of these two line numbers will be printed on paper. They are merely *formatting commands*. Most of the "Control" commands are listed at the bottom of this article.

Then if I want to center a title (or date) or some other heading at the top of my article, on line 0003 I put in another dot command like this: .CE (remember a carriage return is required at the end of all dot commands). If my title is say three lines of type, then make that dot command thusly: .CE3(c/r) otherwise it will "center" only one line. The centering command at the top of this article was ".CE5" because of the blank line in it. The lines you wish centered have to immediately follow the centering command.

The automatic page length is 66 lines. This gives you about six blank lines at the top and bottom of your page, and only fifty some actual lines of type. You can, with a dot command change your page length with this: ".PL ##" as I did in line 0002 of this article. (Not enough room in 0001.)

Then you start typing your article, letter, whatever. If you wish each paragraph to be indented, it takes another dot command of: .IN(number). If, as in my suggested margin settings of .LM 20;RM 120, you wished to indent each paragraph five spaces, the command would be: .IN 25 because the counting starts at zero or left edge of the paper. If you include the indent command with others in line 0001, the semicolon replaces all but the first dot, thus .LM 20;RM 120;IN 25. You may put more than one dot command on one line, or the Control commands, but never both of them on the same line.

The *fun* part of a word processor is the capability of inserting or deleting a word or an entire phrase without having to retype the entire page or article. Another fun thing is the ability to move a sentence or an entire paragraph to another place in your work. This is all done very simply. Just place your cursor in the last space before where you wish to insert another word and press the **FCTN** key and the number **2**. This causes everything beyond your cursor to move down one line, then type in your new word or sentence and after the space at the end of it press the **CTRL** and the **2** (just once) and everything will jump back up to your cursor! If you are near the beginning of a long paragraph it takes a little longer (a couple or three seconds) to reformat the paragraph, than it does if you are near the bottom of that same paragraph — DON'T get impatient and hit the keys again, just wait a couple of seconds!

To move let's say paragraph #10 into the #3 spot is just as easy. First look at paragraph #10 and make a note (mental??) of the line numbers on the first and last line. **FCTN** and **0** (zero) shows the line numbers or moves them off the screen. Suppose they were 0076 and 0093. Then determine what line number you wish it to be after. Let's suppose it was 0023. Then with **FCTN 9** go to the "command" line, type **M** (for Move) and hit **ENTER**. Then type in 0076 0093 0023 and hit **ENTER** again. Look at those numbers and read the instructions on the Quick Reference Card for MOVE.

(ESC G)

That last one turned "ON" double print on most dot matrix printers, there are two different commands to make neat printing. They are called "emphasized" and "double strike". You can't use (on *my* printer at least) the emphasized method while in condensed size of type. But I can use double strike. The difference is basically this. Both commands print each letter twice, but in two different ways. One of them (emphasized) moves the head slightly to the right so that each letter is a little thicker. Double strike just prints the line twice. I think emphasized is slightly faster than double strike, but I've never timed either of them. Since I use condensed printing almost exclusively, and can't use emphasized, I don't worry about it. Incidentally, you may enter these commands throughout your article. You just have to have them begin at the left margin of your work. As long as you begin dot commands with a period, and the control commands with **CTRL U** (and end dot commands with a carriage return, and control commands with **CTRL U** and/or a capital letter) you'll be O.K. Only *this* paragraph was using "double strike", look at the difference.

(ESC H)

An interesting fact about most printers is that it not only inserts unobtrusive spaces here and there to *ADJUST* each line to the predetermined right margin, IT PRINTS EVERY OTHER LINE FROM THE RIGHT TO THE LEFT while doing all that FILLING and ADJUSTING. It will also correctly number your pages if you give it the FO command, which is another dot command.

I find once in awhile, some one command (never the same one twice) seems to falter. Just redo it. sometimes I think some command must be there that is invisible (this is possible!) so when you run into an unexplainable problem, go back to your formatting command line(s) — which are usually lines 0001 and 0002 — put the cursor at the end of each of your commands then press **FCTN** and **1** and hold them for a couple of seconds to delete any possible typing errors that placed some sort of "hidden" command in that line.

Another *good* command to learn is the "OOOPS" command. Merely Control and the figure one. This eliminates only your last change just now typed in, and returns your work to its former self (hopefully!).

Another good habit to get yourself into, is "SAVING" your work every few minutes (or every few pages). Power glitches do occur from *any* power company. Either surges, or stumbles. Sometimes just an electric motor in your home (refrigerator, etc.) kicking in will cause a momentary change in the power supplied to your computer (you've seen your lights flicker). If you save your work every once in awhile, you someday will be glad you were in the habit. Especially if you have just put in to the word processor a 20,000 word story. The power glitch could cause you to lose it all! If you have been saving it on a disk, when that glitch occurs you will have all but a small part of it saved. When you save something to a disk, then come back to that same disk and save something else with the same name, it replaces the first item with the second. It does not become two separate items on the disk. Of course, if you are really a worry-wart, you will do the saving on two disks, alternating back and forth, just in case that glitch comes while you are in the act of saving your work.

When you wish to reload a file from a disk back into the word processor, it's *EASY*! When you first bring up the word processor in the Editor mode, you are automatically in the command line. Just type LF (for Load File) and hit **ENTER**, then type in DSK1.(and the name you gave it) then hit **ENTER** again and wait a few seconds for the work to be loaded into your computer from the disk.

If you want a rough draft of your work on paper (I find it easier to proof than on the screen) just remove your commands for double strike or emphasizing to conserve your printer ribbon. It will not be so easy to read, unless your ribbon is new, but it will be done faster, as well as not using up ribbon ink unnecessarily.

In the book you will find two methods of going to the disk, then to your printer. Printing should be done from the disk, not from the computer. You will find a command of "Print File". That's not the one I use! The one I have become accustomed to using may take a few seconds longer, but it is the one I learned first, and I have just stuck with it. It is as follows. After I have finished typing my letter or whatever, return to the *command* line with **FCTN 9**, there type a **Q** (for Quit) hit **ENTER**, then **S** (for Save) and **ENTER**, then DSK1.TERRY or whatever name I want to give the file instead of TERRY, then **ENTER**. I usually use a short two or three character name. I have even been known to use #1, or #2, or something like that (the file name can*not* be more than 10 characters long, and you can't have any spaces in a file name). Then, after the work goes from the computer to the disk, you can either print it now or sometime next week. The command to go to the printer at this point is like this: **Q** (for Quit) **ENTER**, then **E** (for Exit) and **ENTER** again. This takes you back to the master menu. This time, you select #2, or THE FORMATTER. After it comes up, you have to type in DSK1.(filename) and hit **ENTER**. Then you have to type in the command telling it to go from the disk to the printer, instead of to the screen. (With the use of DISKO or some such assembly language repair program, you can insert the command to your printer so that it is a default just like all the other selections on the screen. It is in "EDITA1" of your TI-Writer disk.) Without knowing what kind of printer you have, I can't give exactly the correct command here, but it will be something like this: PIO or RS232.BA=4800.LF, then you will have five more choices, mostly for which you will just press **ENTER** for each of them. Perhaps you might wish more than one copy, so on the correct one you would punch in that number. Be sure your printer is turned "on" before hitting the last **ENTER**, (the one that says "PAUSE AT END OF THE PAGE?) because you will be printing immediately.

.LS 2
.That one caused double spacing

For *your* purposes (manuscript writing) you will want it double spaced. That is simply a dot command of ".LS 2" (LS for Line Spacing of course!) and if you want it triple spaced, just change the 2 to a 3. Or of course use it for a rough draft or some such. I'm mostly just rambling here, to give this particular paragraph some length, so that you can see double spacing at work. I can't seem to think of anything else to say, so I will just end it here.

.LS

There are many, many more commands available, such as merging either parts of two different files, or merging a whole file into the middle of another, or putting in headers at the tops of every page, and footers at the bottom, all automatically. Such things as page numbers, or requirements for manuscripts, etc., but those can be found as you need em.

The word processor *does* have a capacity beyond which you have to save your work to disk, and start with a clean slate. It is approximately 20,000 characters including blanks. I have only run into it when transferring a long story to disk. I was entering a 10,000 word story, and I got "MEMORY FULL-SAVE OR PURGE" flashing at me at the top or *command* line after about 4,000 words (I wish it would ring a bell or something). At that point "save" your work and retire that file name. Perhaps in this article I am writing for you I will reach that point again. Right now I am typing on line number 466. I think it was at about line 400 plus (but I was using 80 column width that time for a special project, I think) that the MEMORY FULL thing happened to me. You will just have to *trial and error* it for your job! Of course, the length can*NOT* be judged just by the line numbers on the left side of your screen. Think about whether you are using only one window, or two, or the maximum of three. I am using just one window while I do this work, as I explained earlier, so that will make my capacity come much farther down the line numbers than if I were using all three windows! 80 characters (or columns) wide, instead of the 37 I am using. If and when the MEMORY FULL bit happens to you, remember that when you save it this time to a disk, then for Pete's sake don't save the next time to the same file name! In other words, my name for this file at the moment is TI-WRITER. If I need to make a new file, it will become TI-WRITER2.

The little 25 page booklet from Dr. Bill Browning is very good, don't ignore it when you are trying to learn the TI-Writer word processor. 7541 Jersey Avenue North, Brooklyn Park, MN 55428. Price just $6.50 and worth every penny.

There is also available in "freeware" circles an excellent disk called "TK-Writer" which was done by Tom Knight, thus the "TK". It replaces the need for a cartridge to have TI-Writer word processing capabilities. As far as I can tell, it does exactly the same things the cartridge does, except for Show Directory — which is inconsequential, and won't go direct from the Editor stage to the Formatting stage. You can probably find it in the same library you obtained this disk from.

The command for the underscore is merely the ampersand (**SHIFT 7**) and it can be used anywhere. Note even in the middle of the word "cannot". If you want to underline *more than one word* you have to connect them with what is called a caret. It is above the 6, or **SHIFT 6**. If you wish, the Ampersand *can* be printed in your work, but not the caret. Merely type in *two* ampersands and only one of them will be printed! & & &

Believe me, all of this *will* become easy and second nature to a good typist in a very short time! But if you don't use it for a month or two, you will find yourself going back and back and back to the big book!

Thanks so much to Dr. Guy Romano for his assistance in writing this article. Plus his enormous patience with my *dumb* questions over the past few months while I was learning the TI-Writer. Also to Hal White and to Larry Rosenberg for their invaluable assistance. And to Terry and Paul Anderman for their desire to have word processing capabilities, which forced me to finally write this that had been nagging at me so long.

CONTROL COMMANDS

****

ASCII
CODES FUNCTION FORMAT

0 Terminate Tabulation CTRL U, SHIFT 2, CTRL U
7 Sound the buzzer CTRL U, SHIFT G, CTRL U
8 Backspace CTRL U, SHIFT H, CTRL U
9 Horizontal tabulation CTRL U, SHIFT I, CTRL U
10 Line feed CTRL U, SHIFT J, CTRL U
11 Vertical tabulation CTRL U, SHIFT K, CTRL U
12 Form feed CTRL U, SHIFT L, CTRL U
13 Carriage return CTRL U, SHIFT M, CTRL U
14 Print enlarged characters CTRL U, SHIFT N, CTRL U
15 Print condensed characters CTRL U, SHIFT O, CTRL U
17 Select printer CTRL U, SHIFT Q, CTRL U
18 Turn off condensed printing CTRL U, SHIFT R, CTRL U
19 Disable printer CTRL U, SHIFT S, CTRL U
20 Turn off enlarged printing CTRL U, SHIFT T, CTRL U
27 Escape CTRL U, FCTN R, CTRL U
27;48 Set line spacing 8 per inch CTRL U, FCTN R, CTRL U, 0
27;50 Set line spacing 6 per inch CTRL U, FCTN R, CTRL U, 2
27;51 Set line spacing n/216 per inch CTRL U, FCTN R, CTRL U, 3,n
27;52 Turn Italic Character set on CTRL U, FCTN R, CTRL U, 4
27;53 Turn Italic Character set off CTRL U, FCTN R, CTRL U, 5
27;56 Disable paper-end detector CTRL U, FCTN R, CTRL U, 8
27;57 Select paper-end detector CTRL U, FCTN R, CTRL U, 9
27;65 Set line spacing(1/72 to 85/72 inch) CTRL U, FCTN R, CTRL U, A,n
27;66 Set up 8 vertical tab pos. CTRL U, FCTN R, CTRL U, B
27;67 Set form length up to 127 lines CTRL U, FCTN R, CTRL U, C,n
27;68 Set up to 12 horizontal tab positions CTRL U, FCTN R, CTRL U, D
27;69 Turn on emphasized printing CTRL U, FCTN R, CTRL U, E
27;70 Turn off emphasized printing CTRL U, FCTN R, CTRL U, F
27;71 Turn on double printing CTRL U, FCTN R, CTRL U, G

```
=============================================================================
27;72 Turn off double printing CTRL U, FCTN R, CTRL U, H
27;75 Turn on normal density graphic printing CTRL U, FCTN R, CTRL U, K
=============================================================================
27;76 Turn on dual density graphic printing CTRL U, FCTN R, CTRL U, L
27;77 Turn Elite mode ON CTRL U, FCTN R, CTRL U, M
=============================================================================
27;78 Set skip-over perforation CTRL U, FCTN R, CTRL U, N
27;79 Release skip-over perforations CTRL U, FCTN R, CTRL U, O
=============================================================================
27;80 Turn Elite mode OFF CTRL U, FCTN R, CTRL U, P
27;81 Set a column width CTRL U, FCTN R, CTRL U, Q
=============================================================================
27;82 Select 1 of 8 int'l char.sets CTRL U, FCTN R, CTRL U, R
```

Disk 29. More Printer Utilities

Version: *Author*:
Requires: XB *Language*: *Updated*: 08/16/87

Banner, Gothic, multiple columns, sideways, printer set-ups, and more. An extremely useful printer disk. Primarily for Epson and compatibles. Some for Gemini.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-PRINT
Sectors total           = 360
Sectors used            = 168
Sectors available       = 190
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       8 DIS/VAR 80   >022 007
002 >003 BANNER       20 PROGRAM    Y >029 019
003 >004 CLOCK         9 PROGRAM    Y >03c 008
004 >005 EPSON         7 PROGRAM    Y >044 006
005 >006 GEMINI       10 PROGRAM    Y >04a 009
006 >007 GOTHIC       49 INT/VAR254 Y >053 048
007 >008 LOAD         13 PROGRAM    Y >083 012
008 >009 PRINT        16 PROGRAM    Y >08f 015
009 >00a SIDEWYS/CH    9 PROGRAM    Y >09e 008
010 >00b SIDEWYS/SC    9 PROGRAM    Y >0a6 008
011 >00c SIDEWYSTXT   18 DIS/VAR 80 Y >0ae 017
```

Disk 29. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

Print Utilities Disk

Notes by J. Peter Hoddie

LOAD and CLOCK: Two Extended BASIC programs put together by Bill Wallbank.

BANNER: Program in Extended BASIC to print banners on your printer. Will work with any printer. By Jim Dresser.

EPSON: Program to set up printing characteristics for an Epson or Epson-compatible printer. By Bill Wallbank in Extended BASIC.

GEMINI: Program in Extended BASIC to allow you to design and download custom characters to a Gemini printer. By Jim Peterson of Tigercub Software.

GOTHIC: Program in Extended BASIC of unknown origin. Allows you to print in a fancy Gothic print style on Epson and Epson-compatible printers.

PRINT: Program in Extended BASIC by J. Peter Hoddie to allow you to print in multiple columns to create newsletters and the like. The best method for using this program is as follows. Create text file using TI-Writer. Run it through the TI-Writer formatter to set the text to the desired column width (such as 40). Then go back into the editor and remove all line feed characters (Control J) as these mess up the PRINT program. Then save the file back to disk using the PF (print file) option . . . not SaveFile as this saves tabs and makes a mess at the end of the file. Then run PRINT. It's much easier then it sounds.

SIDEWAYS: Program by Tom Freeman which allows you to print out text sideways across a page. Documentation in file SIDEWYSTXT.

Disk 29. Contents of file SIDEWSYTXT

MAKE YOUR PRINTER PRINT SIDEWAYS!!

by Tom Freeman

[See The Cyc, Appendix, *Genial TRAVelER* for the contents of this file — Ed.]

Disk 30. Forth Programs #1

Version: *Author*: John Volk
Requires: EA and Forth Disk *Language*: Forth *Updated*:

Collection of Forth programs and utilities. Includes graphics, a word to "dis-assemble" a Forth word, and over 25 others.

```
dskdir. v2.0. 12-dec-96
Disk name               = FORTH
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 0
Number of sides         = 0
Density                 = unknown

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 SCREENS     358 DIS/FIX128   >022 326 >003 031
```

Disk 31. Forth Programs #2

Version: *Author*: John Volk
Requires: EA and Forth Disk *Language*: Forth *Updated*:

Another collection of Forth programs. Primarily games. A great way to learn how to translate from BASIC to Forth and how to use graphics (including bitmap) in Forth.

```
dskdir. v2.0. 12-dec-96
Disk name               = FORTH
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 0
Number of sides         = 0
Density                 = unknown

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 SCREENS     358 DIS/FIX128   >022 326 >003 031
```

Disk 32. Music Programs

Version: *Author*: Jay Guirleo, others
Requires: XB *Language*: XB *Updated*:

A collection of music programs from a variety of sources including Jay Guirleo. Some of the best music programs for the TI. Many with graphics.

```
dskdir. v2.0. 12-dec-96
Disk name               = MUSIC1-BCS
Sectors total           = 360
Sectors used            = 316
Sectors available       = 42
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00e -README       5 DIS/VAR 80   >14d 004
002 >00a BEETHOVEN    30 PROGRAM      >0f2 029
003 >003 CHORAL        8 PROGRAM      >03d 007
004 >004 FUGUE        45 PROGRAM      >044 044
005 >009 LUDWIG/MMM   17 PROGRAM      >0e2 016
006 >008 MOONLIGHT    49 PROGRAM      >0b2 048
007 >007 MORNING      42 PROGRAM      >089 041
008 >00b MUSIC-BOX    23 PROGRAM      >10f 022
009 >005 PRELUDE      14 PROGRAM      >070 013
010 >00d SCALE        17 PROGRAM      >13d 016
011 >00c TI-DEMO      25 PROGRAM      >125 024
012 >002 TOCATTA      28 PROGRAM      >022 027
013 >006 TRUMPET      13 PROGRAM      >07d 012
```

Disk 32. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston MA, 02108

Music Disk #1

Notes by J. Peter Hoddie

CHORAL, FUGUE, TRUMPET, PRELUDE. Four programs by Jay Giurleo. All run in Extended BASIC from CHORAL.

BEETHOVEN: Medley of tunes by Beethoven. With graphics. In Extended BASIC.

LUDWIG/MMM: Music by Beethoven along with a demo of creating sprites using Mini Memory Module by Bill Wallbank and Joe Rawlins.

MOONLIGHT: Beethoven's Moonlight sonata with river graphics. In Extended BASIC.

MORNING: Morning Has Broken by Cat Stevens. Program by Sam Moore. In Extended BASIC. With graphics.

MUSIC BOX: Plays Music Box Dancer. In Extended BASIC by Charles Ehninger.

TOCATTA: An incredible demo of music on the TI. With graphics. In Extended BASIC.

TI-DEMO: Demo music by Texas Instruments. No graphics. Better in TI-BASIC.

SCALE: Program to play all types of scales in any key by David Taub. In Extended BASIC.

Disk 33. Music Programs #2

Version:
Author: Sam Moore Jr.
Requires: XB
Language: XB
Updated:

Another collection of music programs including Flight of the Bumblebee, Western Boogie, and Boatsong. All of these programs have graphics. Many are animated.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-33
Sectors total           = 360
Sectors used            = 333
Sectors available       = 25
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       2 DIS/VAR 80   >022 001
002 >003 BEER/POLKA   35 PROGRAM      >023 034
003 >004 BOATSONG     27 PROGRAM      >045 026
004 >005 BOOGIE       26 PROGRAM      >05f 025
005 >006 BUMBLEBEE    43 PROGRAM      >078 042
006 >007 GODFATHER    37 PROGRAM      >0a2 036
007 >008 MAINSCREEN   17 PROGRAM      >0c6 016
008 >009 PIANO        30 PROGRAM      >0d6 029
009 >00a PUPPYTOWN    34 PROGRAM      >0f3 033
010 >00b RAINBOW      24 PROGRAM      >114 023
011 >00c SPACEMUSIC   23 PROGRAM      >12b 022
012 >00d STAR-TREK    35 PROGRAM      >141 034
```

Disk 33. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

All programs will run in Extended BASIC.

Many thanks to Sam Moore Jr. who wrote most of the programs on this disk.

Disk 34. Statistics and Sorting

Version:
Author: John Clulow, David Romer
Requires: XB, EA
Language: AL
Updated:

Sort allows you to sort by two separate fields and to choose between two types of sorts. STAT is a set of statistics routines for use in Extended BASIC.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-34
Sectors total           = 360
Sectors used            = 192
Sectors available       = 166
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       3 DIS/VAR 80   >022 002
002 >003 CHARA1        9 PROGRAM      >024 008
003 >004 DEMOFILE      3 DIS/VAR 80   >02c 002
004 >005 DOCUMENT     32 DIS/VAR 80   >02e 031
005 >006 SDEMO         4 PROGRAM      >04d 003
006 >007 STAT         23 DIS/FIX 80   >050 022
007 >008 STAT-DOC     31 DIS/VAR 80   >066 030
008 >009 STATSOURCE   54 DIS/VAR 80   >084 053
009 >00a UTIL1        33 PROGRAM      >0b9 032
```

Disk 34. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

Documents for the SORT utility can be found in the DOCUMENT file.

Documents for the STAT routines can be found in the STAT-DOC file.

Many thanks to John Clulow and his New Horizons User Group for these and their other fine programs.

Disk 34. Contents of file DOCUMENT

TI-SORT UTILITY

Overview

This program consists of the following tools for the manipulation of DIS/VAR 80 files:

(1) a choice of single or bi-level ascending ASCII Quicksort or Shellsort routines on user defined primary and secondary sort fields;

(2) a loader routine which reads the DIS/VAR 80 file into RAM; and

(3) a save routine which writes the file back to the disk or to a printer.

With these routines the user can load any Dis/Var 80 file of 300 records or less into high RAM to form a simulated "RAM disk". At this point either of the two sort routines may be called; Shell or Quicksort. The choice of sort algorithm depends on the amount of sorting required. When the file is already nearly sorted (e.g., a few new entries have been appended to an already sorted file), then the Shell sort will be the faster. On the other hand, when a file is in random order (e.g., to be sorted on a new key) then the Quicksort will be faster. These criteria apply to either a single or bi-level sort. The sorted file can then be saved on the disk, under a new file name if desired.

Hardware Requirements

TI-SORT requires the 32k memory expansion and one disk drive.

Software Requirements

TI-SORT requires either the TI-Writer or Editor/Assembler command module or the TK-Writer loader package for TI-Writer.

Creating a TI-SORT Utility File

Files may be created with TI-Writer, Editor/Assembler or any program that creates DIS/VAR 80 files. The sort routines reference a "field" on each record. A field is a group of consecutive bytes referred to by byte position within the record. The first byte in a record is byte 1 and the last byte is byte 80. If, for example, the "last name" entry of an address list were placed in the first 15 bytes of each record then that field would be referenced by the beginning byte 1 and ending byte 15. To sort correctly, alpha fields should be left justified and numeric fields should be right justified. Dates must be entered in Year/Month/Day format to sort correctly. For instance:

```
 123456789012345678901234567890123456
Clark Mary F 38 46/01/15
Hennessy Herbert M 5 80/11/29
Clown Bozo ? 25 60/04/11
```

In this example, last name is field 1-12, first name is 13-19 and sex is 21-21. These three fields are left justified because they will be sorted as alpha fields. Age is field 23-25. This field will be sorted as a number so it is right justified. Birth date is field 27-34. Because this field will be sorted as a date, it is expressed in year/month/day format with individual entries right justified.

Loading TI-SORT Utility

TI-SORT is a program file, named UTIL1, that may be loaded and run with option 3 from the TI-Writer menu or option 5 from the Editor/Assembler menu. To load TI-SORT with the utility load selection in TK-Writer you must make certain that TI-SORT is named UTIL1 on a disk in drive 1 that also has the CHARA1 file on it. Once TI-SORT has loaded you will be presented with the TI-SORT menu.

```
1. TO LOAD FILE
2. SAVE FILE
3. QUICK SORT
4. SHELL SORT
```

Arrow keys, function DEL, INS, ERASE and BACK are enabled in TI-SORT.

LOAD FILE

To load a file select option 1 and at the prompt enter a legal disk file name (e.g. DSK2.DEMOFILE). The other TI-SORT menu choices are NOT enabled until a successful load file has been completed.

SAVE FILE

To save a file select option 2 and at the prompt enter a legal disk or printer file name. The printer specification may be for either a serial or parallel setup.

QUICK or SHELL SORT

A choice of menu options 3 or 4 chooses the type of sorting algorithm you wish to use. With either choice you are shown the current first record in the file along with an indicator scale of byte position. Enter the beginning and ending byte positions of the primary sort field first followed by the beg- inning and ending positions for the secondary sort field. If you leave the beginning secondary position blank a one level sort based on the byte positions in the primary field will be executed. For instance in our above example file, a two level sort of first name within last name would be entered as byte positions 1 and 12 for the primary sort and byte positions 13 and 20 for the secondary sort. The ending byte positions must be greater than the beginning byte positions. The primary and secondary sort fields must not overlap and the combination of the two fields must not exceed 80 bytes. You may sort on a field of a single byte. While the sort is working the word 'WORKING' along with a flashing cursor will be displayed at the bottom of the screen. A sort of the maximum 300 records on the full 80 byte length may take approx. 25 seconds. Upon completing the sort TI-SORT will return to its' main menu.

DISK CONTENTS

UTIL1	TI-SORT program.
DOCUMENT	Documentation file
DEMOFILE	Demonstration file of 15 records
CHARA1	Character set file required for TK-Writer

ADDRESS INQUIRIES TO:

David R. Romer
213 Earl St.
Walbridge, Oh. 43465

Disk 34. Contents of file STAT-DOC

Extended BASIC A/L Routines. New Horizons Users Group

STATISTICS

This routine is provided by New Horizons Users Group updating the procedures provided on the Extended BASIC A/L disk. There are no restrictions regarding its distribution; however, we ask that the author and source be acknowledged.

STAT — (J. Clulow)

This routine calculates correlation (Pearson Product Moment) and variance - covariance matrices for raw data input. In addition, a separate matrix of means and standard deviations is computed.

Return values are passed to the Ext. BASIC program via two arrays. The first is N X N and the second 2 X N where N is the number of variables. Data are passed to the assembly language routine by executing a CALL LINK statement for each observation or case. The data are contained in a one dimension array D(N) with N being the number of variables.

First the STAT routines must be loaded. This is done with the statements:

```
100 CALL INIT
110 CALL LOAD("DSK1.STAT")
```

OPTION BASE 1 must be in effect and the arrays must be DIMensioned for the proper number of variables. If a study measured four traits on several persons;

```
120 OPTION BASE 1
130 DIM A(4,4),B(2,4),D(4)
```

Before any data are entered, the arrays and internal observation counter must be initialized. This is done with the statement:

```
140 CALL LINK("INIT",A(,),B(,))
```

Errors made in the DIM statement will be reported upon execution of the statement.

Data are entered with the CALL LINK:

```
300 CALL LINK("ENTER",D(),A(,),B(,))
```

This statement returns the running sum of each variable in B(1,) and the sum of X ^ 2 and X*Y in the upper right of array A(,). The Ext. BASIC program may obtain data from any I/O device - keyboard, disk, etc.

After all of the observations have been entered, the statistics are calculated with the statement:

```
400 CALL LINK("CALC",A(,),B(,))
```

This statement returns the means and standard deviations in B(1,) and B(2,), respectively. It returns the variable intercorrelations in the lower left of A(,), the covariance values in the upper right of A(,) and the variances on the diagonal. N-1 weighting is used for all statistics.

If prediction equations are desired, they may readily be obtained from the statistics returned. Given a predictive relation of the form Y = m*X + k, where values of Y are to be predicted given values of X, then m = r*(Sy/Sx) where r is the correlation coefficient and Sy and Sx are the standard deviations of the variables, and k = Y.-m*X. where Y. and X. are the means on Y and X.

DISASSEMBLER — (Texas Instruments) (mod. J. Clulow)

Given a starting and ending address in hexadecimal, the program determines assembly language equivalents of machine code. On pass one the result is displayed on the screen and optionally transferred to disk or printer. The left column contains the hex memory location in which the instruction starts. This is followed by the corresponding assembly language mnemonics and operands. The last column contains the first word of machine code at the address.

A second pass option was added to this program in which labels are inserted in the output. Up to 260 labels are available (A0 - A9 ... Z9). This allows easier modification of output for re-assembly if the objective is to produce relocateable code. However, it is unlikely that disassembled code will re-assemble without error. If an operand is in the range of addresses to be disassembled but if it does not represent the first word of an instruction, then it will be _____ an undefined symbol. After the first attempt at assembly, it is an easy matter to edit the source file, and make the necessary operand changes from the machine code in the comment field.

It is for this reason that the program was also modified to output full machine code in addition to the starting address in both passes and, in the right column, any displayable ASCII characters are printed — non-displayable characters are indicated with an asterisk *.

Three object programs are provided for various configurations: Mini-Memory (MMDIS), Extended BASIC (XBDIS), and Editor/Assembler (E/ADIS). In the E/A or Mini-Memory cases, the appropriate object file should be loaded with the LOAD AND RUN option and executed with the program name START. With Extended BASIC the load sequence is:

```
CALL INIT
CALL LOAD("DSK1.DSRLNK","DSK1.XBDIS")
CALL LINK("START")
```

The source program is DISASSM/S. When assembled, this version yields the Mini Memory object file. The source code contains full instructions for producing the Ext BASIC and E/A versions if required.

The MMDIS version will fit in the Mini Memory. It can then be used to disassemble programs loaded into the memory expansion. First initialize the Mini Memory. Then in TI BASIC enter the statement:

```
CALL LINK(28706,0,0,0,0,0,0,0,0)
```

Now load the program with the LOAD AND RUN option. Finally, enter the TI BASIC statement:

```
CALL LINK(28706,160,0,255,224,32,0,63,255)
```

The first CALL LINK forces the loader to load MMDIS into the Mini Memory module, and the second CALL LINK restores the pointers used by the loader to allow subsequent code to go into the 32K memory expansion unit.

Disk 34. Contents of file STATSOURCE

```
****************************************
*
* CORRELATIONAL STATISTICS ROUTINE
*
* Calculates Pearson Correlations
*            Variances
*            Covariances
*            Means
*            Standard Deviations
*
* By J. Clulow - New Horizons
*
****************************************
*
       DEF  INIT,ENTER,CALC
ARG    EQU  >835C
FAC    EQU  >834A
STATUS EQU  >837C
FADD   EQU  >0D80
FSUB   EQU  >0D7C
FDIV   EQU  >0FF4
FMUL   EQU  >0E88
FCOMP  EQU  >0D3A
VMBR   EQU  >202C
XMLLNK EQU  >2018
NUMREF EQU  >200C
NUMASG EQU  >2008
ERR    EQU  >2034
BADVAL EQU  >1E14
H1     DATA >0102
ONE    DATA >4001,>0000,>0000,>0000
NCASES DATA >0000,>0000,>0000,>0000
SAVE   DATA 0
MAXDIM DATA 0
WS     BSS  32
BUF    BSS  8
XBUF   BSS  8
*
****************************************
*
* ARRAY INITIALIZATION ROUTINE
*
* CALL LINK("INIT",A(,),B(,))
*
* WHERE A(,) IS AN N X N ARRAY
*       B(,) IS AN 2 X N
*
*       WITH N = # VARIABLES
```

```
*--------------------------------------
*
* R5 = MAX N DIMENSION 1st
* R6 = MAX N DIMENSION 2nd
* R7 = MAX D DIMENSION (2)
* R8 = MAX N DIMENSION 3rd
*
****************************************
*
INIT   MOV  R11,@SAVE
       LWPI WS
       LI   R0,16      1 * 8 + 8
       BL   @GTDIM
       MOV  @BUF,R5
       MOV  @BUF+2,R6
       LI   R0,24      2 * 8 + 8
       BL   @GTDIM
       MOV  @BUF,R7
       MOV  @BUF+2,R8
       C    R5,R6
       JNE  BADDIM
       C    R5,R8
       JNE  BADDIM
       CI   R7,2
       JNE  BADDIM
       JMP  OKDIM
BADDIM LI   R0,BADVAL
       SWPB R0          BAD SUBSCRIPT
       BLWP @ERR
OKDIM  CLR  R0
       MOV  R0,@FAC     CLEAR FAC
       MOV  R0,@FAC+2
       MOV  R0,@FAC+4
       MOV  R0,@FAC+6
       LI   R1,FAC
       LI   R2,NCASES   INIT. NCASES
       BL   @TRANSF
       MOV  R5,@MAXDIM  SAVE MAX DIM
       LI   R0,1
       LI   R1,1
       MOV  R6,R2
       MOV  R5,R3
       BL   @ERASE      ERASE 1ST ARRAY
       LI   R0,1
       LI   R1,2
       MOV  R8,R2
       MOV  R7,R3
       BL   @ERASE      ERASE 2ND ARRAY
       B    @RTN
****************************************
*
```

```
* DATA ENTRY ROUTINE
*
* CALL LINK("ENTER",D(),A(,),B(,))
*
* WHERE D()  DATA ARRAY OF DIM N
*       A(,) VAR/COVAR MATRIX DIM N X N
*       B(,) MEAN,SD MATRIX 2 X N
*
*--------------------------------------
*
* R5  NUMBER OF VARIABLES
* R10 I COUNTER - COLUMNS
* R9  J COUNTER - ROWS
*
**************************************
*
ENTER  MOV  R11,@SAVE
       LWPI WS
       MOV  @MAXDIM,@MAXDIM CHECK INIT
       JEQ  ERDIM
       LI   R0,16           1ST PARAM.
       BL   @GTDIM
       MOV  @BUF,R5
       C    R5,@MAXDIM   CHECK vs INIT
       JGT  ERDIM
       JMP  DIMOK
ERDIM  LI   R0,BADVAL
       SWPB R0           BAD SUBSCRIPT
       BLWP @ERR
DIMOK  LI   R1,NCASES
       LI   R2,FAC
       BL   @TRANSF
       LI   R1,ONE
       LI   R2,ARG
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FADD
       LI   R1,FAC
       LI   R2,NCASES
       BL   @TRANSF
       LI   R10,1     I = 1 TO R5
ENTST  MOV  R10,R9    J = I TO R5
       MOV  R10,R0    Ith ELEMENT
       LI   R1,1      DATA ARRAY
       BLWP @NUMREF
       LI   R1,FAC
       LI   R2,ARG    PUT X(I,J) IN ARG
       BL   @TRANSF
       LI   R1,FAC
       LI   R2,BUF    MAKE COPY IN BUF
```

```
       BL   @TRANSF
       MOV  R10,R0    Ith ELEMENT
       LI   R1,3      SUM/MEAN ARRAY
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA FADD      ADD TO SUM
       BLWP @NUMASG   PUT BACK IN SUM
ELOOP1 LI   R1,BUF    RELOAD X(I,J)
       LI   R2,ARG    INTO ARG
       BL   @TRANSF
       MOV  R9,R0     Jth ELEMENT
       LI   R1,1      DATA ARRAY
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA FMUL      PRODUCT
       LI   R1,FAC
       LI   R2,ARG    COPY TO ARG
       BL   @TRANSF
       MOV  R9,R0     Jth ELEMENT
       MOV  R10,R1  -- ADJ FOR COL --
ELOOP2 DEC  R1
       JEQ  ENEXT1    R0 IS ELEMENT
       A    R5,R0     IN SS/SP ARRAY
       JMP  ELOOP2
ENEXT1 LI   R1,2      SS/SP ARRAY
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA FADD
       BLWP @NUMASG   PUT IT BACK SS/SP
       INC  R9
       C    R9,R5
       JGT  ENEXT2
       JMP  ELOOP1    NEXT J
ENEXT2 INC  R10
       C    R10,R5
       JGT  NRET
       JMP  ENTST     NEXT I
NRET   B    @RTN      BACK TO BASIC
*
**************************************
*
* CALCULATION ROUTINE
*
* CALL LINK("CALC",A(,),B(,))
*
* WHERE A(,) IS VAR/COVAR MATRIX
*       B(,) IS MEAN/SD MATRIX
*
*********************************
*
CALC   MOV  R11,@SAVE
```

```
       LWPI WS
       MOV  @MAXDIM,@MAXDIM CHECK INIT
       JEQ  DERROR
       LI   R0,16           1ST PARAM.
       BL   @GTDIM
       MOV  @BUF,R5
       C    R5,@MAXDIM   CHECK vs INIT
       JNE  DERROR
       LI   R1,NCASES
       LI   R2,ARG
       BL   @TRANSF
       LI   R1,ONE
       LI   R2,FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FCOMP
       MOVB @STATUS,R1
       ANDI R1,>4000
       JNE  OK
OVFL   MOVB @H1+1,R0
       BLWP @ERR
DERROR LI   R0,BADVAL
       SWPB R0           BAD SUBSCRIPT
       BLWP @ERR
*
*        ROUTINE FOR...
*
*    A(I,I)   VARIANCE
*    A(I,J)   COVARIANCE
*    B(1,I)   MEANS
*    B(2,I)   STANDARD DEVIATIONS
*
OK     LI   R10,1     I = 1 TO R5
CLOOP1 MOV  R10,R9    J = I TO R5
CLOOP2 MOV  R9,R6
       MOV  R10,R1
CLOOP3 DEC  R1
       JEQ  CNEXT1
       A    R5,R6     R6 A(,) OFFSET
       JMP  CLOOP3
CNEXT1 MOV  R6,R0     GET SUM (XI.XJ)
       LI   R1,1
       BLWP @NUMREF
       LI   R1,FAC
       LI   R2,BUF
       BL   @TRANSF      SAVE IN BUF
       MOV  R10,R0
       LI   R1,2
       BLWP @NUMREF      GET SUM XI
       LI   R1,FAC
```

```
       LI   R2,ARG
       BL   @TRANSF      SAVE IN ARG
       MOV  R9,R0
       LI   R1,2
       BLWP @NUMREF      GET SUM XJ
       BLWP @XMLLNK
       DATA FMUL         SUMXJ.XUMXI
       LI   R1,FAC
       LI   R2,ARG
       BL   @TRANSF      STORE IN ARG
       LI   R1,NCASES    GET N
       LI   R2,FAC       INTO FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FDIV         DIV (EXI.EXJ)/N
       LI   R1,BUF
       LI   R2,ARG
       BL   @TRANSF      PUT EXI.XJ ARG
       BLWP @XMLLNK
       DATA FSUB         GET SS OR SP
       LI   R1,FAC
       LI   R2,BUF
       BL   @TRANSF
       LI   R1,NCASES
       LI   R2,ARG
       BL   @TRANSF
       LI   R1,ONE
       LI   R2,FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FSUB         N-1
       LI   R1,BUF
       LI   R2,ARG
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FDIV
       MOV  R6,R0
       LI   R1,1
       BLWP @NUMASG      PUT IT BACK
       C    R9,R10
       JNE  CNEXT2
       MOVB @FAC,@FAC
       JNE  SDCOMP
       B    @OVFL      OVERFLOW ERR
SDCOMP BLWP @SQRT
       MOV  R9,R0
       A    R5,R0      COLUMN 2
       LI   R1,2
       BLWP @NUMASG
CNEXT2 INC  R9
       C    R9,R5
```

```
       JGT  CNEXT3
       JMP  CLOOP2       NEXT J
CNEXT3 MOV  R10,R0
       LI   R1,2
       BLWP @NUMREF      GET SUM
       LI   R1,FAC
       LI   R2,ARG
       BL   @TRANSF
       LI   R1,NCASES
       LI   R2,FAC
       BL   @TRANSF      GET N
       BLWP @XMLLNK
       DATA FDIV         DIVIDE BY N
       MOV  R10,R0
       LI   R1,2
       BLWP @NUMASG      PUT MEAN BACK
       INC  R10
       C    R10,R5
       JGT  CNEXT4
       B    @CLOOP1      NEXT I
*
* ROUTINE FOR CORRELATION COEFFICIENTS
*
CNEXT4 LI   R10,1     I = 1 TO R5
CLOOP4 MOV  R10,R9    J = I TO R5
       INC  R9
       MOV  R10,R0    GET SD(I)
       A    R5,R0
       LI   R1,2
       BLWP @NUMREF
       LI   R1,FAC
       LI   R2,BUF
       BL   @TRANSF
CLOOP5 MOV  R9,R6
       MOV  R10,R1
CLOOP6 DEC  R1
       JEQ  CNEXT5
       A    R5,R6     R6 A(I,J) OFFSET
       JMP  CLOOP6
CNEXT5 MOV  R10,R7
       MOV  R9,R1
CLOOP7 DEC  R1
       JEQ  CNEXT6
       A    R5,R7     R7 A(J,I) OFFSET
       JMP  CLOOP7
CNEXT6 MOV  R9,R0     GET SD(J)
       A    R5,R0
       LI   R1,2
       BLWP @NUMREF
       LI   R1,BUF
```

```
       LI   R2,ARG
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FMUL      (SS(J).SS(I))^.5
       LI   R1,FAC
       LI   R2,XBUF
       BL   @TRANSF
       MOV  R6,R0     GET A(I,J)
       LI   R1,1
       BLWP @NUMREF
       LI   R1,FAC
       LI   R2,ARG
       BL   @TRANSF
       LI   R1,XBUF
       LI   R2,FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FDIV     CALC CORR R
       BL   @CHECK
       MOV  R7,R0
       LI   R1,1
       BLWP @NUMASG
       INC  R9
       C    R9,R5
       JGT  CNEXT7
       JMP  CLOOP5
CNEXT7 INC  R10
       C    R10,R5
       JEQ  CRTN
       JMP  CLOOP4
CRTN   B    @RTN
*
*********************************
*
* RETURN TO BASIC ROUTINE
*
*********************************
*
RTN    CLR  R1
       MOVB R1,@STATUS
       LWPI >83E0
       MOV  @SAVE,R11
       B    *R11
*
***************************************
*
* BLWP @SQRT  FLOATING POINT SQUARE ROOT
*
* by John Clulow - New Horizons
*
*
```

```
*     FAC CONTAINS THE NUMBER FOR WHICH
*     THE SQUARE ROOT IS REQUIRED IN
*     RADIX 100 FORMAT
*
*     UPON RETURN FAC CONTAINS THE
*     SQUARE ROOT OF THE NUMBER IN
*     RADIX 100 NOTATION.
*
*************************************
*
SQRT   DATA SRWS
       DATA STSRT
SRWS   BSS  32
BUFR   BSS  8
BUFX   BSS  8
BUFS   BSS  8
TWO    DATA >4002,>0000,>0000,>0000
CRIT   DATA >3B10,>0000,>0000,>0000
STSRT  MOVB @FAC,@FAC IS NUMBER NEGATIVE
       JLT  NEG
       JMP  POS
NEG    LI   R0,BADVAL ISSUE BAD VALUE
       BLWP @ERR
POS    LI   R1,FAC
       LI   R2,ARG   MOVE X TO ARG
       BL   @TRANSF
       LI   R1,FAC
       LI   R2,BUFX  SAVE X IN BUFX
       BL   @TRANSF
       LI   R2,FAC   MOVE TWO TO FAC
       LI   R1,TWO
       BL   @TRANSF
       BLWP @XMLLNK  S=X/2
       DATA FDIV
       BL   @CHECK
       LI   R1,FAC
       LI   R2,BUFS  COPY S TO BUF
       BL   @TRANSF
SRLP   BLWP @XMLLNK  R=X/S
       DATA FDIV
       BL   @CHECK
       LI   R1,FAC
       LI   R2,BUFR  SAVE R IN BUFR
       BL   @TRANSF
       LI   R2,ARG   MOVE S TO ARG
       LI   R1,BUFS
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FSUB
       MOV  @FAC,R1
```

```
       JGT  PDIF
       NEG  R1
       MOV  R1,@FAC
PDIF   LI   R1,FAC
       LI   R2,ARG
       BL   @TRANSF
       LI   R1,BUFR
       LI   R2,FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FDIV
       BL   @CHECK
       LI   R1,CRIT
       LI   R2,ARG
       BL   @TRANSF
       BLWP @XMLLNK    COMPARE S AND R
       DATA FCOMP
       MOVB @STATUS,R1 GET RESULT
       ANDI R1,>4000   CHECK EQ BIT
       JNE  ENDSRT     IF SET STOP
       LI   R1,BUFR
       LI   R2,FAC
       BL   @TRANSF
       LI   R1,BUFS
       LI   R2,ARG
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FADD       S=S+R
       LI   R1,FAC
       LI   R2,ARG     MOVE IT TO ARG
       BL   @TRANSF
       LI   R1,TWO
       LI   R2,FAC     PUT 2 IN FAC
       BL   @TRANSF
       BLWP @XMLLNK
       DATA FDIV       S=S/2
       BL   @CHECK
       LI   R1,FAC
       LI   R2,BUFS    COPY S TO BUF
       BL   @TRANSF
       LI   R1,BUFX
       LI   R2,ARG     PUT X IN ARG
       BL   @TRANSF
       B    @SRLP
ENDSRT LI   R1,BUFR    RELOAD SQRT
       LI   R2,FAC
       BL   @TRANSF
       RTWP            RETURN TO CALLER
*
****************************************
*
```

```
* BL   @GTDIM - GET MAX DIMS 2 DIM ARRAY
*
* R0   PARAMETER # * 8 + 8
*
*****************************************
*
GTDIM  A    @>8310,R0  STACK ADDR
       LI   R1,BUF     CPU BUFFER
       LI   R2,8       GET TABLE
       BLWP @VMBR
       MOV  @BUF+4,R0  GET VALUE ADDR
       LI   R2,4       GET 1st TWO WORDS
       BLWP @VMBR
       B    *R11       RETURN
*
*****************************************
*
* BL   @ERASE - INITIALIZE 2 DIM ARRAY
*               WITH WHATEVER'S IN FAC
*
* R0 - STARTING PARAMETER # (1)
* R1 - LINK PARAMETER #
* R2 - 2nd DIM MAX
* R3 - 1st DIM MAX
*
*****************************************
*
ERASE  MOV  R2,R4
ERLP1  BLWP @NUMASG
       INC  R0
       DEC  R4
       JNE  ERLP1
       DEC  R3
       JNE  ERASE
       B    *R11
*
*****************************************
*
* BL  @TRANSF  RADIX 100 DATA TRANSFER
*
* R1  SOURCE ADDR OF 8 BYTE BUFFER
* R2  DESTIN ADDR OF 8 BYTE BUFFER
*
*****************************************
*
TRANSF LI   R3,4       COUNTER
FACLP  MOV  *R1+,*R2+  MOVE WORD
       DEC  R3         CHECK IF DONE
       JNE  FACLP      IF NOT DO AGAIN
       RT              B *R11 TO CALLER
```

```
*
**************************************
*
* CHECK FOR OVERFLOW IN XMLLNK ROUTINE
*
**************************************
*
CHECK  CB   @>8354,@H1
       JNE  CKRN
       MOVB @H1+1,R0
       BLWP @ERR
CKRN   B    *R11
       END
```

Disk 35. Extended BASIC Games

Version: *Author*:
Requires: XB *Language*: XB *Updated*:

Some of the best games available. Includes Frogger, Extended Golf, Krazy Koala, Air Rescue, King Tut, and many more. Most do NOT require joysticks.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-35
Sectors total           = 360
Sectors used            = 353
Sectors available       = 5
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01b -README       2 DIS/VAR 80   >01c 001
002 >009 AIR-RESCUE   21 PROGRAM      >0e8 020
003 >00c AIRDEFENSE   20 PROGRAM      >13d 019
004 >002 BLKJAK       33 PROGRAM      >022 032
005 >00a CAST-HALLS   29 PROGRAM      >0fc 028
006 >003 CHICKHELP    15 PROGRAM      >042 014
007 >004 CRAZYCLIMB   18 PROGRAM      >050 017
008 >005 EXTENDGOLF   35 PROGRAM      >061 034
009 >006 FROGGER      37 PROGRAM      >083 036
010 >00d KING-TUT     38 PROGRAM      >150 024 >00e 013
011 >007 KRAZYKOALA   40 PROGRAM      >0a7 039
012 >008 RUNWAY180    27 PROGRAM      >0ce 026
013 >00b ZANQUEST     38 PROGRAM      >118 037
```

Disk 35. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

All programs will run in Extended BASIC.

Most do NOT require joysticks.

Disk 36. Speech/Graphics Demos

Version: *Author*:
Requires: XB *Language*: XB, AL *Updated*:

A collection of demo programs that incorporate speech. See ET from Australia, hear Abraham Lincoln read his Gettysburg Address, Ernie and Bert fight, astronauts work on the space shuttle, and Rocky the Robot sing Old MacDonald.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-36
Sectors total           = 360
Sectors used            = 330
Sectors available       = 28
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       3 DIS/VAR 80   >022 002
002 >003 ERNIE&BERT   39 PROGRAM      >024 038
003 >004 ETI          29 PROGRAM      >04a 028
004 >005 FINALE       18 PROGRAM      >066 017
005 >006 LINCOLN      35 PROGRAM      >077 034
006 >007 LOAD          2 PROGRAM      >099 001
007 >008 LOGO         26 PROGRAM      >09a 025
008 >009 OLD-MACDON   23 PROGRAM      >0b3 022
009 >00a SH/1         35 PROGRAM      >0c9 034
010 >00b SH/2         52 INT/VAR254   >0eb 051
011 >00c ST           36 PROGRAM      >11e 035
012 >00d TITLE        32 PROGRAM      >141 031
```

Disk 36. Contents of file -README

Boston Computer Society
TI User Group
One Center Plaza
Boston, MA 02108

All of these programs require that the speech synthesizer be attached in order for them to run, except LOGO.

The LOAD program will run the following programs in X-BASIC:

LOAD
FINALE
SH/1
SH/2
ST
TITLE

LOGO and ERNIE&BERT must be run in Extended BASIC.

ETI, LINCOLM, and OLD-MACDON must be run in TI BASIC with the Terminal Emulator 2 cartridge in place.

Disk 37. Graphics Demos

Version: *Author*:
Requires: XB *Language*: AL, BASIC *Updated*:

The Apesoft bitmap graphics demo — the best TI graphics demo! Visions, an incredible moving color pattern in TI BASIC. Lines, the original bitmap demo from TI with printer dump.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS/JPH/37
Sectors total           = 360
Sectors used            = 169
Sectors available       = 189
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >008 -README       5 DIS/VAR 80   >0e1 004
002 >004 @XBDEMO      47 DIS/FIX 80   >046 046
003 >005 @XBGRAFIC    77 DIS/FIX 80   >074 076
004 >006 APELOADER     2 PROGRAM      >0c0 001
005 >002 LINES        13 DIS/FIX 80   >022 012
006 >003 VISIONS      25 PROGRAM      >02e 024
```

Disk 37. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain disk #37

APELOADER
@XBDEMO
@XBGRAFIC

These three files make up the Apesoft graphics demo from Germany. This program can be run by executing the file "APELOADER" in Extended BASIC. This demo is for a version of Extended BASIC produced by Apesoft that has graphics commands to allow you to great graphics like those in this demo.

LINES

This is a version of the demo program that came with the Mini-Memory Module except that it may be run from the Editor/Assembler cartridge using option 3 (Load and Run). The program name is LINES. Press P to print (I'm not sure what printers will work . . .), Q to quit, and C to lock/unlock colors.

VISIONS. By John B. Priser. January, 1981.

This has become one of the favorite graphics demos of the BCS TI user group. It runs in TI BASIC and produces some incredible graphics effects.

Disk 38. Assembly Language Games

Version: *Author*: TI
Requires: EA *Language*: AL *Updated*:

Three games unreleased by TI: Beyond Parsee, Lasso (requires speech), and Submarine. Also Blitz, a Donkey-Kong type program. All are great fun.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-38
Sectors total           = 360
Sectors used            = 318
Sectors available       = 40
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >008 -README       4 DIS/VAR 80   >156 003
002 >004 BEY-PARSEC   35 DIS/FIX 80 Y >04a 034
003 >006 BLITZA       33 PROGRAM      >11a 032
004 >007 BLITZB       29 PROGRAM      >13a 028
005 >005 LASSO       175 DIS/FIX 80   >06c 174
006 >002 SUB1         33 PROGRAM      >022 032
007 >003 SUB2          9 PROGRAM      >042 008
```

Disk 38. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain disk #39

BEY-PARSEC

Beyond Parsec. 2-player game. Load with Editor/Assembler option 3. The program name is OMEGA.

BLITZA

Load with Editor/Assembler option 5 or TI-Writer option 3. Difficult game. You have to get a cannon ball to fall on a see-saw that you are on to get to the next level. Either that or grab a balloon. Not easy.

LASSO

Load with Editor/Assembler option 3. Speech synthesizer MUST be plugged in for this one to work.

SUB1

Submarine game. Load from Editor/Assembler option 5 or TI-Writer option 3.

Disk 39. Forth Tutorial

Version:
Author: Howie Rosenberg
Requires: EA, Forth Disk
Language: Forth
Updated:

An excellent lesson on programming in Forth. Explains very completely how to create graphics and how to do It fast. Comes with several Forth "words" including one to draw cycloids in bitmap.

```
dskdir. v2.0. 12-dec-96
Disk name               = FORTH*V8
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = yes
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 FORTH         6 DIS/FIX 80 Y >022 005
002 >003 FORTHSAVE    39 PROGRAM    Y >027 038
003 >004 SYS-SCRNS   313 DIS/FIX128 Y >04d 283 >005 029
```

Disk 39. Contents of Forth disk ASCII screens

```
Screen 035 ----------------------------------------------------------

00   ( READTHIS FILE FOR DOODLES PG 1)  CLS 0 0 GOTOXY
01   ."  Dedicated to Ron albright "         CR
02   ."  We've learned from each other but"  CR
03   ."  the greatest lesson he's taught me" CR
04   ."  is in caring and sharing."          CR CR
05   ."  These demonstrations in themselves " CR
06   ."  serve no useful purpose. They are "  CR
07   ."  intended as a learning tool for the" CR
08   ."  novice and more experienced "        CR
09   ."  programmer who can't or hasn't got"  CR
10   ."  started to program in this exciting" CR
11   ."  language. Experienced FORTH users"   CR
12   ."  will most likely look at the demo "   CR
13   ."  once and put the disk in the back "   CR
14   ."  of the box. The novice hopefully"    CR
15   ."  will read this text, and the text"   CR    -->

Screen 036 ----------------------------------------------------------

00   ( READTHIS for DOODLES pg2)
01   ."  file DETAILS. If some of the users" CR
02   ."  are encouraged in learning FORTH "    CR
03   ."  then my effort has indeed been "      CR
04   ."  worthwhile."  1000 DELAYS CLS 0 0 GOTOXY
05   0 0 GOTOXY
06   ."  DOODLES Copyright (C) 1985" CR
07   ."  Howie Rosenberg"            CR
08   ."  19 7TH Ave"                 CR
09   ."  Farmingdale N.Y. 11735"     CR
10   ."  CompuServe I.D. 74216,1640" CR
11   ."  All users are licensed and" CR
12   ."  encouraged to GIVE copies of this"   CR
13   ."  disk to anyone who desires it.   "   CR
14   ."  User Groups are encouraged to put"   CR
15   ."  it in their libraries provided   "   CR  -->

Screen 037 ----------------------------------------------------------

00   ( READTHIS for DOODLES pg3)
01   ."  that FREE access be allowed to all"  CR
02   ."  members at NO charge inclusive of"   CR
03   ."  'service' charges." CR CR
04   ."  NOTE: To print out this file or "    CR
05   ."  the DETAILS file, first load -PRINT" CR
06   ."  from your master disk. The sequence" CR
07   ."  SWCH READIT (or DETAILS) UNSWCH    " CR
```

```
08   ."  will then print out the file"        CR
09   ."  Any questions, comments, or    "     CR
10   ."  suggestions, please contact me,    " CR
11   ."  either at the address given, or "    CR
12   ."  via C.I.S."  1200 DELAYS CLS 0 0 GOTOXY
13
14
15             -->

Screen 038 ----------------------------------------------------------

00   ( READTHIS pg4)
01   ."  BIT PLOT MODE--overview " CR
02   ."  NOVICES NOTE this is only a "                CR
03   ."  philosophical background. You can"           CR
04   ."  begin to 'use' this disk with little"        CR
05   ."  understanding in this area."                 CR CR
06   ."  There are generally 3 ways to create"       CR
07   ."  pictures in BIT PLOT. One can plot "        CR
08   ."  points on paper, Calculate the codes "      CR
09   ."  of memory locations corresponding to"       CR
10   ."  these points and 'move' the BYTES"          CR
11   ."  required into desired locations in"         CR
12   ."  the pattern descriptor table. The"          CR
13   ."  procedure is tedious but results can"       CR
14   ."  be rewarding"                               CR  1000 DELAYS
15   CLS 0 0 GOTOXY     -->

Screen 039 ----------------------------------------------------------

00   ( READTHIS pg 5)
01   ."  The second method consists of using"         CR
02   ."  an external source to define the"           CR
03   ."  locations of ' ON dots'. Arrow Keys,"        CR
04   ."  Joystick, Sketchpads or Lightpens all"       CR
05   ."  can be used to locate desired ON dots"       CR
06   ."  The conversion from position to bit"         CR
07   ."  in memory is nandled in FORTH rather"        CR
08   ."  easily."                          1200 DELAYS CLS 0 0 GOTOXY
09   ."  The third method is to define"               CR
10   ."  functions and equations, and perform"        CR
11   ."  calculations to plot the functions"          CR
12   ."  In short let the machine do the "            CR
13   ."  hard work that it was designed to do"        CR
14   ."  This is the method that this program"        CR
15   ."  attempts to illustrate."                     CR   -->

Screen 040 ----------------------------------------------------------

00   ( READTHIS PG6)
```

```
01   ."  The -GRAPH option in T.I. FORTH have"      CR
02   ."  the words DOT and LINE which actually"     CR
03   ."  are basic words of the method 3 kind"      CR
04   ."  DOT places a dot at the specified"         CR
05   ."  row-column location, while LINE"           CR
06   ."  draws a line from the specified"           CR
07   ."  row-column start point to the row"         CR
08   ."  column end point specified. This is"       CR
09   ."  accomplished by calling the DOT"           CR
10   ."  word repetitively, while calculating"      CR
11   ."  position for each dot"           1600 DELAYS CLS 0 0 GOTOXY
12   ."  The demonstrations merely are an"          CR
13   ."  extension of these two words. The"         CR
14   ."  words added make it convenient to"         CR
15   ."  'draw' some rather intricate designs"      CR   -->

Screen 041 ------------------------------------------------------------

00   ( READTHIS PG 7)
01   ."  The additional words created are "        CR
02   ."  described in the file DETAILS. Rather"    CR
03   ."  then creating an applications program"    CR
04   ."  i.e. menu driven prompts etc,"            CR
05   ."  extending the demonstration requires"     CR
06   ."  that the user use FORTH syntax."          CR
07   ."  By using simply the words already"        CR
08   ."  defined, some interesting graphics"       CR
09   ."  can be developed. By extending the"       CR
10   ."  vocabulary, the possibilities are"        CR
11   ."  limited only by your imagination"   1200 DELAYS  CLS
12   0 0 GOTOXY
13   ."  The screen in BIT map mode consists"      CR
14   ."  of 256 columns and 192 rows. A dot"       CR
15   ."  at each location on the screen can"       CR   -->

Screen 042 ------------------------------------------------------------

00   ( READTHIS Pg 8)
01   ." be turned 'on or off' by changing the"      CR
02   ." BYTE which holds the ON/OFF information"    CR
03   ." for that pixel (dot) and 7 others. Just"    CR
04   ." as one does in assembly. All that one "     CR
05   ." really has to be aware of in using"         CR
06   ." FORTH in this application is that"          CR
07   ." the column-row arrangement of the"          CR
08   ." screen. 10 20 DOT places a dot at"          CR
09   ." column 10 row 20. All else(I hope)"         CR
10   ." will follow." 1000 DELAYS  CLS 0 0 GOTOXY
11   menu
12
13
```

```
14
15

Screen 071 ----------------------------------------------------------

00   ( DETAILS pg 2 )
01   ."  NOTE: The demo is saved at screen 20"           CR
02   ."  as a binary image. The screens which"          CR
03   ."  comprise the demo start at screen 55"          CR
04   ."  text files READTHIS, and DETAILS are"          CR
05   ."  located at screens 35 and 70"               CR CR
06   ."  HOW TO ADD CODE OR FOOL AROUND"             CR CR
07   ."  All the Demo's are executed in BIT"         CR
08   ."  MAP mode. Executing the word"                  CR
09   ."  INITGRAPH2, enters Bit Map mode"               CR
10   ."  ( GRAPH2). Although you can't  see it"         CR
11   ."  text from the text buffer can be"              CR
12   ."  entered and executed while in GRAPHICS2"       CR
13   ."  Thus entering INITGRAPH2 followed by"          CR
14   ."  a sequence of executable words will"           CR
15   ."  execute. If you ever get stuck simply"         CR  -->

Screen 072 ----------------------------------------------------------

00   ( DETAILS PG 3)
01   ."  enter the FORTH word TEXT to return"          CR
02   ."  to TEXT mode. Fooling around this"            CR
03   ."  way is a good way to get ideas. The"          CR
04   ."  garbage on the screen occurs each"            CR
05   ."  time FORTH tries to leave a message"          CR
06   ."  while in GRAPHICS2."   CR   1200 DELAYS CLS 0 0 GOTOXY
07   ."  As an alternate, a series of words"           CR
08   ."  can be typed(up to two rows) and then"        CR
09   ."  entered. i.e. typing INITGRAPH2 20 20"        CR
10   ."  30 60 LINE 500 DELAYS TEXT, then"             CR
11   ."  pressing the enter key, will enter"           CR
12   ."  GRAHICS2 draw the specified line,"            CR
13   ."  and after a delay return to TEXT"             CR
14   ."  mode. This method of entry cannot"            CR
15   ."  be used for loops."  CR     -->

Screen 073 ----------------------------------------------------------

00   ( DETAILS pg 4)
01   ."  You can always define new colon "           CR
02   ."  words directly in command mode"             CR
03   ."  For more permanent applications,"           CR
04   ."  or extension of the demo, you'll"           CR
05   ."  have to use the editor(not resident"         CR
06   ."  in the application, and another "           CR
```

```
07   ."  FORTH disk for development. The"            CR
08   ."  screens starting at 55 should be"          CR
09   ."  (at least to the extent you require)"      CR
10   ."  compiled as part of the new"               CR
11   ."  application and then BSAVED"               CR
12   1200 DELAYS CLS 0 0 GOTOXY
13   ."  A WORD ABOUT DOCUMENTING FORTH WORDS"      CR CR
14   ."  Because of the necessity for having"       CR
15    -->

Screen 074 ------------------------------------------------------------

00   ( DETAILS pg 5)
01   ."  The numbers required by a FORTH word"       CR
02   ."  prior to execution, and the necessity"      CR
03   ."  to keep track of the stack, standard"       CR
04   ."  FORTH notation is to place the before"      CR
05   ."  after stack condition as a remark as"       CR
06   ."  part of the word definition. Thus"          CR
07   ."  : SAMPLEWORD ( row column- flag) "          CR
08   ."  OLDWORD1 OLDWORD2 ;  is the   "             CR
09   ."  FORTH definition of a word "                CR
10   ."  SAMPLEWORD consisting of two "              CR
11   ."  previously defined words and "              CR
12   ."  requiring two numbers on the stack"         CR
13   ."  prior to execution, row, and column."       CR
14   ."  After the word is executed a flag "         CR
15   ."  remains on the stack." 1200 DELAYS CLS 0 0 GOTOXY -->

Screen 075 ------------------------------------------------------------

00   ( DETAILS pg 6)
01   ."  I've attempted to be as descriptive"          CR
02   ."  as possible on the screens as to the "        CR
03   ."  inputs for the words, and hopefully"          CR
04   ."  with some additional information"             CR
05   ."  in this text, their use will be "             CR
06   ."  quite clear. In addition I've chosen"         CR
07   ."  descriptive names for the new FORTH"          CR
08   ."  words which make up the demo. Only "          CR
09   ."  those words which are used in the "           CR
10   ."  final demo are described here. Some "         CR
11   ."  of the lower level words, which were"         CR
12   ."  used as a foundation for these words"         CR
13   ."  are not described, except for a few"          CR
14   ."  cases in which an explanation is in"          CR
15   ."  order. Thus ONEROT which converts  "          CR -->

Screen 076 ------------------------------------------------------------

00   ( DETAILS pg 7)
```

```
01   ."  angles to less then 360 degrees is"           CR
02   ."  used by several routines but not"             CR
03   ."  described herein"     1200 DELAYS CLS 0 0 GOTOXY
04   ."  A WORD ABOUT TRANSCENDENTALS"               CR CR
05   ."  A number of the words used herein"           CR
06   ."  utilize SIN and COS functions. While"        CR
07   ."  FORTH is a rather fast language"             CR
08   ."  console GPLLNK routines are slooow"          CR
09   ."  The Floating point routines in T.I."         CR
10   ."  FORTH make extensive use of GPLLNK."         CR
11   ."  If you want to do something fast, "          CR
12   ."  it's better to find an alternate "           CR
13   ."  approach. In this case I've set up"          CR
14   ."  a SIN table. During compile the "            CR
15   ."  table is filled by the word FILLTAB."   CR   -->

Screen 077 ----------------------------------------------------------

00   ( DETAILS pg 8)
01   ."  Note that FILLTAB makes widespread use"     CR
02   ."  of Floating point routines. It's only"      CR
03   ."  use is during compile, the application"     CR
04   ."  being saved as an image. If you go "        CR
05   ."  through the process you will find that"     CR
06   ."  it takes a minute or so to draw"            CR
07   ."  ONE CIRCLE using repeated calls to"         CR
08   ."  float routines. Perhaps a hardware"         CR
09   ."  floating point package from one of"         CR
10   ."  our hardware developers?" CR 1200 DELAYS CLS 0 0 GOTOXY
11   ."  POSITION ON SCREEN: A number of words"      CR
12   ."  make use of variables XCENTER, and"         CR
13   ."  YCENTER as well as constants XSHIFT"        CR
14   ."  and YSHIFT. The original intent was"        CR
15   ."  to use the constants strictly to set"       CR   -->

Screen 078 ----------------------------------------------------------

00   ( DETAILS pg 9)
01   ."  up a bias to allow use in both "         CR
02   ."  GRAPHICS2 and SPLIT modes. This"         CR
03   ."  has been somewhat corrupted and"         CR
04   ."  I trust doesn't lead to too much"        CR
05   ."  confusion. As a result one should"       CR
06   ."  check the location of center "           CR
07   ."  after execution of a routine,"           CR
08   ."  and set the center for the next"         CR
09   ."  routine using MIDCENTER,      "          CR
10   ."  TOPLCENTER etc. (see screen 61) "        CR
11   ."  COLORS: Some words have been added"      CR
12   ."  to allow rapid line color changes"       CR
```

```
13   ."  one shade of a number of colors "        CR
14   ."  have been chosen as words BLUE etc."     CR
15   ."  (screen 62). "   CR  1200 DELAYS CLS 0 0 GOTOXY  -->

Screen 079 ----------------------------------------------------------

00   ( DETAILS pg 10)
01   ."  DELAYS: The word( number-) DELAYS "        CR
02   ."  sets up a time delay based on number"      CR
03   ."  WIPES: The words WIPEUP, WIPEDOWN and"     CR
04   ."  WIPEFAST delay and clear the screen."      CR CR
05   ."  CIRCLES: The word ( radius xcenter"        CR
06   ."  ycenter) CIRCLE draws a circle "           CR
07   ."  displaced from X,Y=0 by xcenter"          CR
08   ."  and ycenter. X,Y,=0 is defined by"        CR
09   ."  XSHIFT,YSHIFT thus TOPLCENTER which"      CR
10   ."  changes these values, places the "        CR
11   ."  coordinate system, centered in the"       CR
12   ."  top left quadrant. "                      CR
13   ."  NOTE: THERE IS NO BOUNDARY CHECKING"      CR
14   ."  FOR CIRCLES OR ANY OTHER WORDS.    "      CR
15   ."  What this means is that if you   "        CR -->

Screen 080 ----------------------------------------------------------

00   ( DETAILS pg 11)
01   ."  choose too large a radius, you will"      CR
02   ."  write to a location in video memory"      CR
03   ."  outside the screen map. Don't worry"      CR
04   ."  you'll recognize the goof, and will"      CR
05   ."  most likely have to reboot"  CR 1200 DELAYS CLS 0 0 GOTOXY
06   ."  CYCLOIDS: Many of the illustrations"      CR
07   ."  make use of EPICYCLOIDS and"              CR
08   ."  HYPOCYCLOIDS. These curves are "          CR
09   ."  defined as the locus of a point"          CR
10   ."  on a circle revolving outside(EPI)"       CR
11   ."  or inside(HYPO) another circle."          CR
12   ."  Epicycloids may have either radius"       CR
13   ."  larger. The inner radius of"              CR
14   ."  Hypocycloids must be the larger."         CR
15   ."  If the rolling radius is an integer"      CR  -->

Screen 081 ----------------------------------------------------------

00   ( DETAILS pg 12)
01   ."  submultiple of the fixed radius"       CR
02   ."  the curve closes in one rotation"     CR
03   ."  and stops, otherwise it continues"    CR
04   ."  until it does close. both words"      CR
05   ."  require fixed radius, and rolling"    CR
06   ."  radius on the stack. The center of"   CR
```

```
07   ."  the CYCLOID is controlled by the "     CR
08   ."  position words previously described"   CR
09   ."  Two additional Cycloid words used"     CR
10   ."  are ROTEPIS and ROTHYPOS."             CR
11   ."  ( radius1 radius2 angle increment"     CR
12   ."  A series of curves are drawn, each"    CR
13   ."  rotated by angle increment." CR 1200 DELAYS CLS 0 0 GOTOXY
14   ."  ARC AND RELATED WORDS. "               CR
15   ."  Screen 56 contains the word ARC"       CR   -->

Screen 082 ----------------------------------------------------------

00   ( DETAILS pg 13)
01   ."  which may be used to draw arcs at"          CR
02   ."  any location on the screen by "             CR
03   ."  storing end angle, start angle, "           CR
04   ."  radius, xcenter and ycenter on the"         CR
05   ."  stack. End angle and start angle"           CR
06   ."  must not cross 0. Break it up into"         CR
07   ."  two arcs. I was somewhat in error"          CR
08   ."  in defining the circular functions"         CR
09   ."  as the mathematicians among you have"       CR
10   ."  by now noticed. In consequence 0 "          CR
11   ."  degrees is on the negative Y axis"          CR
12   ."  This seemed appropriate for an infix"       CR
13   ."  language. I hope it does not cause"         CR
14   ."  too much confusion. The word calcloc"       CR
15   ."  enables locating the end points of"         CR -->

Screen 083 ----------------------------------------------------------

00   ( DETAILS pg 14)
01   ."  an arc, quite convenient in drawing"        CR
02   ."  a PIECHART as illustrated on screen"        CR
03   ."  68. This word is used to implement"         CR
04   ."  STAR1 which in turn is used in the "        CR
05   ."  random star patterns (screen 67"            CR
06     1200 DELAYS     CLS 0 0 GOTOXY
07   ."  THE DIAMOND WORDS:"
08   ."  Screen 65 illustrates the buildup"          CR
09   ."  of a few simple patterns strictly"          CR
10   ."  based on the FORTH resident word "          CR
11   ."  LINE. The word DIAMOND creates a "          CR
12   ."  diamond pattern of specified size"          CR
13   ."  anyplace on the screen. As always"          CR
14   ."  if the stack input places the dot"          CR
15   ."  outside the map, the results are "          CR  -->

Screen 084 ----------------------------------------------------------
```

```
00   ( DETAILS pg 15)
01   ."   unpredictable."                            CR CR
02   ."   PARABOLAS AND SINES."
03   ."   Screens 63 and 64 illustrate the"         CR
04   ."   ease of generating these curves in"       CR
05   ."   either a design or math application"      CR
06   ."   In a math application, the curves "       CR
07   ."   are easily combined with the "            CR
08   ."   DRAWAXIS and ADDMARKS words of"           CR
09   ."   screen 62. Of course ADDMARKS "           CR
10   ."   would be changed to an appropriate"       CR
11   ."   scale."   CR     1200 DELAYS CLS 0 0 GOTOXY
12   ."   SOME FINAL REMARKS"                       CR  CR
13   ."   FORTH is an individual language."         CR
14   ."   There are many ways of tackling"          CR
15   ."   a problem. Start 10 programmers"          CR  -->

Screen 085 ----------------------------------------------------------

00   ( DETAILS pg 16)
01   ."   on a set of tasks in ANY language"        CR
02   ."   and more then likely you'll have 10"      CR
03   ."   solutions. Do so in FORTH and you'll"     CR
04   ."   have 10 new languages. Basic in this"     CR
05   ."   language is the ability to make it "      CR
06   ."   yours. In this vein, the structure"       CR
07   ."   of this 'application' is not one to"      CR
08   ."   'use' in the sense that one 'uses' "      CR
09   ."   say a graphics package written in"        CR
10   ."   BASIC but as a source of ideas on"        CR
11   ."   which to build your own FORTH."           CR
12   1000 DELAYS CLS 0 0 GOTOXY menu
13
14
15
```

Disk 40. C Tutorial

Version: 3.0 *Author*: Donald Mahler
Requires: EA *Language*: c99 *Updated*: 10/28/86

This disk contains many C programming examples of various levels of complexity. It explains how to create C functions, how to save C programs in Program Image format, and shows the steps involved in developing C programs. Also includes a set of functions to allow you to use strings in C. Requires c99 (disks 27 and 45).

```
dskdir. v2.0. 12-dec-96
Disk name               = C-TUTOR:V3
Sectors total           = 360
Sectors used            = 347
Sectors available       = 11
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = yes
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README      15 DIS/VAR 80 Y >022 014
002 >003 ADD10C        4 DIS/VAR 80 Y >030 003
003 >004 ARR12C        5 DIS/VAR 80 Y >033 004
004 >005 ARR1C         3 DIS/VAR 80 Y >037 002
005 >006 ARR2C         3 DIS/VAR 80 Y >039 002
006 >007 ARR3C         5 DIS/VAR 80 Y >03b 004
007 >008 CD3C          4 DIS/VAR 80 Y >03f 003
008 >009 CDC           5 DIS/VAR 80 Y >042 004
009 >00a CDO          14 DIS/FIX 80 Y >046 013
010 >00b CDS          33 DIS/VAR 80 Y >053 032
011 >00c CFDC          5 DIS/VAR 80 Y >073 004
012 >00d CHANGE        2 DIS/VAR 80 Y >077 001
013 >00e CPOS          2 DIS/VAR 80 Y >078 001
014 >00f CPOS;C        4 DIS/VAR 80 Y >079 003
015 >010 CSEG          3 DIS/VAR 80 Y >07c 002
016 >011 CSEG;C        4 DIS/VAR 80 Y >07e 003
017 >012 CSEG;O       15 DIS/FIX 80 Y >081 014
018 >013 CSORT         3 DIS/VAR 80 Y >08f 002
019 >015 CTITLE       20 PROGRAM    Y >09e 019
020 >016 DCHANGE      12 PROGRAM    Y >0b1 011
021 >017 DISPLAYC      5 DIS/VAR 80 Y >0bc 004
022 >018 GETINT        4 DIS/VAR 80 Y >0c0 003
023 >019 INDEX         2 DIS/VAR 80 Y >0c3 001
024 >01a INTDEMC       4 DIS/VAR 80 Y >0c4 003
025 >01b MAX           2 DIS/VAR 80 Y >0c7 001
```

```
026 >01c MIN           2 DIS/VAR 80 Y >0c8 001
027 >01d MODROM        7 PROGRAM    Y >0c9 006
028 >01e MODROMC       4 DIS/VAR 80 Y >0cf 003
029 >01f PMK2C         4 DIS/VAR 80 Y >0d2 003
030 >020 PMK2O        10 DIS/FIX 80 Y >0d5 009
031 >0ee PRF           2 DIS/VAR 80 Y >0ed 001
032 >0f2 PRIM2C        4 DIS/VAR 80 Y >0ef 003
033 >0fc PRIM2O       10 DIS/FIX 80 Y >0f3 009
034 >112 PRIM2S       22 DIS/VAR 80 Y >0fd 021
035 >11c PRIME        10 PROGRAM    Y >113 009
036 >11f RK3C          3 DIS/VAR 80 Y >11d 002
037 >124 SCINC         5 DIS/VAR 80 Y >120 004
038 >127 SCPDEM;C      3 DIS/VAR 80 Y >125 002
039 >12b SIMP3C        4 DIS/VAR 80 Y >128 003
040 >131 SIMP3O        6 DIS/FIX 80 Y >12c 005
041 >135 SLDM;C        4 DIS/VAR 80 Y >132 003
042 >014 SPTST;C       5 DIS/VAR 80 Y >091 004
043 >021 SPTST;O      19 DIS/FIX 80 Y >095 009 >0de 009
044 >137 STNCPY        2 DIS/VAR 80 Y >136 001
045 >13b STNCPYDMC     4 DIS/VAR 80 Y >138 003
046 >13f STOI          4 DIS/VAR 80 Y >13c 003
047 >141 STRCAT        2 DIS/VAR 80 Y >140 001
048 >144 STRCMP        3 DIS/VAR 80 Y >142 002
049 >146 STRCPY        2 DIS/VAR 80 Y >145 001
050 >14a STRLEN        4 DIS/VAR 80 Y >147 003
051 >14e STRPOS;C      4 DIS/VAR 80 Y >14b 003
052 >155 STRPOS;O      7 DIS/FIX 80 Y >14f 006
053 >159 STRPS;C       4 DIS/VAR 80 Y >156 003
054 >15c TESTMXC       3 DIS/VAR 80 Y >15a 002
055 >15e TSEGC         2 DIS/VAR 80 Y >15d 001
056 >162 TSTGIC        4 DIS/VAR 80 Y >15f 003
```

Disk 40. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

c-tutorial

This disk contains a number of simple c programs. Programs ending in 'c' are c.code, those ending in 's' are compiled source.code, and those ending in 'o' are assembled object code. (Most of the source codes have been eliminated to save space, but can easily be compiled as practice).On the other hand, you can use the supplied s.codes to test your assembling technique, and the o.codes to immediately try "LOAD and RUN". Those few without these designations are either short library functions or longer program files ready to run in opt. 5 of Editor/Assembler. (*not* Funlwriter opt. 3!)

Some of these programs have been modified from texts to run in c99 and others are original. This is the 2nd update of an earlier tutorial; this disk uses v2.0 of Clint Pulley's c99.

One of the advantages of c is its ability to use a "library" of functions in programs by the statement, for example, "#include dsk1.roman". With two disk drives, it is most convenient to have the "library disk" in drive 2, and use statement "#include dsk2.roman". With only one drive, the library functions (from c99 and this disk) should be copied onto a blank disk, which is then used for compiling your program, because the library must be available during compiling. (change references to #include dsk1.****)

The usual procedure for writing a c program in c99:

1. Write the program, using Editor and SAVE as "-----c"

2. Compile using c99c and the library to form "-----s"

3. Assemble to form "-----o"

4. "load and run" your object code, 'csup'(from c99 dsk), plus any obj codes used (e.g. grf1);program name is "START"

5. If desired, convert to program format by "load and run":

    a. c99pfi (from c99 disk)
    b. your object code
    c. csup
    d. other needed obj codes (e.g. PRINTF)
    e. c99pff (from c99 disk)
    f."SAVE" (on Editor/Assembler B)

Included on this disk are the library functions:

1. change
2. conv;c
3. csort
4. cpos
5. cseg
6. max
7. min
8. prf
9. rom
10. stoi
11. strcat
12. stncpy
13. strlen

c.codes (some examples):

prim2c	find all primes below1000
pmk2c	tests integer for primeness
modromc	prints 1-100 in roman numbers
rk3c	changes number to roman using function rom
scinc	totals your change
simp2c	simple string manipulation
testmxc	test max function
tstgic	tests getint funct
add10c	does 10 additions
cseg;c	equivalent to basic SEG$
strpos;c	equiv to basic POS
intdemc	getint demo
arr1c	array programs
arr2c	
arr3c	
arr12c	
stncpydmc	demo of stncpy

Finally, there is a progressive set of "change for a dollar" programs, showing my steps in writing it.

cfdc	is the original program
cdc,cds,cdo	uses function, quits if 100 entered
cd3c	uses library function
dchange	program format (E/A 5)

D.L. Mahler, 36 Boulder Road, Newton, MA 02159

Disk 40. Contents of file ADD10C

```
/* add10c    */
/* PROGRAM WITH KEYBOARD INPUT */
/* ALLOWS 10 ADDITIONS   */
#include dsk2.conv;c
main()
  {int i;
   char str[10];
   char buff[10];
   int sum,item1,item2;
   i=0;
   while (++i<=10) /*loop counter*/
{  sum=item1=item2=0;
   /* clears slate for each add */
   puts("enter a number: ");
   item1=atoi(gets(buff));
/* gets number from keyboard and
   changes it to int  */
   puts("second number: ");
   item2=atoi(gets(buff));
   sum=item1+item2;
   itod(sum,str,10);
/* changes sum to char 'str' */
   puts(str);
   putchar ('\n');/* new line */
                }
putchar ('\n');
puts("end of program");}
```

Disk 40. Contents of file ARR12C

```
/* test of array  arr12c   */
#define S 10
/* this lets us change size of array
  by changing single number in code-
  Note that 'S' appears several times*/
#include dsk2.prf
 /* in library files   */
#include dsk2.csort
 /* in library files   */
#include dsk2.conv;c
 /* in library files   */
main()
{ char buff[80];
 /* Room for BIG numbers ! */
  int val[S]; int a,b,c,d;
  a=c=d=0; b=1; /* b is flag */

  while (++d<=S) /* increment d each
                    loop! */
  { printf("\n enter item #%d:",d);
    val[d]=atoi(gets(buff)); }
/* change string to integer  */
  printf("here is original list\n");
  d=0;  /* so we can start d at 1 */
  while (++d<=S)
    { printf("item#%d = %d \n",d,
    val[d]);}
  sort(a,b,c,d,val);  /* see below*/
  puts("\n end of sort program"); }
```

Disk 40. Contents of file ARR1C

```
/* sum an array  arr1c  */
# include dsk2.prf
main()
  { int sum; int item[6],i;
   i=0; item[0]=2;
   item[1]=4; item[2]=6;
   item[3]=8; item[4]=10;
   item[5]=5;
   sum=0 ;

   while (i<6)
  {printf("item no.  %d is
   %d \n",i,item[i]);
  /*  %d  means decimal
value of variable at end of
phrase; in this case i and
later item[i] */

sum =sum + item[i++];}
/* increment i each loop */
printf("\n sum is
%d\n",sum);
 }
```

Disk 40. Contents of file ARR2C

```
/* test of array  arr2c     */
#include dsk2.prf
#include dsk2.conv;c
main()
{ int sum;  char buff[80];
  int item[6],i; i=0; sum=0;
  while(i++<6)
  {printf("\n enter item no. %d :"
     ,i);
    item[i]=atoi(gets(buff));
/* this converts keyboard value
 to integer stored as item[i] */
    printf("\n item no. %d is %d
      \n",i,item[i]);
    sum=sum + item[i];}

  printf("\n sum is %d\n",sum); }
```

Disk 40. Contents of file ARR3C

```
/* test of array  arr3c    */
#include dsk2.prf

main()
{
  int val[10]; int a,b,c,d ;
  val[0]=2;   a=c=d=0; b=1;
  val[1]=4;
  val[2]=6;
  val[3]=8;
  val[4]=10;
  val[5]=3;
  val[6]=17;
  val[7]=11;
  val[8]=5;
  val[9]=0;

  while (d<=9)
 { printf("\n item# %d= %d",d,
 val[d]);  d=d+1; }

  sort(a,b,c,d,val);
/* a,b,c,d,and val are the ACTUAL
 arguments needed by 'sort' */
  puts("\n end of sort program"); }

  sort(i,f,s,k,item)
/* i,f,s,k,and item are the FORMAL
 arguments used by 'sort'  */
  int i,f,s,k,item[];
    { puts("\n \n here is your
       sorted list!");
      while (f==1)
      { i=f=s=k=0;
       while(i<=9)
       { if (item[i]<= item[i+1])
        { s= item[i]; f=1;
          item[i]=item[i+1];
          item[i+1]=s; }
       i=i+1;  }}
      while(++k<=10)
  { printf("\n item# %d= %d",k,
     item[k]);} }
/* when finished with sort, we
go back to next line of 'main'
i.e. the 'puts' line    */
```

Disk 40. Contents of file CD3C

```
/* change for a dollar       cd3c  */

#include dsk2.conv;c
#include dsk2.change
main()
 { int c,p,h,q,d,n,s;
   char buff[4];
   char hs[2],qs[2],ds[2],ns[2],ss[2];
   while(p !=100)
  {puts("enter price in cents:\n");
   p=atoi(gets(buff));
   c=100-p;
   putchar('\n');
  c= change(c,50,h,hs," half-dollar");
  c= change(c,25,q,qs," quarters");
  c= change(c,10,d,ds," dimes");
  c= change(c,5,n,ns," nickels");
  c= change(c,1,s,ss," pennies");

 putchar('\n');
 puts("that's your change! \n"); }
 putchar('\n');
 puts("have a nice day!"); }
```

Disk 40. Contents of file CDC

```
/* change for a dollar       cdc  */
/* uses function 'change()' */
#include dsk2.conv;c
main()
 { int c,p,h,q,d,n,s;
   char buff[4];
   char hs[2],qs[2],ds[2],ns[2],ss[2];
   puts("enter price in cents:\n\t");
   p=atoi(gets(buff));
   c=100-p;
   putchar('\n');
  c= change(c,50,h,hs," half-dollar");
  c= change(c,25,q,qs," quarters");
  c= change(c,10,d,ds," dimes");
  c= change(c,5,n,ns," nickles");
  c= change(c,1,s,ss," pennies");
/* main keeps calling 'change' with
   different ACTUAL arguments */
 putchar('\n');
 puts("that's your change! \n"); }

change(i,j,k,ks,str)
/* i,j,k,ks,str are FORMAL arguments */
  int i,j,k;
  char ks,str;
 { if    (i>=j)
   {k = i/j;
    i = i % j;
   itod(k,ks,2);
   puts(ks);
   puts(str );
   putchar('\n');}
   return(i);   }
/* so it can be used for next run
   of 'change()' */
```

Disk 40. Contents of file CDS

```
*c99 v2.0 (c) 1986 Clint Pulley
 REF C$CIND,C$DIV,C$REM,C$ASR,C$ASL,C$EQ,C$NE,C$LT,C$LE
 REF C$GT,C$GE,C$ULT,C$ULE,C$UGT,C$UGE,C$LNEG,C$SWCH
 REF GETCHA,GETS,PUTCHA,PUTS,LOCATE,POLL,TSCRN,EXIT
*/* change for a dollar       cdc  */
*/* uses function 'change()' */
*#include dsk2.conv;c
*/* simple conversion functions */
*/*
*** n=atoi(s) - convert string to integer
**/
*atoi(s)  char *s;
ATOI
*{ int sign,n;
*  while(*s==' ')++s;
 AI 14,-4
C$2
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
 SRL 8,8
 DECT 14
 MOV 8,*14
 LI 8,32
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$3
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
 B @C$2
C$3
*  sign=1;
 MOV 14,8
 INCT 8
 DECT 14
 MOV 8,*14
 LI 8,1
 MOV *14+,9
 MOV 8,*9
*  if(*s=='-') { sign=-1; ++s; }
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
```

```
 SRL 8,8
 DECT 14
 MOV 8,*14
 LI 8,45
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$4
 MOV 14,8
 INCT 8
 DECT 14
 MOV 8,*14
 LI 8,1
 NEG 8
 MOV *14+,9
 MOV 8,*9
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
*  if(*s=='+') ++s;
C$4
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
 SRL 8,8
 DECT 14
 MOV 8,*14
 LI 8,43
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$5
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
*  n=0;
C$5
 MOV 14,8
 DECT 14
 MOV 8,*14
 CLR 8
 MOV *14+,9
 MOV 8,*9
*  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
C$6
 MOV 14,8
 AI 8,6
```

```
MOV *8,8
MOVB *8,8
SRL 8,8
DECT 14
MOV 8,*14
LI 8,48
BL @C$GE
DECT 14
MOV 8,*14
MOV 14,8
AI 8,8
MOV *8,8
MOVB *8,8
SRL 8,8
DECT 14
MOV 8,*14
LI 8,57
BL @C$LE
INV *14
SZC *14+,8
ABS 8
JNE $+6
B @C$7
MOV 14,8
DECT 14
MOV 8,*14
LI 8,10
DECT 14
MOV 8,*14
MOV 14,8
AI 8,4
MOV *8,8
MOV 8,7
MPY *14+,7
DECT 14
MOV 8,*14
MOV 14,8
AI 8,10
MOV *8,9
INC *8
MOV 9,8
MOVB *8,8
SRL 8,8
A *14+,8
DECT 14
MOV 8,*14
LI 8,48
S *14+,8
NEG 8
MOV *14+,9
MOV 8,*9
```

```
 B @C$6
C$7
*  return(sign*n);
 MOV 14,8
 INCT 8
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 INCT 8
 MOV *8,8
 MOV 8,7
 MPY *14+,7
 AI 14,4
 B *13
*}
*/*
*** itod(nbr,str,sz) -
***        convert nbr to signed decimal string of width sz
***        right justified, blank filled, result in str[].
***        sz includes 0-byte string terminator
**/
*itod(nbr,str,sz) int nbr,sz; char str[];
ITOD
*{ char sgn;
*  sgn=' ';
 DECT 14
 MOV 14,8
 DECT 14
 MOV 8,*14
 LI 8,32
 MOV *14+,9
 MOVB @>8311,*9
*  if(nbr<0) { nbr=-nbr; sgn='-'; }
 MOV 14,8
 AI 8,8
 MOV *8,8
 DECT 14
 MOV 8,*14
 CLR 8
 BL @C$LT
 ABS 8
 JNE $+6
 B @C$9
 MOV 14,8
 AI 8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,10
```

```
 MOV *8,8
 NEG 8
 MOV *14+,9
 MOV 8,*9
 MOV 14,8
 DECT 14
 MOV 8,*14
 LI 8,45
 MOV *14+,9
 MOVB @>8311,*9
*  str[--sz]=0;
C$9
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 DECT 14
 MOV 8,*14
 CLR 8
 MOV *14+,9
 MOVB @>8311,*9
*  while(sz)
C$10
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$11
*  { str[--sz]=nbr%10+'0';
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,10
 MOV *8,8
```

```
 DECT 14
 MOV 8,*14
 LI 8,10
 BL @C$REM
 DECT 14
 MOV 8,*14
 LI 8,48
 A *14+,8
 MOV *14+,9
 MOVB @>8311,*9
*    if(!(nbr=nbr/10))break;
 MOV 14,8
 AI 8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,10
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,10
 BL @C$DIV
 MOV *14+,9
 MOV 8,*9
 BL @C$LNEG
 ABS 8
 JNE $+6
 B @C$12
 B @C$11
*  }
C$12
 B @C$10
C$11
*  if(sz) str[--sz]=sgn;
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$13
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
```

```
 DECT 14
 MOV 8,*14
 MOV 14,8
 INCT 8
 MOVB *8,8
 SRL 8,8
 MOV *14+,9
 MOVB @>8311,*9
*  while(sz) str[--sz]=' ';
C$13
C$14
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$15
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 DECT 14
 MOV 8,*14
 LI 8,32
 MOV *14+,9
 MOVB @>8311,*9
 B @C$14
C$15
*}
 INCT 14
 B *13
*main()
 DEF MAIN
MAIN
* { int c,p,h,q,d,n,s;
*   char buff[4];
*   char hs[2],qs[2],ds[2],ns[2],ss[2];
*   puts("enter price in cents:\n\t");
 AI 14,-28
 LI 8,C$16+0
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTS
 INCT 14
```

```
*   p=atoi(gets(buff));
 MOV 14,8
 AI 8,24
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 DECT 14
 MOV 8,*14
 BL *12
 DATA GETS
 INCT 14
 DECT 14
 MOV 8,*14
 BL *12
 DATA ATOI
 INCT 14
 MOV *14+,9
 MOV 8,*9
*   c=100-p;
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 LI 8,100
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 S *14+,8
 NEG 8
 MOV *14+,9
 MOV 8,*9
*   putchar('\n');
 LI 8,10
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTCHA
 INCT 14
*  c= change(c,50,h,hs," half-dollar");
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
```

```
 MOV 8,*14
 LI 8,50
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,16
 DECT 14
 MOV 8,*14
 LI 8,C$16+24
 DECT 14
 MOV 8,*14
 BL *12
 DATA CHANGE
 AI 14,10
 MOV *14+,9
 MOV 8,*9
*  c= change(c,25,q,qs," quarters");
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,25
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,26
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,14
 DECT 14
 MOV 8,*14
 LI 8,C$16+37
 DECT 14
 MOV 8,*14
 BL *12
 DATA CHANGE
 AI 14,10
 MOV *14+,9
 MOV 8,*9
```

```
*  c= change(c,10,d,ds," dimes");
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,10
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,24
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 DECT 14
 MOV 8,*14
 LI 8,C$16+47
 DECT 14
 MOV 8,*14
 BL *12
 DATA CHANGE
 AI 14,10
 MOV *14+,9
 MOV 8,*9
*  c= change(c,5,n,ns," nickles");
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,5
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,22
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,10
```

```
 DECT 14
 MOV 8,*14
 LI 8,C$16+54
 DECT 14
 MOV 8,*14
 BL *12
 DATA CHANGE
 AI 14,10
 MOV *14+,9
 MOV 8,*9
*  c= change(c,1,s,ss," pennies");
 MOV 14,8
 AI 8,26
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,28
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,1
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,20
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,8
 DECT 14
 MOV 8,*14
 LI 8,C$16+63
 DECT 14
 MOV 8,*14
 BL *12
 DATA CHANGE
 AI 14,10
 MOV *14+,9
 MOV 8,*9
*/* main keeps calling 'change' with
*   different ACTUAL arguments */
* putchar('\n');
 LI 8,10
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTCHA
 INCT 14
* puts("that's your change! \n"); }
 LI 8,C$16+72
```

```
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTS
 INCT 14
 AI 14,28
 B *13
C$16 BYTE 101,110,116,101,114,32,112,114,105,99,101,32
 BYTE 105,110,32,99,101,110,116,115,58,10,9,0
 BYTE 32,104,97,108,102,45,100,111,108,108,97,114
 BYTE 0,32,113,117,97,114,116,101,114,115,0,32
 BYTE 100,105,109,101,115,0,32,110,105,99,107,108
 BYTE 101,115,0,32,112,101,110,110,105,101,115,0
 BYTE 116,104,97,116,39,115,32,121,111,117,114,32
 BYTE 99,104,97,110,103,101,33,32,10,0
 EVEN
*
*change(i,j,k,ks,str)
CHANGE
*/* i,j,k,ks,str are FORMAL arguments */
*  int i,j,k;
*  char ks,str;
* { if    (i>=j)
 MOV 14,8
 AI 8,10
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,10
 MOV *8,8
 BL @C$GE
 ABS 8
 JNE $+6
 B @C$18
*   {k = i/j;
 MOV 14,8
 AI 8,6
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 MOV *8,8
 BL @C$DIV
 MOV *14+,9
```

```
 MOV 8,*9
*    i = i % j;
 MOV 14,8
 AI 8,10
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,12
 MOV *8,8
 BL @C$REM
 MOV *14+,9
 MOV 8,*9
*   itod(k,ks,2);
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 MOV 14,8
 AI 8,6
 MOV *8,8
 DECT 14
 MOV 8,*14
 LI 8,2
 DECT 14
 MOV 8,*14
 BL *12
 DATA ITOD
 AI 14,6
*   puts(ks);
 MOV 14,8
 AI 8,4
 MOV *8,8
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTS
 INCT 14
*   puts(str );
 MOV 14,8
 INCT 8
 MOV *8,8
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTS
```

```
 INCT 14
*   putchar('\n');}
 LI 8,10
 DECT 14
 MOV 8,*14
 BL *12
 DATA PUTCHA
 INCT 14
*   return(i);   }
C$18
 MOV 14,8
 AI 8,10
 MOV *8,8
 B *13
*/* so it can be used for next run
*   of 'change()' */
*
 END
```

Disk 40. Contents of file CFDC

```
/* change for a dollar       cfdc  */

#include dsk2.conv;c
  int c;
main()
  {int p,h,q,d,n;
   char buff[5];
   char hs[5];
   char qs[5];
   char ds[5];
   char ns[5];
   char cs[5];

   puts("enter price in cents \n");
   p=atoi(gets(buff)); /* char to int*/
   c=100-p;

 { if  (c>=50)
  {h = c/50;
   c=c % 50; /* % gives remainder */
  itod(h,hs,2);
/* int 'h' to char 'hs'-len 2 */
  puts(hs);
  puts(" half-dollar  \n");}
       }
 { if    (c>=25)
   {q = c/25;
    c = c % 25;
   itod(q,qs,2);
   puts(qs);
   puts(" quarters \n");}
                }
 { if (c>=10)
   {d=c/10;
    c=c % 10;
   itod(d,ds,2);
   puts(ds);
   puts(" dimes \n");}
             }
 { if (c>=5)
   {n=c/5;
    c=c%5;
   itod(n,ns,2);
   puts(ns);
   puts(" nickels \n");}
          }
 { if (c>=1)
   {itod(c,cs,2);
```

```
    puts(cs);
    puts(" pennies \n"); }}
 putchar('\n');
 puts("that's your change! \n");
     }
```

Disk 40. Contents of file CHANGE

```
/* change as a library function    */


change(i,j,k,ks,str)
  int i,j,k;
  char ks,str;
 { if    (i>=j)
   {k = i/j;
    i = i % j;
   itod(k,ks,2);
   puts(ks);
   puts(str );
   putchar('\n');}
   return(i);   }
```

Disk 40. Contents of file CPOS

```
/* CPOS  dlm */
/* library function */

cpos(s,c,n)
 char *s; char *c; int n;
{ n=1;
  while (*s)
    { if  ( *s == *c)
       { return (n);
         break ;     }
      else {
            ++s;
            ++n ;   }
               }
  n=0;     }
```

Disk 40. Contents of file CPOS;C

```
/* CPOS  dlm */
/* library function */

cpos(s,c,n)
 char *s; char *c; int n;
{ n=1;
  while (*s)
    { if  ( *s == *c)
       { return (n);
         break ;     }
      else {
            ++s;
            ++n ;   }
               }
  n=0;     }
```

Disk 40. Contents of file CSEG

```
/* CSEG    */
/* makes new str s from ss starting
at m, n chars long */

cseg(s,ss,m,n)
 int n; int m; char *s; char *ss;
{ int i; i=0;
  while (++i<30)
{ s[i] = '\0' ; }
  ss = ss + (m-1);
  while (n--)
   (*s++) = (*ss++);
            ;
  if  (n=0)
{ *s == '\0';
  return ;   }  }
```

Disk 40. Contents of file CSEG;C

```
/* CSEG    */
/* makes new str s from ss starting
at m, n chars long */

cseg(s,ss,m,n)
 int n; int m; char *s; char *ss;
{ int i; i=0;
  while (++i<30)
{ s[i] = '\0' ; }
  ss = ss + (m-1);
  while (n--)
   (*s++) = (*ss++);
            ;
  if  (n=0)
{ *s == '\0';
  return ;   }  }
```

Disk 40. Contents of file CSORT

```
/* sort as a library funct */

 sort(i,f,s,k,item)
 int i,f,s,k,item[];
 { puts("\n \n here is your sorted list!");
  while (f==1)
  { i=f=s=k=0;
   while(i<=(S-1))
   { if (item[i]<= item[i+1])
    { s= item[i]; f=1;
     item[i]=item[i+1];
     item[i+1]=s; }
   i=i+1;  }}
  while(++k<=S)
     { printf("\n item# %d= %d",k,item[k]);}
            }
```

Disk 40. Contents of file DISPLAYC

```
/* test of graphic functions  */
  #include dsk2.grf1rf
main()
{ int r;  char *s;
/*  *s is 'address of s'   */
  s="FFFFFFFFFFFFFFFF"; r=7;
 grf1();
 screen(5);
 chrdef(136,s);
 chrdef(152,s);
/* these set 136,152 to square */
 color(17,9,5); color(18,11,11);
 color(19,6,11);
/* char set 3 higher than XB */
 hchar(7,4,136,27); /*top of box*/
 hchar(17,4,136,27);/* bottom */
 vchar(8,4,136,9);
 vchar(8,30,136,9); /* sides */
 while (++r<=16)
 {hchar(r,5,144,25); }
 hchar(9,6,152,5); hchar(15,6,152,5);
 vchar(10,6,152,5); vchar(9,19,152,7);
 hchar(9,15,152,4); hchar(12,15,152,4);
 vchar(10,15,152,2); vchar(9,28,152,7);
 hchar(9,24,152,4);  hchar(12,24,152,4);
 vchar(10,24,152,2); /* letters */
 putchar('\n'); /* next line */
 puts(" this is c graphics!");
```

Disk 40. Contents of file GETINT

```
/* getint()                      */
/* format:  status=getint(&num) */
/* status contains:              */
/*    -1 : EOF was found         */
/*     1 : error (no #s)         */
/*     0 : successful input      */

getint(ptrint)
int *ptrint;
{
  char buffer[81];
  int index,ch;
  index=0;

  while((ch=getchar())=='\n'|ch==' ')
    ; /* do-nothing */
  while(ch!=EOF & ch!='\n' & ch!=' ' & index<81)
    {
    buffer[index++]=ch;
    ch=getchar();
    }
  buffer[index]=0;
  if(ch==EOF)
    return(STOP);
  else
    return(stoi(buffer,ptrint));
}
```

Disk 40. Contents of file INDEX

```
/* (Mod from WIBLE by dlm) */

index(s,c)
char *s,c;
{ int n; n=1;
  puts(s);putchar('\n');puts(c);
  while(*s++)
   {printf("%d",*s);
     if (*s == c)
     return(n);
     else
      { ++s;
        ++n;}  }
    return (0); }
```

Disk 40. Contents of file INTDEMC

```
/* INTDEMC    */
/* getint() demo  */

#include dsk2.conv;c
#define STOP -1
#define NO 1
#define YES 0
#define EOF -1
#include dsk2.getint
#include dsk2.stoi
main()
{
  int num,stat;
  char string[81];
  puts("This reads in integers until it detects\n");
  puts("a CTRL-Z.\n");
  while((stat=getint(&num))!=STOP)
    if(stat==YES) {
        itod(num,string,5);
        puts(string);
        puts(" is the number accepted.\n");
      }
    else
      puts("That was no integer...try again!!\n");
  puts("We're finished!\n");
}
```

Disk 40. Contents of file MAX

```
max(a,b)
int a,b;
{
  if(b<a) return(a); else return(b);
}
```

Disk 40. Contents of file MIN

```
min(a,b) int a,b;
{ if(b>a)return(a); else return(b);
}
```

Disk 40. Contents of file MODROMC

```
/* modromc */
/* Prints #1-100 in Roman numbers */

main()
{ int x;
  x=1;
  while(++x<=100)
{ rom(x); }  }

rom(a)
int a;
{
   a=rm(a,1000,"M");
   a=rm(a,900,"CM");
   a=rm(a,500,"D");
   a=rm(a,400,"CD");
   a=rm(a,100,"C");
   a=rm(a,90,"XC");
   a=rm(a,50,"L");
   a=rm(a,40,"XL");
   a=rm(a,10,"X");
   a=rm(a,9,"IX");
   a=rm(a,5,"V");
   a=rm(a,4,"IV");
   rm(a,1,"I");
   putchar('\n'); }


rm(i,j,c)
char c[3];
int i,j;
{
    while(i>=j)
     {
     puts(c);
     i=i-j;
       }
    return(i);
     }
```

Disk 40. Contents of file PMK2C

```
 /*  PMK2C  */
/* TEST AN INTEGER from keyboard
   FOR PRIMENESS */
/*  '1000' STOPS PROGRAM */

  #include dsk2.conv;c


main()
   { char buff[10];
   int i,n;

   while(n != 1000)
      { puts("Enter number");
        putchar('\n');
        i=1;
        n=atoi(gets(buff));
        while (++i<n)
           if (n%i==0)
         /* no remainder */
              {puts("not prime\n");
               break; }
       /*jump out of loop */
           if (i==n)
       /* only factor is itself */
                puts("prime\n");
                   }
       putchar('\n');
       puts("come back soon!"); }
```

Disk 40. Contents of file PRF

```
/* printf references */
/* loaded into lib as 'prf' */
  #asm REF PRINTF
  #endasm
```

Disk 40. Contents of file PRIM2C

```
/* PRIM2C   */
/* c-code;must be compiled and assembled */
/* modified for c99 by d.l.mahler       */

/* FIND ALL PRIMES BELOW 1000    */
    #include DSK2.conv;c
main()
  {int i,n;
   char pr[4];
   i=1;
   n=2;

   while (++n<=1000) /* n prime candidate */
     { i=1;
       while (++i<n)
        { if (n%i==0 )  /* no remainder  */
          break; } /*jump out of loop*/
       if (i==n) /*only factor is n*/
          { putchar('\n');
            itod(n,pr,4);
             /*int to char*/
             putchar('\t');
             puts ( pr );} }
   putchar('\n');
   puts(" end of list"); }
```

Disk 40. Contents of file PRIM2S

```
*c99 v2.0 (c) 1986 Clint Pulley
 REF C$CIND,C$DIV,C$REM,C$ASR,C$ASL,C$EQ,C$NE,C$LT,C$LE
 REF C$GT,C$GE,C$ULT,C$ULE,C$UGT,C$UGE,C$LNEG,C$SWCH
 REF GETCHA,GETS,PUTCHA,PUTS,LOCATE,POLL,TSCRN,EXIT
*/* PRIM2C   */
*/* c-code;must be compiled and assembled */
*/* modified for c99 by d.l.mahler       */
*
*/* FIND ALL PRIMES BELOW 1000    */
*    #include DSK2.conv;c
*/* simple conversion functions */
*/*
*** n=atoi(s) - convert string to integer
**/
*atoi(s)  char *s;
ATOI
*{ int sign,n;
*  while(*s==' ')++s;
 AI 14,-4
C$2
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
 SRL 8,8
 BL 15
 LI 8,32
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$3
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
 B @C$2
C$3
*  sign=1;
 MOV 14,8
 INCT 8
 BL 15
 LI 8,1
 MOV *14+,9
 MOV 8,*9
*  if(*s=='-') { sign=-1; ++s; }
 MOV 14,8
 AI 8,6
 MOV *8,8
```

```
 MOVB *8,8
 SRL 8,8
 BL 15
 LI 8,45
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$4
 MOV 14,8
 INCT 8
 BL 15
 LI 8,1
 NEG 8
 MOV *14+,9
 MOV 8,*9
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
*  if(*s=='+') ++s;
C$4
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
 SRL 8,8
 BL 15
 LI 8,43
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$5
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
*  n=0;
C$5
 MOV 14,8
 BL 15
 CLR 8
 MOV *14+,9
 MOV 8,*9
*  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
C$6
 MOV 14,8
 AI 8,6
 MOV *8,8
 MOVB *8,8
 SRL 8,8
```

```
 BL 15
 LI 8,48
 BL @C$GE
 BL 15
 MOV 14,8
 AI 8,8
 MOV *8,8
 MOVB *8,8
 SRL 8,8
 BL 15
 LI 8,57
 BL @C$LE
 INV *14
 SZC *14+,8
 ABS 8
 JNE $+6
 B @C$7
 MOV 14,8
 BL 15
 LI 8,10
 BL 15
 MOV 14,8
 AI 8,4
 MOV *8,8
 MOV 8,7
 MPY *14+,7
 BL 15
 MOV 14,8
 AI 8,10
 MOV *8,9
 INC *8
 MOV 9,8
 MOVB *8,8
 SRL 8,8
 A *14+,8
 BL 15
 LI 8,48
 S *14+,8
 NEG 8
 MOV *14+,9
 MOV 8,*9
 B @C$6
C$7
*  return(sign*n);
 MOV 14,8
 INCT 8
 MOV *8,8
 BL 15
 MOV 14,8
 INCT 8
 MOV *8,8
```

```
 MOV 8,7
 MPY *14+,7
 AI 14,4
 B *13
*}
*/*
*** itod(nbr,str,sz) -
***         convert nbr to signed decimal string of width sz
***         right justified, blank filled, result in str[].
***         sz includes 0-byte string terminator
**/
*itod(nbr,str,sz) int nbr,sz; char str[];
ITOD
*{ char sgn;
*  sgn=' ';
 DECT 14
 MOV 14,8
 BL 15
 LI 8,32
 MOV *14+,9
 MOVB @>8311,*9
*  if(nbr<0) { nbr=-nbr; sgn='-'; }
 MOV 14,8
 AI 8,8
 MOV *8,8
 BL 15
 CLR 8
 BL @C$LT
 ABS 8
 JNE $+6
 B @C$9
 MOV 14,8
 AI 8,8
 BL 15
 MOV 14,8
 AI 8,10
 MOV *8,8
 NEG 8
 MOV *14+,9
 MOV 8,*9
 MOV 14,8
 BL 15
 LI 8,45
 MOV *14+,9
 MOVB @>8311,*9
*  str[--sz]=0;
C$9
 MOV 14,8
 AI 8,6
 MOV *8,8
```

```
 BL 15
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 BL 15
 CLR 8
 MOV *14+,9
 MOVB @>8311,*9
*  while(sz)
C$10
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$11
*  { str[--sz]=nbr%10+'0';
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 BL 15
 MOV 14,8
 AI 8,10
 MOV *8,8
 BL 15
 LI 8,10
 BL @C$REM
 BL 15
 LI 8,48
 A *14+,8
 MOV *14+,9
 MOVB @>8311,*9
*    if(!(nbr=nbr/10))break;
 MOV 14,8
 AI 8,8
 BL 15
 MOV 14,8
 AI 8,10
 MOV *8,8
 BL 15
 LI 8,10
 BL @C$DIV
 MOV *14+,9
```

```
 MOV 8,*9
 BL @C$LNEG
 ABS 8
 JNE $+6
 B @C$12
 B @C$11
*  }
C$12
 B @C$10
C$11
*  if(sz) str[--sz]=sgn;
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$13
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
 A *14+,8
 BL 15
 MOV 14,8
 INCT 8
 MOVB *8,8
 SRL 8,8
 MOV *14+,9
 MOVB @>8311,*9
*  while(sz) str[--sz]=' ';
C$13
C$14
 MOV 14,8
 AI 8,4
 MOV *8,8
 ABS 8
 JNE $+6
 B @C$15
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,6
 DEC *8
 MOV *8,8
```

```
 A *14+,8
 BL 15
 LI 8,32
 MOV *14+,9
 MOVB @>8311,*9
 B @C$14
C$15
*}
 INCT 14
 B *13
*main()
 DEF MAIN
MAIN
*  {int i,n;
*   char pr[4];
*   i=1;
 AI 14,-8
 MOV 14,8
 AI 8,6
 BL 15
 LI 8,1
 MOV *14+,9
 MOV 8,*9
*   n=2;
 MOV 14,8
 AI 8,4
 BL 15
 LI 8,2
 MOV *14+,9
 MOV 8,*9
*
*   while (++n<=1000) /* n prime candidate */
C$17
 MOV 14,8
 AI 8,4
 INC *8
 MOV *8,8
 BL 15
 LI 8,1000
 BL @C$LE
 ABS 8
 JNE $+6
 B @C$18
*     { i=1;
 MOV 14,8
 AI 8,6
 BL 15
 LI 8,1
 MOV *14+,9
 MOV 8,*9
*       while (++i<n)
```

```
C$19
 MOV 14,8
 AI 8,6
 INC *8
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL @C$LT
 ABS 8
 JNE $+6
 B @C$20
*        { if (n%i==0 )  /* no remainder  */
 MOV 14,8
 AI 8,4
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,8
 MOV *8,8
 BL @C$REM
 BL 15
 CLR 8
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$21
*          break; } /*jump out of loop*/
 B @C$20
C$21
 B @C$19
C$20
*       if (i==n) /*only factor is n*/
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL 15
 MOV 14,8
 AI 8,6
 MOV *8,8
 BL @C$EQ
 ABS 8
 JNE $+6
 B @C$22
*          { putchar('\n');
 LI 8,10
 BL 15
 BL *12
 DATA PUTCHA
```

```
 INCT 14
*            itod(n,pr,4);
 MOV 14,8
 AI 8,4
 MOV *8,8
 BL 15
 MOV 14,8
 INCT 8
 BL 15
 LI 8,4
 BL 15
 BL *12
 DATA ITOD
 AI 14,6
*             /*int to char*/
*             putchar('\t');
 LI 8,9
 BL 15
 BL *12
 DATA PUTCHA
 INCT 14
*             puts ( pr );} }
 MOV 14,8
 BL 15
 BL *12
 DATA PUTS
 INCT 14
C$22
 B @C$17
C$18
*   putchar('\n');
 LI 8,10
 BL 15
 BL *12
 DATA PUTCHA
 INCT 14
*   puts(" end of list"); }
 LI 8,C$16+0
 BL 15
 BL *12
 DATA PUTS
 INCT 14
 AI 14,8
 B *13
C$16 BYTE 32,101,110,100,32,111,102,32,108,105,115,116
 BYTE 0
 EVEN
 END
```

Disk 40. Contents of file RK3C

```
 /* RK3C- main only  */
/* Change Arabic number from
keyboard to Roman */

  #include dsk2.conv;c
  #include dsk3.rom
  #include dsk2.prf
main()
{ int x;
  char buff[10];
  printf("input an arabic number: ");
  x=atoi(gets(buff));
  {rom(x);  }
  printf("that is the roman equivalent \n ");
  printf("of %d \n",x);  }
```

Disk 40. Contents of file SCINC

```
/*This is c.code-must be compiled
and assembled (d.l.mahler) */
/* SCALED INTEGERS   scinc     */

#include dsk2.conv;c

main()

{ char dollsgn,centsgn;
  char str[5],sr[3],buff[10];
 /* [] indicates length */
  int  m,d,c,p,n,dm,q,h,db;
  puts("how many pennies? \n");
  p=atoi(gets(buff));
  puts("how many nickels? \n");
  n=atoi(gets(buff));
  n=n*5;
  puts("how many dimes? \n");
  dm=atoi(gets(buff));
  dm=dm*10 ;
  puts("how many quarters? \n");
  q=atoi(gets(buff));
  q=q*25;
  puts("how many half-dollars? \n");
  h=atoi(gets(buff));
  h=h*50;
  puts("and, finally, how many dollar bills? \n");
  db=atoi(gets(buff));
  db=db*100;
  m=p+n+dm+q+h+db;
  dollsgn='$';
  centsgn='.';
  d=m/100;
  c=m-(d*100);
  putchar('\n');
  puts("you have exactly ");
  putchar(dollsgn);
  itod(d,str,3);
 /*changes int 'd' to string 'str'*/
  puts(str);
  putchar(centsgn);
  itod(c,sr,3);
  puts(sr);
  putchar('\n');
      }
```

Disk 40. Contents of file SCPDEM;C

```
/* scpdem  */
char s1[80];
char s2[80];
main()
{ int c;
puts("Enter the old string:");
c=gets(s2);
puts(s2); puts(" is old string");
putchar('\n');
strcpy(s1,s2);

puts(s1); puts(" is the new string");
putchar('\n');
           }
strcpy(s,ss)
 char *s; char *ss;
{

  while  ((*s++) = (*ss++))
           ;
return ;       }
```

Disk 40. Contents of file SIMP3C

```
/* simple string manipulation  */
/* SIMP3C  */
#define MAXLENGTH 10
  /* assigns constant 10 to
               MAXLENGTH */
main()
  { char data[MAXLENGTH]; int n;
    char *s;  s="brenda";
   /*  s is an address and *s is
        "the variable stored at s"
    OR "s is a pointer to *s" */
    n=0;
    puts("enter your name \n");
    gets(data);
    /*gets value from keyboard */
    puts("hello,");
    puts(data);
    putchar('\n');
    while (++n<10)
     {putchar('\n');
      puts("Where are the cookies,");
      puts(s); putchar('?');}
   /* uses the pointer s  */
    putchar('\n');
    puts("goodbye,");
    puts(data);
    putchar('\n');    }
```

Disk 40. Contents of file SLDM;C

```
/* SLENDEM     */
#include dsk2.conv;c
char s1[80],s2[80];
char answer[3];
main()
{
  int c,a;
  puts("Simple test of strlen\n\n");
  puts("Enter large  string:");
  c=gets(s1);
  puts("\n\nEnter small string:");
  c=gets(s2);

  a=strlen(s1);
  itod(a,answer,3);
  puts("\nLength of first string is:");
  puts(answer);
  a=strlen(s2);
  itod(a,answer,3);
  puts("\nLength of second string is:");
  puts(answer);       }


/* Tom Wible */
strlen(s)
char *s;
{
  int n; n=0;
  while(*s++)
     ++n;
  return(n);
}
```

Disk 40. Contents of file SPTST;C

```
/* test of graphics 1 library
*/
#include "dsk2.grf1rf"
#include dsk2.conv;c
main()
{ int n,flg,m1,q,m2;
  char cb[7]; char *t; char *s;
/* *t = "value at address t" */
  s="3C00BDA5A5BD003C"; m1=9;
  t="2466C31818C36624"; m2=-8;
  grf1();
  clear();
  chrdef(128,s); chrdef(129,t);
  sprite(1,128,2,80,120);
  sprite(0,129,16,80,0);
  spmag(2);
  screen(6);
  spmotn(0,0,m1); spmotn(1,0,m2);
  spmct(2);  /* enable motion */
  flg=n=q=0;
while(q<10)
  {locate(24,2); puts("sprdist=");
   puts(itod(spdist(0,1),cb,4));
   flg = spcnc(0,1,1);  /* coinc */
   if(flg)
     {locate(5,15);
     puts("BANG!");
     n=n+1;}
   if(n>2) /* each coinc -> >1 flg */
    {m1=-m1;  m2=-m2; /* reverse */
     spmotn(0,0,m1); spmotn(1,0,m2);
    n=0; q=q+1; /* reset flg count */
    locate(5,15); puts("BOOM!");}
   poll(1); }  /* key stops sprite */
spdall();
putchar('\n\n');
puts("That's the end!");       }
```

Disk 40. Contents of file STNCPY

```
/* STNCPY */
/* library function */

stncpy(s,ss,n)   /* dlm */
/* modified from T.Wible */
 char *s; char *ss;   int n;
{ int i; i=0;
  while (++i<30)
   { s[i]= '0' ;}
  while (n--)
    (*s++) = (*ss++) ;
           ;
  return ;       }
```

Disk 40. Contents of file STNCPYDMC

```
/* STNCPYDMC*/
/* demo of stncpy  */
#include dsk2.conv;c
char s1[80];
char s2[80];
main()
{ int c; int a; char buf[10];
puts("Enter the old string:");
c=gets(s2);
puts(s2); puts(" is old string");
putchar('\n');
puts("How many letters?  ");
a=atoi(gets(buf));

stncpy(s1,s2,a);

puts(s1); puts(" is the new string");
putchar('\n');
            }
stncpy(s,ss,n)
/*  dlm  modified from T.Wible */
 char *s; char *ss;   int n;
{ int i; i=0;
  while (++i<30)
   { s[i]= '0' ;}
  while (n--)
          (*s++) = (*ss++) ;
           ;
  return ;       }
```

Disk 40. Contents of file STOI

```
/* stoi(string,intptr) -   */
/* converts string to integer (intptr) */
/* and returns status report. */
stoi(string,intptr)
char string[];
int *intptr;
{
  int sign;
  int index;
  sign=1;  index=0;
  if(string[index]=='-'|string[index]=='+') {

    if(string[index++]=='-')
      sign= -1;
    else
      sign= 1;
    }
  *intptr=0;
  while(string[index]>='0' & string[index]<='9')
    *intptr=10*(*intptr)+string[index++]-'0';
  if(string[index]==0)
    {
    *intptr=sign*(*intptr);
    return(YES);
      }
  else
    return(NO);
}
```

Disk 40. Contents of file STRCAT

```
/* concatenate t to end of s
*/

strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
}
```

Disk 40. Contents of file STRCMP

```
/*
** Compare 2 strings; returns a # <0 if s<t, 0 if s==t,
** >0 if s>t
** Taken (and slightly adapted) from "The C Programming Language" by K&R
*/

strcmp(s,t)
char s[],t[];
{
  int i;
  i=0;
  while(s[i]==t[i])
       if(s[i++]==0
           return(0);
  return(s[i]-t[i]);
}
```

Disk 40. Contents of file STRCPY

```
/* copy t to s
*/

strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
```

Disk 40. Contents of file STRLEN

```
/* (? Warren Agee)
  "My own version of strlen(), which MUST be used with
   match() below. "
*/

strlen(string)          /* NOTE: This is kind of non-standard, cuz it returns*/
char string[];          /* the real length-1, to be consistent with the      */
{                       /* number conventions of char arrays                 */
  int len; len=0;       /* (first element==0).                               */
  while(string[len])
     len++;
  return(len-1);
}
```

Disk 40. Contents of file STRPOS;C

```
/* STRPOS;C  */
/* demo of index using global int */
#include dsk2.prf
char s1[80];
char s2[4];  int a;
main()
{ int c;
 char *ptr ; char *ptr2; a=1;
puts("Enter the big string:");
c=gets(s1); ptr=s1;
puts("Enter desired character:");
c=gets(s2); ptr2=s2;
 index(ptr,ptr2);
if(a)
printf (" '%c' is the # %d letter in %s", *ptr2,a,s1);
else
printf ("character not found!");
            }
index(s,c) /* mod from T.Wible */
 char *s; char *c;
{
  while (*s)
    { if  ( *s == *c)
        { return (a);
          break ;     }
      else {
            ++s;
            ++a ;   }
               }
  a=0;     }
```

Disk 40. Contents of file STRPS;C

```
/* STRPS;C  */
/*demo of cpos */
#include dsk2.prf
char s1[80];
char s2[4];
main()
{ int c; int a;
 char *ptr ; char *ptr2; a=1;
puts("Enter the big string:");
c=gets(s1); ptr=s1;
puts("Enter desired character:");
c=gets(s2); ptr2=s2;
a=cpos(ptr,ptr2,a);
if(a)
printf (" '%c' is the # %d letter in %s", *ptr2,a,s1);
else
printf ("character not found!");
            }

cpos(s,c,n)    /* dlm */
 char *s; char *c; int n;
{
  while (*s)
    { if  ( *s == *c)
       { printf("%d\n",n);
          return (n);
         break ;     }
      else {
            ++s;
            ++n ;   }
               }
  n=0;     }
```

Disk 40. Contents of file TESTMXC

```
/* testmxc    */
/* test max function */
/* quits if '999' entered */

#include dsk2.conv;c
#define STOP -1
#define NO 1
#define YES 0
#define EOF -1
#include dsk2.stoi
#include dsk2.getint
#include dsk2.max
main()
{
  int a,b,c;
  char string[81];
  char buff[81];
while (a !=999)
 { puts(" enter a number,then another:\n");
  a=atoi(gets(buff));
  b=atoi(gets(buff));
  c=max(a,b);
  itod(c,string,5);
  puts(string);
  puts(" is the bigger number.\n"); }
  puts("We're finished!\n");
            }
```

Disk 40. Contents of file TSEGC

```
/* testing cseg   */
#include dsk2.cseg
char q1[20];  char q2[];
main()
{ q2 ="donaldmahler";
puts("   orig string is  ");
puts(q2);
putchar('\n');
 cseg(q1,q2,7,6);
puts("   new string is  ");
puts(q1);  }
```

Disk 40. Contents of file TSTGIC

```
/* tstgic   CONT-Z ends  */
/* getint() test  */
/* accepts number from keyboard;
  rejects if not integer  */
#include dsk2.conv;c
#define STOP -1
#define NO 1
#define YES 0
#define EOF -1
#include dsk2.getint
#include dsk2.stoi
main()

{ int num,stat;
  char string[81];
  char str[81];
  puts("enter a number: \n");
  while((stat=getint(&num))!=STOP)
/* &num is 'address of number */
    if(stat==YES)
      { itod(num,string,6);
        puts(string);
        puts(" is the number accepted.\n");
        itod(stat,str,5);
        puts(str);
        puts(" is the status.\n"); }
   else
   puts("That was no integer...try again!!\n");
  puts("We're finished!\n");   }
```

Disk 41. PULSAR Utilities

Version: *Author*: Michael Amundsen
Requires: EA *Language*: AL *Updated*: 08/17/86

Set of assembly language utilities (with source code) to allow you to easily translate BASIC programs to assembly. Contains several demos. Great for quickly writing assembly code or for "borrowing" code for your own programs.

```
dskdir. v2.0. 12-dec-96
Disk name               = PULSAR1
Sectors total           = 360
Sectors used            = 306
Sectors available       = 52
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >017 *INPUT/O     24 DIS/FIX 80   >14f 023
002 >018 *INPUT/S      5 DIS/VAR 80   >122 004
003 >010 *RANDOM/O    17 DIS/FIX 80   >0f6 016
004 >011 *RANDOM/S    10 DIS/VAR 80   >106 005 >11e 004
005 >012 *READ/O      20 DIS/FIX 80   >10b 019
006 >00f *READ/S      16 DIS/VAR 80   >0e7 015
007 >013 -README       4 DIS/VAR 80   >126 003
008 >002 DATASET/S    39 DIS/VAR 80   >022 038
009 >003 FLOAT/S      20 DIS/VAR 80   >048 019
010 >004 GRAPH1/S     15 DIS/VAR 80   >05b 014
011 >005 IFSET/S      15 DIS/VAR 80   >069 014
012 >006 INIT-EA/S    22 DIS/VAR 80   >077 021
013 >007 INTMATH/S     7 DIS/VAR 80   >08c 006
014 >008 KEYSET/S     11 DIS/VAR 80   >092 010
015 >009 LOOPSET/S     5 DIS/VAR 80   >09c 004
016 >00a PULSAR/O     38 DIS/FIX 80   >0a0 037
017 >00b PULSAR/S      4 DIS/VAR 80   >0c5 002 >0e6 001
018 >00c RANDSET/S     8 DIS/VAR 80   >0c7 007
019 >00d SCRN-IO/S    15 DIS/VAR 80   >0ce 014
020 >00e START-EA/S   11 DIS/VAR 80   >0dc 010
```

Disk 41. Contents of file *INPUT/S

```
*
* INPUT AT DEMO
*

       COPY "DSK2.START-EA/S"

       BL   @CHRSET
       DATA LRGSET,CHAR1
MAIN   BL   @SCRCLR
       BL   @PRNTAT
       DATA D1,D9,L1
       BL   @PRNTAT
       DATA D2,D9,L2
       BL   @PRNTAT
       DATA D9,D3,L3
       BL   @INPTAT
       DATA D12,D3,D20,LINE
       BL   @PRNTAT
       DATA D17,D3,L5
       BL   @PRNTAT
       DATA D20,D3,LINE
       BL   @PRNTAT
       DATA D23,D6,L4
REGET  BL   @KEYCON
       BL   @IFTHEN
       DATA KEYHIT,EQ,FCTN8,MAIN
       BL   @IFELSE
       DATA KEYHIT,EQ,QUITKY,EXIT,REGET
EXIT   LIMI 2
       LWPI GPLWS
       BLWP @>0000

* USER VARIABLES

L1     DATA 13
       TEXT 'INPUT-AT DEMO'
L2     DATA 13
       TEXT '=============='
L3     DATA 27
       TEXT 'ENTER A STRING(20 CHR MAX):'
L4     DATA 18
       TEXT 'PRESS REDO OR QUIT'
L5     DATA 10
       TEXT 'YOU TYPED:'
LINE   BSS  >FF
CHAR1  DATA >0900

       END
```

Disk 41. Contents of file *RANDOM/S

```
*******************************
*                             *
*     RANDOM NUMBER DEMO      *
*     ==================      *
*                             *
*     M Amundsen 3-30-85      *
*                             *
*    Uses PULSAR Routines     *
*                             *
*******************************

* LOAD DEF/REF TABLE

       COPY "DSK2.START-EA/S"

* MAIN PROGRAM

       BL   @SCRCLR           100 CALL CLEAR
       BL   @CHRSET           110 (* LOAD BIG LETTERS *)
       DATA LRGSET,CHAR1
       BL   @PRNTAT           120 DISPLAY AT(3,3):M1$
       DATA D3,D3,M1
       BL   @PRNTAT           130 DISPLAY AT(4,3):M2$
       DATA D4,D3,M2
       BL   @PRNTAT           140 DISPLAY AT(8,3):M3$
       DATA D8,D3,M3
       BL   @PRNTAT           150 DISPLAY AT(10,3):CLM$
       DATA D10,D3,CLM
       BL   @INPTAT           160 ACCEPT AT(10,3)SIZE(12):I1$
       DATA D10,D3,D12,I1
       BL   @STRNUM           170 FP1=STR$(I1)
       DATA I1,FP1
       BL   @FLTINT           180 INT1=FP1 (* SWITCH FLT PT TO INT *)
       DATA FP1,INT1
       BL   @SETRND           190 (* SET THE RANDOM SEED TO INT1 *)
       DATA INT1
AGAIN  BL   @RANDOM           200 (* GENERATE A RANDOM VALUE *)
       BL   @NUMSTR           210 M5$=VAL(RND)
       DATA RND,M5
       BL   @PRNTAT           220 DISPLAY AT(15,3):M4$
       DATA D15,D3,M4
       BL   @PRNTAT           230 DISPLAY AT(17,3):CLM$
       DATA D17,D3,CLM
       BL   @PRNTAT           240 DISPLAY AT(17,3):M5$
       DATA D17,D3,M5
       JMP  AGAIN             250 GOTO 200
```

```
* VARIABLES

M1      DATA 23
        TEXT 'RANDOM NUMBER GENERATOR'
M2      DATA 23
        TEXT '======================='
M3      DATA 17
        TEXT 'ENTER SEED VALUE:'
M4      DATA 10
        TEXT 'RESULT IS:'
M5      BSS  20
I1      BSS  20
CLM     DATA 20
        TEXT '                     '
FP1     BSS  8
INT1    DATA 0
CHAR1   DATA >0900

        END
```

Disk 41. Contents of file *READ/S

```
*******************************
*                             *
*    SIMPLE READ-DATA DEMO    *
*    ====================     *
*                             *
*     M. Amundsen 3-30-85     *
*                             *
*                             *
*   Example program using the *
* "PULSAR" assembly routines. *
*                             *
* NOTE:                       *
*   The  PULSAR  master  file *
* MUST be loaded into memory! *
*                             *
*******************************

* LOAD DEF/REF TABLE

       COPY "DSK2.START-EA/S"

* PUT UP TITLE                          (* "BASIC" VERSION *)

BEGIN  BL   @SCRCLR                    100 CALL CLEAR
       BL   @PRNTAT                    110 DISPLAY AT(1,8):L1$
       DATA D1,D8,L1
       BL   @PRNTAT                    120 DISPLAY AT(2,8):L2$
       DATA D2,D8,L2
       BL   @PRNTAT                    130 DISPLAY AT(3,8):L3$
       DATA D8,D2,L3

* READ & DISPLAY INTEGERS

NLOOP  BL   @SETDAT                    140 RESTORE "DLIST1" (INTEGER DATA)
       DATA INTDAT,DLIST1
       BL   @FOR                       150 FOR LOOP=10 TO 19 STEP 1
       DATA LOOP,D10,D19,D1
       BL   @GETDAT                    160 READ DTEMP
       DATA DTEMP
       BL   @INTFLT                    170 FLTEMP=DTEMP (TURN INT TO FLT)
       DATA DTEMP,FLTEMP
       BL   @NUMSTR                    180 STRING$=STR$(FLTEMP)
       DATA FLTEMP,STRING
       BL   @PRNTAT                    190 DISPLAY AT(LOOP,8):STRING$
       DATA LOOP,D8,STRING
       BL   @NEXT                      200 NEXT LOOP
       DATA LOOP
```

```
* READ & DISPLAY STRINGS

SLOOP  BL   @PRNTAT                     210 DISPLAY AT(8,15):L4$
       DATA D8,D15,L4
       BL   @SETDAT                     220 RESTORE "DLIST3" (STRING DATA)
       DATA STRDAT,DLIST3
       BL   @FOR                        230 FOR LOOP=10 TO 19 STEP 1
       DATA LOOP,D10,D19,D1
       BL   @GETDAT                     240 READ STRING$
       DATA STRING
       BL   @PRNTAT                     250 DISPLAY AT(LOOP,15):STRING$
       DATA LOOP,D15,STRING
       BL   @NEXT                       260 NEXT LOOP
       DATA LOOP

* WAIT FOR KEYHIT

       BL   @PRNTAT                     270 DISPLAY AT(23,5):L5$
       DATA D23,D5,L5
REGET  BL   @KEYCON                     280 CALL KEY(0,K) *NO "S"
       BL   @IFTHEN                     290 IF K=QUIT THEN 310
       DATA KEYHIT,EQ,QUITKY,EXIT
       BL   @IFELSE                     300 IF K=FCTNS THEN 100 ELSE 280
       DATA KEYHIT,EQ,FCTN8,BEGIN,REGET

* END PROGRAM

EXIT   LIMI 2                         310 *TURN ON INTERRUPTS*
       JMP  $                         320 GOTO 320

* VARIABLES

L1     DATA 14
       TEXT 'READ-DATA DEMO'
       EVEN
L2     DATA 14
       TEXT '=============='
       EVEN
L3     DATA 8
       TEXT 'NUMBERS:'
L4     DATA 8
       TEXT 'STRINGS:'
L5     DATA 18
       TEXT 'PRESS REDO OR QUIT'
LOOP   DATA 0
FLTEMP BSS  8
STRING BSS  20
DTEMP  BSS  20
DLIST1 DATA 1,2,3,4,5,6,7,8,9,0
DLIST3 DATA 1
       TEXT 'A'
```

```
DATA 2
TEXT 'BB'
DATA 3
TEXT 'CCC'
DATA 4
TEXT 'DDDD'
DATA 5
TEXT 'EEEEE'
DATA 6
TEXT 'FFFFFF'
DATA 7
TEXT 'GGGGGGG'
DATA 8
TEXT 'HHHHHHHH'
DATA 9
TEXT 'IIIIIIIII'
DATA 10
TEXT 'JJJJJJJJJJ'
EVEN

END
```

Disk 41. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain disk #41

The PULSAR assembly language utilities were written by Mike Amundsen. They allow you to write assembly language programs in a way very much like you would write a BASIC program. There are three example programs included. They are *RANDOM/S, *READ/S, and *INPUT/S. To run these programs load in PULSAR/O and the object file for the program (example: *READ/O) and then give the program name RUN to start. If you have questions or are interested in helping on future upgrades of PULSAR you can contact Mike Amundsen at:

PO Box 533
Bowling Green, OH 43402

Disk 41. Contents of file DATASET/S

```
*-------------------------*
* GET VDP DATA INTO CPU  *
*                         *
* BL   @GETVDP           *
* DATA SRC,DST,#BYTES    *
*                         *
* PEEKS VDP INTO CPU ADR *
*-------------------------*

GETVDP MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,R0
       MOV  *R2,R2
       BLWP @VMBR
       RT

*-------------------------*
* PUT VDP DATA FROM CPU  *
*                         *
* BL   @PUTVDP           *
* DATA SRC,DST,#BYTES    *
*                         *
* POKES CPU INTO VDP ADR *
*-------------------------*

PUTVDP MOV  *R11+,R1
       MOV  *R11+,R0
       MOV  *R11+,R2
       MOV  *R0,R0
       MOV  *R2,R2
       BLWP @VMBW
       RT

*-------------------------*
* MOV CPU DATA           *
*                         *
* BL   @MOVCPU           *
* DATA SRC,DST,#BYTES    *
*                         *
* MOVES CPU TO CPU ADR   *
*-------------------------*

MOVCPU MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       CLR  R3
MVCPU1 MOVB *R0+,*R1+
```

```
       INC  R3
       C    *R2,R3
       JNE  MVCPU1
       RT

*-------------------------*
* MOVE VDP DATA          *
*                        *
* BL   @MOVVDP           *
* DATA SRC,DST,#BYTES    *
*                        *
* MOVES VDP TO VDP ADR   *
*-------------------------*

MOVVDP MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,@MVVDP1
       MOV  *R1,@MVVDP2
       CLR  R3
MVVDP3 MOV  @MVVDP1,R0
       BLWP @VSBR
       MOV  @MVVDP2,R0
       BLWP @VSBW
       INC  @MVVDP1
       INC  @MVVDP2
       INC  R3
       C    *R2,R3
       JNE  MVVDP3
       RT

*-------------------------*
* MOVE DATA (POINTERS)   *
*                        *
* BL   @MOVDAT           *
* DATA SRC,DST,#BYTES    *
*-------------------------*

MOVDAT MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,R0
       MOV  *R1,R1
       MOV  *R2,R2
MVDAT1 CI   R2,0
       JEQ  MVDAT2
       MOVB *R0+,*R1+
       DEC  R2
       JMP  MVDAT1
MVDAT2 RT
```

```
*------------------------*
* POSITION DATA POINTER  *
*                        *
* BL   @SETDAT           *
* DATA DATYPE,DATADR     *
*------------------------*

SETDAT MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R0,@DATYPE
       MOV  R1,@DATAPT
       RT

*------------------------*
* GET DATA FROM LIST     *
*                        *
* BL   @GETDAT           *
* DATA VARIABLE          *
*------------------------*

GETDAT MOV  *R11+,R0
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  R0,@DATTP1
       BL   @IFTHEN
       DATA DATAPT,EQ,NOKEY,GTDAT4
       BL   @IFTHEN
       DATA DATYPE,EQ,STRDAT,GTDAT3
       BL   @IFTHEN
       DATA DATYPE,EQ,FLTDAT,GTDAT2
GTDAT1 BL   @MOVDAT
       DATA DATAPT,DATTP1,D2
       BL   @INTADD
       DATA DATAPT,D2,DATAPT
       JMP  GTDAT4
GTDAT2 BL   @MOVDAT
       DATA DATAPT,DATTP1,D8
       BL   @INTADD
       DATA DATAPT,D8,DATAPT
       JMP  GTDAT4
GTDAT3 BL   @FETCH1
       DATA DATAPT,DATLEN
       BL   @INTADD
       DATA DATLEN,D2,DATLEN
       BL   @MOVDAT
       DATA DATAPT,DATTP1,DATLEN
       BL   @INTADD
       DATA DATAPT,DATLEN,DATAPT
       BL   @INTDIV
```

```
       DATA DATAPT,D2,INTTP1
       BL   @INTPLY
       DATA INTTP1,D2,INTTP1
       BL   @INTSUB
       DATA INTTP1,DATAPT,INTTP1
       BL   @INTADD
       DATA INTTP1,DATAPT,DATAPT
GTDAT4 DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* ESTABLISH ARRAY        *
*                        *
* BL   @SETDIM           *
* DATA NAME,DN,V1,,,,LEN *
*                        *
* NAME = BUFFER NAME     *
* DN   = # OF DIMENSIONS *
* V1,, = MAX VAL OF DIM  *
* LEN  = LENGTH OF CELL  *
*------------------------*

SETDIM MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R1,R1
       MOV  R1,*R0+
SETDM1 CI   R1,0
       JEQ  SETDM2
       MOV  *R11+,R2
       MOV  *R2,*R0+
       DEC  R1
       JMP  SETDM1
SETDM2 MOV  *R11+,R2
       MOV  *R2,*R0
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       RT

*------------------------*
* CLEAR OUT ARRAY        *
*                        *
* BL   @CLRDIM           *
* DATA NAME              *
*                        *
* SETS ALL CELLS TO >00  *
*------------------------*

CLRDIM MOV  *R11+,R0
       MOV  R0,@DIMSRC
```

```
       MOV  *R0,R0
       MOV  R0,@DIMNUM

       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT

* SET LOOP START VALUE

       MOV  @DIMSRC,@DMIN1
       BL   @INTADD
       DATA DMIN1,D4,DMIN1

* SET LOOP END VALUE

       BL   @FETCH1
       DATA DIMSRC,DIMTP1
       BL   @INTADD
       DATA DIMTP1,D1,DIMTP1
       BL   @INTPLY
       DATA DIMTP1,D2,DIMTP1
       BL   @INTADD
       DATA DIMSRC,DIMTP1,DMIN2

* CALC TOTAL BYTES IN ARRAY

       BL   @FETCH1
       DATA DMIN1,DMADR0
       BL   @FOR
       DATA DMINLP,DMIN1,DMIN2,D2
       BL   @FETCH1
       DATA DMINLP,DMADR1
       BL   @INTPLY
       DATA DMADR1,DMADR0,DMADR0
       BL   @NEXT
       DATA DMINLP

* CALC FIRST/LAST WORDS

       BL   @INTADD
       DATA DMIN2,D2,DMIN2
       MOV  @DMIN2,@DIMTP1
       BL   @INTADD
       DATA DMADR0,DMIN2,DIMTP2

* CLEAN OUT STOARGE AREA

       LI   R0,D0
       MOV  R0,@DMIN1
       BL   @FOR
```

```
       DATA DMINLP,DIMTP1,DIMTP2,D2
       BL   @MOVDAT
       DATA DMIN1,DMINLP,D2
       BL   @NEXT
       DATA DMINLP

       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* GET A VAR FROM ARRAY   *
*                        *
* BL   @GETDIM           *
* DATA NAME,V1,,,,STORE  *
*------------------------*

GETDIM MOV  *R11+,R0
       MOV  R0,@DIMSRC
       MOV  *R0,R0
       MOV  R0,@DIMNUM
       LI   R1,DIMBUF
GETDM1 CI   R0,0
       JEQ  GETDM2
       MOV  *R11+,R2
       MOV  *R2,*R1+
       DEC  R0
       JMP  GETDM1
GETDM2 MOV  *R11+,@DIMDST
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       BL   @PICDIM
       BL   @FETCH1
       DATA DMIN2,DIMTP1
       BL   @MOVDAT
       DATA DMADR0,DIMDST,DIMTP1
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* PUT A VAR INTO ARRAY   *
*                        *
* BL   @PUTDIM           *
* DATA NAME,V1,,,,VARVAL *
*------------------------*

PUTDIM MOV  *R11+,R0
```

```
       MOV  R0,@DIMSRC
       MOV  *R0,R0
       MOV  R0,@DIMNUM
       LI   R1,DIMBUF
PUTDM1 CI   R0,0
       JEQ  PUTDM2
       MOV  *R11+,R2
       MOV  *R2,*R1+
       DEC  R0
       JMP  PUTDM1
PUTDM2 MOV  *R11+,@DIMDST
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       BL   @PICDIM
       BL   @FETCH1
       DATA DMIN2,DIMTP1
       BL   @MOVDAT
       DATA DIMDST,DMADR0,DIMTP1
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
       LIST
*------------------------*
* CALC ARRAY ADDRESS     *
*                        *
* BL   @PICDIM           *
*                        *
* SUPPORT ROUTINE ONLY   *
*------------------------*

PICDIM MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT

* SET OUTER LOOP VALUES

       LI   R0,DIMBUF
       MOV  R0,@DMOUT1
       BL   @INTPLY
       DATA DIMNUM,D2,DIMTP2
       BL   @INTADD
       DATA DIMTP2,DMOUT1,DMOUT2

* SET INNER LOOP VALUES

       MOV  @DIMSRC,@DMIN1
       BL   @INTADD
       DATA DMIN1,D2,DMIN1
```

```
       BL   @INTADD
       DATA DIMTP2,DMIN1,DMIN2

* CALC ARRAY CELL ADDRESS

       BL   @INTADD
       DATA DMIN1,D2,DMIN1
       BL   @FOR
       DATA DMUTLP,DMOUT1,DMOUT2,D2
       BL   @FETCH1
       DATA DMIN1,DMADR1
       BL   @INTADD
       DATA DMIN1,D2,DMIN1
       BL   @FOR
       DATA DMINLP,DMIN1,DMIN2,D2
       BL   @FETCH1
       DATA DMIN1,DMADR2
       BL   @INTPLY
       DATA DMADR1,DMADR2,DMADR1
       BL   @NEXT
       DATA DMINLP
       BL   @FETCH1
       DATA DMUTLP,DMADR2
       BL   @INTPLY
       DATA DMADR1,DMADR2,DMADR1
       BL   @INTADD
       DATA DMADR0,DMADR1,DMADR0
       BL   @NEXT
       DATA DMUTLP

* CALC DIM BLOCK OFFSET VALUE

       BL   @INTADD
       DATA DIMNUM,D2,DIMTP1
       BL   @INTPLY
       DATA DIMTP1,D2,DIMTP1
       BL   @INTADD
       DATA DMADR0,DIMTP1,DMADR0
       BL   @INTADD
       DATA DMADR0,DIMSRC,DMADR0

       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
       UNL
*------------------------*
* GET POINTED VALUE      *
*                        *
* BL   @FETCH1           *
* DATA SRCWRD,DSTWRD     *
```

```
*                         *
* PLACES VAL POINTED TO  *
* BY SCRWRD INTO DSTWRD  *
*-------------------------*

FETCH1 MOV   *R11+,R0
       MOV   *R11+,R1
       MOV   *R0,R0
       MOV   *R0,*R1
       RT

*
* FETCH2 PLACES VAL POINTED
* TO BY THE VAL IN SRCWRD
* INTO DSTWRD!
*

FETCH2 MOV   *R11+,R0
       MOV   *R11+,R1
       MOV   *R0,R0
       MOV   *R0,R0
       MOV   *R0,*R1
       RT

*-------------------------*
* INIT VAR - IMMEDIATE   *
*                         *
* BL   @LETI             *
* DATA VAR,N             *
*                         *
* PLACES THE VAL N INTO  *
* THE VARBL - WORD ONLY! *
*-------------------------*

LETI   MOV *R11+,R0
       MOV *R11+,R1
       MOV R1,*R0
       RT
```

Disk 41. Contents of file FLOAT/S

```
*------------------------*
* SIMPLE MATH ROUTINES   *
*                        *
* BL   @FPMATH           *
* DATA FCTN,V1,V2,RESULT *
*------------------------*

FPMATH MOV  *R11+,R0
       MOV  *R11+,@FLTTP1
       MOV  *R11+,@FLTTP2
       MOV  *R11+,@FLTTP3
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@FPMTH1
       BL   @MOVDAT
       DATA FLTTP1,FACADR,D8
       BL   @MOVDAT
       DATA FLTTP2,ARGADR,D8
       CLR  @STATUS
       BLWP @XMLLNK
FPMTH1 DATA >0000
       BL   @MOVDAT
       DATA FACADR,FLTTP3,D8
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* SPECIAL MATH FUNCTIONS *
*                        *
* BL   @FPFCTN           *
* DATA FCTN,VAR1,RESULT  *
*------------------------*

FPFCTN MOV  *R11+,R0
       MOV  *R11+,@FLTTP1
       MOV  *R11+,@FLTTP2
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@FPFCT1
       BL   @MOVDAT
       DATA FLTTP1,FACADR,D8
       CLR  @STATUS
       BLWP @GPLLNK
FPFCT1 DATA >0000
```

```
       BL   @MOVDAT
       DATA FACADR,FLTTP2,D8
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* INVOLUTION ROUTINE     *
*                        *
* BL   @FPINVL           *
* DATA EXPV,BASEV,RESULT *
*------------------------*

FPINVL MOV  *R11+,@FLTTP1
       MOV  *R11+,@FLTTP2
       MOV  *R11+,@FLTTP3
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       BL   @MOVDAT
       DATA FLTTP1,FACADR,D8
       BL   @MOVDAT
       DATA FLTTP2,ARGADR,D8
       CLR  @STATUS
       BLWP @GPLLNK
       DATA INV
       BL   @MOVDAT
       DATA FACADR,FLTTP3,D8
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* CONVERT FLT TO INT     *
*                        *
* BL   @FLTINT           *
* DATA FLTVAL,INTVAL     *
*------------------------*

FLTINT MOV  *R11+,@FLTTP1
       MOV  *R11+,@FLTTP2
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       BL   @MOVDAT
       DATA FLTTP1,FACADR,D8
FLINT2 CLR  @STATUS
       BLWP @XMLLNK
```

```
       DATA CFI
       MOV  @FLTTP2,R1
       MOV  @FAC,*R1
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* CONVERT INT TO FLT     *
*                        *
* BL   @INTFLT           *
* DATA INTVAL,FLTVAL     *
*------------------------*

INTFLT MOV  *R11+,R0
       MOV  *R11+,@FLTTP1
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@FAC
       CLR  @STATUS
       BLWP @XMLLNK
       DATA CIF
       BL   @MOVDAT
       DATA FACADR,FLTTP1,D8
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* CONVERT STR TO NUM     *
*                        *
* BL   @STRNUM           *
* DATA STR,FLTNUM        *
*------------------------*

STRNUM MOV  *R11+,R1
       MOV  *R11+,@FLTTP1
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       LI   R0,VDPFLT
       MOV  *R1+,R2
       BLWP @VMBW
       MOV  R0,@FAC+12
       BLWP @XMLLNK
       DATA CSN
       BL   @MOVDAT
       DATA FACADR,FLTTP1,D8
```

```
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* CONVERT NUM TO STR     *
*                        *
* BL   @NUMSTR           *
* DATA FLTNUM,STR        *
* LEN OF STR IN WORD 1!  *
*------------------------*

NUMSTR MOV  *R11+,@FLTTP1
       MOV  *R11+,@FLTTP2
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       BL   @MOVDAT
       DATA FLTTP1,FACADR,D8
       CLR  @STATUS
       BLWP @GPLLNK
       DATA STR
       CLR  R0
       MOV  @FLTTP2,R1
       CLR  R2
       MOVB @FAC+11,R0
       SWPB R0
       AI   R0,>8300
       MOVB @FAC+12,R2
       SWPB R2
       MOV  R2,*R1+
       A    R0,R2
NMSTR1 C    R0,R2
       JEQ  NMSTR2
       MOVB *R0+,*R1+
       JMP  NMSTR1
NMSTR2 CLR  R3
       MOVB R3,@FAC+11
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* NUMSTR IN FIX MODE     *
*                        *
* BL   @FIXNUM           *
* DATA SIGL,SIGR         *
*------------------------*
```

```
FIXNUM MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R0,R0
       MOV  *R1,R1
       SWPB R0
       A    R0,R1
       LI   R0,>0100
       MOVB R0,@FAC+11
       MOV  R1,@FAC+12
       RT
```

Disk 41. Contents of file GRAPH1/S

```
*-------------------------*
* REDEFINE CHAR CODE     *
*                        *
* BL   @DEFCHR           *
* DATA CHRVAL,CHRDAT     *
*-------------------------*

DEFCHR MOV  *R11+,R1
       MOV  *R11+,R3
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       LI   R0,CHRTAB
       MOV  *R1+,R2
       MOV  *R3,@INTTP1
       BL   @INTPLY
       DATA D8,INTTP1,INTTP1
       A    @INTTP1,R0
       BLWP @VMBW
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*-------------------------*
* PUT CHAR ON SCREEN     *
*                        *
* BL   @PUTCHR           *
* DATA ROW,COL,CHR       *
*-------------------------*

PUTCHR MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,R3
       MOV  *R1,R0
       MOV  *R2,R1
       SWPB R1
       MPY  @SCRDTH,R3
       A    R4,R0
       BLWP @VSBW
       RT

*-------------------------*
* LOAD CHARSETS          *
*                        *
* BL   @CHRSET           *
* DATA SET,ADR           *
```

```
*------------------------*

CHRSET MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R0,@CHSET1
       MOV  *R1,@FAC
       CLR  @STATUS
       BLWP @GPLLNK
CHSET1 DATA >0000
       RT

*------------------------*
* SET ALL CHRSET COLORS  *
*                        *
* BL   @SETCOL           *
* DATA FG,BG (0-15)      *
*                        *
* SETS CHAR CODES 0-255  *
* TO SAME COLOR COMBO    *
*------------------------*

SETCOL MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@COLTP1
       MOV  *R1,@COLTP2
       BL   @SCRCOL
       DATA COLTP2
       BL   @FOR
       DATA COLTP3,D0,D31,D1
       BL   @CHRCOL
       DATA COLTP3,COLTP1,COLTP2
       BL   @NEXT
       DATA COLTP3
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* SET SCREEN COLOR       *
*                        *
* BL   @SCRCOL           *
* DATA COLOR (0-15)      *
*------------------------*

SCRCOL MOV  *R11+,R1
       MOV  *R1,R0
       AI   R0,>0700
```

```
       BLWP @VWTR
       RT

*-------------------------*
* SET CHARACTER COLOR    *
*                        *
* BL   @CHRCOL           *
* DATA C(0-31),F,B(0-15) *
*-------------------------*

CHRCOL MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  *R11+,R4
       MOV  *R2,R0
       AI   R0,COLTAB
       CLR  R1
       CLR  R2
       MOV  *R3,R2
       SLA  R2,12
       MOV  *R4,R1
       SWPB R1
       A    R2,R1
       BLWP @VSBW
       RT

*-------------------------*
* REPEAT CHARACTER       *
*                        *
* BL   @REPCHR           *
* DATA R,C,CH,RP,ST      *
*                        *
* R,C = START ROW & COL  *
* CH  = ASCII CHAR CODE  *
* RP  = # OF REPITITIONS *
* ST  = STEP VAL FOR REP *
*-------------------------*

REPCHR MOV  *R11+,R2
       MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R4
       MOV  *R11+,R6
       MOV  *R0,R0
       MOV  *R1,R1
       MOV  *R2,R2
       MOV  *R4,R4
       MOV  *R6,R6
       MPY  @SCRDTH,R2
       A    R3,R0
       MPY  R6,R4
```

```
        A     R0,R5
        SWPB  R1
RPCHR1  C     R5,R0
        JEQ   RPCHR2
        BLWP  @VSBW
        A     R6,R0
        JMP   RPCHR1
RPCHR2  RT
```

Disk 41. Contents of file IFSET/S

```
*------------------------*
* IF THEN ROUTINE        *
*                        *
* BL   @IFTHEN           *
* DATA VAR1,CND,VAR2,ACT *
*                        *
* VAR1 = VAR TO CHECK    *
* CND  = LT,EQ,GT,NE     *
* VAR2 = VAR TO CHECK    *
* ACT  = ACTION TO TAKE  *
*------------------------*

IFTHEN MOV  *R11+,R0
       MOV  *R11+,R3
       MOV  *R11+,R1
       MOV  *R11+,R2
       C    *R3,@EQ
       JEQ  IFEQ
       C    *R3,@LT
       JEQ  IFLT
       C    *R3,@GT
       JEQ  IFGT
IFNE   C    *R0,*R1
       JNE  IFTRUE
       JMP  IFALSE
IFEQ   C    *R0,*R1
       JEQ  IFTRUE
       JMP  IFALSE
IFLT   C    *R0,*R1
       JLT  IFTRUE
       JMP  IFALSE
IFGT   C    *R0,*R1
       JGT  IFTRUE
IFALSE RT
IFTRUE B    *R2

*------------------------*
* IF-THEN-ELSE ROUTINE   *
*                        *
* BL   @IFELSE           *
* DATA V1,CND,V2,ACT,ELSE*
*                        *
* V1   = VAR TO CHECK    *
* CND  = LT,EQ,GT,NE     *
* V2   = VAR TO CHECK    *
* ACT  = ACTION IF TRUE  *
* ELSE = ACTION IF FALSE *
*------------------------*
```

```
IFELSE MOV  *R11+,R0
       MOV  *R11+,R3
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R4
       C    *R3,@EQ
       JEQ  ELEQ
       C    *R3,@LT
       JEQ  ELLT
       C    *R3,@GT
       JEQ  ELGT
ELNE   C    *R0,*R1
       JNE  ELTRUE
       JMP  ELFALS
ELEQ   C    *R0,*R1
       JEQ  ELTRUE
       JMP  ELFALS
ELLT   C    *R0,*R1
       JLT  ELTRUE
       JMP  ELFALS
ELGT   C    *R0,*R1
       JGT  ELTRUE
ELFALS B    *R4
ELTRUE B    *R2

*------------------------*
* IF-OR THEN             *
*                        *
*BL   @IFOR              *
*DATA V1,C1,V2,V3,C2,V4,A*
*                        *
* V? = VARIABLES         *
* C? = COMPARE VALUE     *
* A  = ACTION IF TRUE    *
*------------------------*

IFOR   MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  *R11+,R4
       MOV  *R11+,R5
       MOV  *R11+,R6
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@IFTPV1
       MOV  *R1,@IFTPC1
       MOV  *R2,@IFTPV2
       MOV  *R3,@IFTPV3
       MOV  *R4,@IFTPC2
```

```
       MOV  *R5,@IFTPV4
       BL   @IFTHEN
       DATA IFTPV1,IFTPC1,IFTPV2,IFORTR
       BL   @IFTHEN
       DATA IFTPV3,IFTPC2,IFTPV4,IFORTR
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
IFORTR DECT @RETPT
       B    *R6

*------------------------*
* IF-AND THEN            *
*                        *
*BL   @IFAND             *
*DATA V1,C1,V2,V3,C2,V4,A*
*                        *
* V? = VARIABLES         *
* C? = COMPARE VALUE     *
* A  = ACTION IF TRUE    *
*------------------------*

IFAND  MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  *R11+,R4
       MOV  *R11+,R5
       MOV  *R11+,R6
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@IFTPV1
       MOV  *R1,@IFTPC1
       MOV  *R2,@IFTPV2
       MOV  *R3,@IFTPV3
       MOV  *R4,@IFTPC2
       MOV  *R5,@IFTPV4
       BL   @IFTHEN
       DATA IFTPV1,IFTPC1,IFTPV2,IFAND1
       JMP  IFAND2
IFAND1 BL   @IFTHEN
       DATA IFTPV3,IFTPC2,IFTPV4,IFAND3
IFAND2 DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
IFAND3 DECT @RETPT
       B    *R6
```

Disk 41. Contents of file INIT-EA/S

```
*************************
*                       *
*    PULSAR UTILITIES   *
*      VERSION 1.2      *
*        01/18/85       *
*     ED/ASM VERSION    *
*                       *
*   EXTENDED REFS LIST  *
*FOR LANGUAGE DEVELOPMENT*
*************************

*
* HOUSEKEEPING
*

*--------------------*
* AVAILABLE ROUTINES *
*--------------------*

* FLOATING POINT ROUTINES

       DEF  FPMATH,FPFCTN
       DEF  FPINVL,FLTINT,INTFLT
       DEF  STRNUM,NUMSTR,FIXNUM

* GRAPHICS ROUTINES

       DEF  DEFCHR,PUTCHR,CHRSET
       DEF  SETCOL,SCRCOL,CHRCOL
       DEF  REPCHR

* FLOW CONTROL ROUTINES

       DEF  IFTHEN,IFELSE,IFOR,IFAND
       DEF  FOR,NEXT

* INTEGER MATH ROUTINES

       DEF  INTADD,INTSUB,INTPLY,INTDIV

* KEYSTROKE ROUTINES

       DEF  GETKEY,KEYCON,KEYLET,KEYNUM

* MEMORY ACCESS ROUTINES

       DEF  FETCH1,FETCH2,LETI
       DEF  GETVDP,PUTVDP,MOVCPU,MOVVDP
```

```
       DEF  MOVDAT,GETDAT,SETDAT
       DEF  SETDIM,CLRDIM,GETDIM,PUTDIM

* SCREEN I/O ROUTINES

       DEF  SCRCLR,PRNTAT,GETSCR,INPTAT

* MISCELLANEOUS ROUTINES

       DEF  RANDOM,SETRND,RND

*---------------------*
* AVAILABLE ADDRESSES *
*---------------------*

* FUNCTION KEY VALUES

       DEF  FCTN1,FCTN2,FCTN3,FCTN4,FCTN5
       DEF  FCTN6,FCTN7,FCTN8,FCTN9,FCTN0
       DEF  QUITKY,ENTERV,FCTNS,FCTND

* FLOATING POINT VALUES

       DEF  FADD,FSUB,FMUL,FDIV
       DEF  INT,INV,SQR,EXP,LOG
       DEF  COS,SIN,TAN,ATN
       DEF  STR,CSN,CFI,CIF

* STACK AREAS & POINTERS

       DEF  RETADR,BLPSTK,ELPSTK
       DEF  RLPSTK,SLPSTK,DATAPT
       DEF  RETPT,LOPPT,RETBUF

* TEMPORARY AREAS

       DEF  IFTPV1,IFTPV2,IFTPV3
       DEF  IFTPV4,IFTPC1,IFTPC2
       DEF  INTTP1,INTTP2,INTTP3,INTTP4
       DEF  FLTTP1,FLTTP2,FLTTP3,VDPFLT

* PREDEFINED INTEGERS

       DEF  D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10
       DEF  D11,D12,D13,D14,D15,D16,D17,D18
       DEF  D19,D20,D21,D22,D23,D24,D25,D26
       DEF  D27,D28,D29,D30,D31,D32,D33,D34
       DEF  D35,D36,D37,D38,D39,D40,D48

* MISCELLANEOUS VALUES
```

```
       DEF  LRGSET,SMLSET,REGSET
       DEF  SCRTAB,COLTAB,CHRTAB,SCRDTH
       DEF  SAVEXT,KEYHIT,NOKEY
       DEF  LT,GT,EQ,NE
       DEF  KEYADR,KEYVAL,STATUS
       DEF  FAC,ARG,FACADR,ARGADR
       DEF  DATYPE,DATLEN,DATTP1
       DEF  INTDAT,FLTDAT,STRDAT

*-------------*
* ED/ASM REFS *
*-------------*

       REF  VSBW,VMBW,VSBR,VMBR
       REF  VWTR,KSCAN,LOADER
       REF  DSRLNK,GPLLNK,XMLLNK
       REF  GPLWS

* ADDED WITH "BASIC SUPPORT"

       REF  NUMASG,NUMREF
       REF  STRASG,STRREF,ERR

* WORKSPACE AND STACKS

WRKSPC BSS  >0020
RETADR BSS  >0080
BLPSTK BSS  >0080
ELPSTK BSS  >0080
SLPSTK BSS  >0080
RLPSTK BSS  >0080

* POINTERS AND RESERVED AREAS

DATAPT DATA 0
RETPT  DATA 0
LOPPT  DATA 0
SCRDTH DATA 32
SAVEXT DATA 0
KEYHIT DATA 0
NOKEY  DATA -1
INPROW DATA 0
INPCOL DATA 0
INPLEN DATA 0
INSTRT DATA 0
RETBUF DATA 0
STRLEN DATA 0
STRADR DATA 0
LT     DATA 1
EQ     DATA 2
```

```
GT     DATA 3
NE     DATA 4
IFTPV1 DATA 0
IFTPV2 DATA 0
IFTPV3 DATA 0
IFTPV4 DATA 0
IFTPC1 DATA 0
IFTPC2 DATA 0
INTTP1 DATA 0
INTTP2 DATA 0
INTTP3 DATA 0
INTTP4 DATA 0
COLTP1 DATA 0
COLTP2 DATA 0
COLTP3 DATA 0
MVVDP1 DATA 0
MVVDP2 DATA 0
DATYPE DATA 0
DATLEN DATA 0
DATTP1 DATA 0
INTDAT DATA 2
FLTDAT DATA 8
STRDAT DATA 1
DIMBUF BSS  8
DIMDST DATA 0
DIMSRC DATA 0
DIMNUM DATA 0
DIMADR DATA 0
DMOUT1 DATA 0
DMOUT2 DATA 0
DMIN1  DATA 0
DMIN2  DATA 0
DMADR0 DATA 0
DMADR1 DATA 0
DMADR2 DATA 0
DMINLP DATA 0
DMUTLP DATA 0
DIMTP1 DATA 0
DIMTP2 DATA 0
KYNUM1 DATA 0
KYNUM2 DATA 0
FCTN1  DATA 3
FCTN2  DATA 4
FCTN3  DATA 7
FCTN4  DATA 2
FCTN5  DATA 14
FCTN6  DATA 12
FCTN7  DATA 1
FCTN8  DATA 6
FCTN9  DATA 15
FCTN0  DATA 188
```

```
QUITKY DATA 5
ENTERV DATA 13
FCTNS  DATA 8
FCTND  DATA 9
LRGSET DATA >0016
REGSET DATA >0018
SMLSET DATA >004A

* SYSTEM EQUATED ADDRESSES

KEYADR EQU  >8374
KEYVAL EQU  >8375
STATUS EQU  >837C
FAC    EQU  >834A
ARG    EQU  >835C
SCRTAB EQU  >0000
COLTAB EQU  >0380
CHRTAB EQU  >0800

* FLOATING POINT DATA

FACADR DATA >834A
ARGADR DATA >835C
STR    EQU  >0014
CSN    EQU  >1000
CFI    EQU  >1200
CIF    EQU  >2300
FADD   DATA >0600
FSUB   DATA >0700
FMUL   DATA >0800
FDIV   DATA >0900
INT    DATA >0022
INV    EQU  >0024
SQR    DATA >0026
EXP    DATA >0028
LOG    DATA >002A
COS    DATA >002C
SIN    DATA >002E
TAN    DATA >0030
ATN    DATA >0032
FLTTP1 DATA 0
FLTTP2 DATA 0
FLTTP3 DATA 0
VDPFLT EQU  >0F80        >TEMP BUFFER

* PREDEFINED CONSTANTS

D0     DATA 0
D1     DATA 1
D2     DATA 2
```

```
D3      DATA 3
D4      DATA 4
D5      DATA 5
D6      DATA 6
D7      DATA 7
D8      DATA 8
D9      DATA 9
D10     DATA 10
D11     DATA 11
D12     DATA 12
D13     DATA 13
D14     DATA 14
D15     DATA 15
D16     DATA 16
D17     DATA 17
D18     DATA 18
D19     DATA 19
D20     DATA 20
D21     DATA 21
D22     DATA 22
D23     DATA 23
D24     DATA 24
D25     DATA 25
D26     DATA 26
D27     DATA 27
D28     DATA 28
D29     DATA 29
D30     DATA 30
D31     DATA 31
D32     DATA 32
D33     DATA 33
D34     DATA 34
D35     DATA 35
D36     DATA 36
D37     DATA 37
D38     DATA 38
D39     DATA 39
D40     DATA 40
D48     DATA 48
```

Disk 41. Contents of file INTMATH/S

```
*------------------------*
* INTEGER ADDITION       *
*                        *
* BL   @INTADD           *
* DATA VAR1,VAR2,RSLT    *
*------------------------*

INTADD MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,R0
       MOV  *R1,R1
       A    R0,R1
       MOV  R1,*R2
       RT

*------------------------*
* INTEGER SUBTRACTION    *
*                        *
* BL   @INTSUB           *
* DATA VAR1,VAR2,RSLT    *
*------------------------*

INTSUB MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R0,R0
       MOV  *R1,R1
       S    R0,R1
       MOV  R1,*R2
       RT

*------------------------*
* INTEGER MULTIPLICATION *
*                        *
* BL   @INTPLY           *
* DATA VAR1,VAR2,RSLT    *
*------------------------*

INTPLY MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R1,R3
       MPY  *R0,R3
       MOV  R4,*R2
       RT

*------------------------*
```

```
* INTEGER DIVISION       *
*                        *
* BL   @INTDIV           *
* DATA VAR1,VAR2,RSLT    *
* ->RSLT+2 = REMAINDER   *
*------------------------*

INTDIV MOV  *R11+,R1
       MOV  *R11+,R0
       MOV  *R11+,R3
       MOV  *R1,R2
       CLR  R1
       DIV  *R0,R1
       MOV  R1,*R3+
       MOV  R2,*R3
       RT
```

Disk 41. Contents of file KEYSET/S

```
*------------------------*
* GET A KEYSTROKE        *
*                        *
* BL   @GETKEY           *
* DATA VAR1              *
*------------------------*

GETKEY MOV  *R11+,R0
       MOV  @NOKEY,*R0
       CLR  @STATUS
       BLWP @KSCAN
       MOVB @STATUS,@STATUS
       JEQ  GTKEY1
       CLR  *R0
       MOVB @KEYVAL,*R0
       SWPB *R0
GTKEY1 RT

*------------------------*
* WAIT UNTIL KEY IS HIT  *
*                        *
* BL   @KEYCON           *
*------------------------*

KEYCON MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
KYCON2 BL   @GETKEY
       DATA KEYHIT
       BL   @IFTHEN
       DATA KEYHIT,EQ,NOKEY,KYCON2
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
*------------------------*
* SELECT FOR NUMERIC KEY *
*                        *
* BL   @KEYNUM           *
* DATA NUMLO,NUMHI       *
*------------------------*

KEYTP1 DATA 0
KEYTP2 DATA 0
KEYNUM MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
```

```
       INCT @RETPT
       MOV  *R0,@KEYTP1
       MOV  *R1,@KEYTP2
       BL   @INTADD
       DATA KEYTP1,D48,KEYTP1
       BL   @INTADD
       DATA KEYTP2,D48,KEYTP2
KYNUM4 BL   @KEYCON
       BL   @IFTHEN
       DATA KEYHIT,LT,KEYTP1,KYNUM4
       BL   @IFTHEN
       DATA KEYHIT,GT,KEYTP2,KYNUM4
       BL   @INTSUB
       DATA D48,KEYHIT,KEYHIT
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* SELECT FOR ASCII KEY   *
*                        *
* BL   @KEYLET           *
* DATA LETBUF            *
*                        *
* LETBUF = LIST BUF ADR  *
* LEN = WORD 1 OF LETBUF *
*------------------------*


KEYLET MOV  *R11+,R0
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0+,@STRLEN
       MOV  R0,@STRADR
       BL   @INTSUB
       DATA D1,STRLEN,STRLEN
KYLET1 BL   @KEYCON
       BL   @FOR
       DATA KEYTP1,D0,STRLEN,D1
       MOV  @STRADR,R0
       CLR  @KEYTP2
       MOVB *R0,@KEYTP2
       SWPB @KEYTP2
       BL   @IFTHEN
       DATA KEYHIT,EQ,KEYTP2,KYLET2
       INC  @STRADR
       BL   @NEXT
       DATA KEYTP1
       JMP  KYLET1
```

```
KYLET2 DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
```

Disk 41. Contents of file LOOPSET/S

```
*-------------------------*
* FOR-NEXT-STEP           *
*                         *
* BL   @FOR               *
* DATA VAR1,BGN,END,STP   *
*                         *
* BL   @NEXT              *
* DATA VAR1               *
*-------------------------*

FOR    MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  @LOPPT,R9
       MOV  *R1,@BLPSTK(R9)
       MOV  *R2,@ELPSTK(R9)
       MOV  *R3,@SLPSTK(R9)
       MOV  R11,@RLPSTK(R9)
       MOV  @BLPSTK(R9),*R0
       INCT @LOPPT
       RT

NEXT   MOV  *R11+,R0
       DECT @LOPPT
       MOV  @LOPPT,R9
       C    @BLPSTK(R9),@ELPSTK(R9)
       JHE  NEXT1
       A    @SLPSTK(R9),@BLPSTK(R9)
       MOV  @BLPSTK(R9),*R0
       MOV  @RLPSTK(R9),R11
       INCT @LOPPT
NEXT1  RT
```

Disk 41. Contents of file PULSAR/S

```
*******************************
*                             *
* PULSAR MAIN ASSEMBLY DRIVER *
*                             *
*    Version 1.2  1/18/85     *
*                             *
*     Michael C. Amundsen     *
*  New Horizons Users' Group  *
*                             *
*******************************

       UNL
       COPY "DSK.PULSAR1.INIT-EA/S"
       COPY "DSK.PULSAR1.DATASET/S"
       COPY "DSK.PULSAR1.IFSET/S"
       COPY "DSK.PULSAR1.LOOPSET/S"
       COPY "DSK.PULSAR1.INTMATH/S"
       COPY "DSK.PULSAR1.FLOAT/S"
       COPY "DSK.PULSAR1.SCRN-IO/S"
       COPY "DSK.PULSAR1.KEYSET/S"
       COPY "DSK.PULSAR1.GRAPH1/S"
       COPY "DSK.PULSAR1.RANDSET/S"
       LIST
       END
```

Disk 41. Contents of file RANDSET/S

```
*------------------------*
* RANDOM NUM GENERATOR   *
*                        *
* BL   @RANDOM           *
*                        *
* RND = FLT PT VAL (0-1) *
*------------------------*

RND    BSS  8
RND1   BSS  8
RND2   DATA >4101,>0000,>0000,>0000
RND3   DATA >4202,>2A62,>0000,>0000
RND4   DATA >4209,>635B,>0000,>0000
RND5   DATA >4213,>5A11,>0000,>0000
SEED   DATA 0

RANDOM MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT

       BL   @IFTHEN
       DATA SEED,EQ,D0,RSEED
       BL   @FPMATH
       DATA FMUL,RND,RND2,RND
       BL   @FPMATH
       DATA FMUL,RND,RND3,RND1
       BL   @FPMATH
       DATA FADD,RND1,RND4,RND1
       BL   @FPMATH
       DATA FDIV,RND5,RND1,RND1
       BL   @FPFCTN
       DATA INT,RND1,RND
       BL   @FPMATH
       DATA FSUB,RND,RND1,RND

       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* GET NEW RANDOM SEED    *
*                        *
* BL   @RSEED            *
*                        *
* SUPPORT ROUTINE ONLY   *
*------------------------*
```

```
RSEED  MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT

       MOV  @>8378,@SEED
       BL   @INTFLT
       DATA SEED,RND

       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*-------------------------*
* USER SET RANDOM SEED   *
*                        *
* BL   @SETRND           *
* DATA INTVAL            *
*-------------------------*

SETRND MOV  *R11+,R0
       MOV  *R0,@SEED
       RT
```

Disk 41. Contents of file SCRN-IO/S

```
*-------------------------*
* CLEAR SCREEN           *
*                        *
* BL   @SCRCLR           *
*-------------------------*

SCRCLR LI   R0,0
       LI   R1,>20
       LI   R2,24
       CLR  R3
       MPY  @SCRDTH,R2
SCRCL1 BLWP @VSBW
       INC  R0
       C    R0,R3
       JNE  SCRCL1
       RT

*-------------------------*
* PRINT AT ROUTINE       *
*                        *
* BL   @PRNTAT           *
* DATA ROW,COL,STR       *
*                        *
* ROW = 0-23             *
* COL = 0-SCRDTH(32,40)  *
* STR = TEXT TO PRINT    *
* LEN IN WORD 1 OF STR!  *
*-------------------------*

PRNTAT MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@INTTP1
       MOV  *R1,@INTTP2
       MOV  *R2+,@STRLEN
       MOV  R2,@STRADR
       BL   @INTPLY
       DATA SCRDTH,INTTP1,INTTP1
       BL   @INTADD
       DATA INTTP1,INTTP2,INTTP2
       MOV  @INTTP2,R0
       MOV  @STRADR,R1
       MOV  @STRLEN,R2
       BLWP @VMBW
       DECT @RETPT
```

```
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* GET DATA FROM SCREEN   *
*                        *
* BL   @GETSCR           *
* DATA ROW,COL,BUF,LEN   *
*------------------------*

GETSCR MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@INTTP1
       MOV  *R1,@INTTP2
       MOV  *R2,@STRADR
       MOV  *R3,@STRLEN
       BL   @INTPLY
       DATA SCRDTH,INTTP1,INTTP1
       BL   @INTADD
       DATA INTTP1,INTTP2,INTTP2
       MOV  @INTTP2,R0
       MOV  @STRADR,R1
       MOV  @STRLEN,R2
       BLWP @VMBR
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT

*------------------------*
* INPUT AT ROUTINE       *
*                        *
* BL   @INPTAT           *
* DATA ROW,COL,LEN,BUF   *
*------------------------*

INPTAT MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       MOV  *R11+,R3
       MOV  @RETPT,R9
       MOV  R11,@RETADR(R9)
       INCT @RETPT
       MOV  *R0,@INPROW
```

```
       MOV  *R1,@INPCOL
       MOV  *R1,@INSTRT
       MOV  *R2,@INPLEN
       MOV  R3,@RETBUF
       INCT @RETBUF
       CLR  @STRLEN
INPGET BL   @PUTCHR
       DATA INPROW,INPCOL,D30
       BL   @KEYCON
       BL   @IFTHEN
       DATA KEYHIT,EQ,ENTERV,INPEXT
       BL   @IFTHEN
       DATA KEYHIT,EQ,FCTNS,INLEFT
       BL   @IFTHEN
       DATA KEYHIT,EQ,FCTND,INRITE
       BL   @IFTHEN
       DATA STRLEN,EQ,INPLEN,INPGET
       BL   @PUTCHR
       DATA INPROW,INPCOL,KEYHIT
       A    @D1,@INPCOL
       A    @D1,@STRLEN
       JMP  INPGET
INLEFT BL   @IFTHEN
       DATA STRLEN,EQ,D0,INPGET
       BL   @PUTCHR
       DATA INPROW,INPCOL,D32
       S    @D1,@INPCOL
       S    @D1,@STRLEN
       JMP  INPGET
INRITE BL   @IFTHEN
       DATA STRLEN,EQ,INPLEN,INPGET
       BL   @PUTCHR
       DATA INPROW,INPCOL,D32
       A    @D1,@INPCOL
       A    @D1,@STRLEN
       JMP  INPGET
INPEXT BL   @PUTCHR
       DATA INPROW,INPCOL,D32
       BL   @IFTHEN
       DATA INPCOL,EQ,INSTRT,INPXT1
       BL   @GETSCR
       DATA INPROW,INSTRT,RETBUF,STRLEN
INPXT1 DECT @RETBUF
       MOV  @RETBUF,R0
       MOV  @STRLEN,*R0
       DECT @RETPT
       MOV  @RETPT,R9
       MOV  @RETADR(R9),R11
       RT
```

Disk 41. Contents of file START-EA/S

```
       UNL
**************************
*                        *
*  PULSAR USER FILE INIT  *
*                        *
* ED/ASM VERSION  1/18/85 *
*                        *
**************************

*--------------------*
* AVAILABLE ROUTINES *
*--------------------*

* FLOATING POINT ROUTINES

       REF  FPMATH,FPFCTN
       REF  FPINVL,FLTINT,INTFLT
       REF  STRNUM,NUMSTR,FIXNUM

* GRAPHICS ROUTINES

       REF  DEFCHR,PUTCHR,CHRSET
       REF  SETCOL,SCRCOL,CHRCOL
       REF  REPCHR

* FLOW CONTROL ROUTINES

       REF  IFTHEN,IFELSE,IFOR,IFAND
       REF  FOR,NEXT

* INTEGER MATH ROUTINES

       REF  INTADD,INTSUB,INTPLY,INTDIV

* KEYSTROKE ROUTINES

       REF  GETKEY,KEYCON,KEYNUM,KEYLET

* MEMORY ACCESS ROUTINES

       REF  FETCH1,FETCH2,LETI
       REF  GETVDP,PUTVDP,MOVCPU,MOVVDP
       REF  MOVDAT,SETDAT,GETDAT
       REF  SETDIM,CLRDIM,PUTDIM,GETDIM

* SCREEN I/O ROUTINES

       REF  SCRCLR,PRNTAT,GETSCR,INPTAT
```

```
* MISCELLANEOUS ROUTINES

       REF  RANDOM,SETRND,RND

*---------------------*
* AVAILABLE ADDRESSES *
*---------------------*

* FUNCTION KEY VALUES

       REF  FCTN1,FCTN2,FCTN3,FCTN4,FCTN5
       REF  FCTN6,FCTN7,FCTN8,FCTN9,FCTN0
       REF  QUITKY,ENTERV,FCTNS,FCTND

* FLOATING POINT VALUES

       REF  FADD,FSUB,FMUL,FDIV
       REF  INT,INV,SQR,EXP,LOG
       REF  COS,SIN,TAN,ATN
       REF  STR,CSN,CFI,CIF

* STACK AREAS & POINTERS

       REF  RETADR,BLPSTK,ELPSTK
       REF  RLPSTK,SLPSTK,DATAPT
       REF  RETPT,LOPPT,RETBUF

* TEMPORARY AREAS

       REF  IFTPV1,IFTPV2,IFTPV3
       REF  IFTPV4,IFTPC1,IFTPC2
       REF  INTTP1,INTTP2,INTTP3,INTTP4
       REF  FLTTP1,FLTTP2,FLTTP3,VDPFLT

* PREDEFINED INTEGERS

       REF  D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10
       REF  D11,D12,D13,D14,D15,D16,D17,D18
       REF  D19,D20,D21,D22,D23,D24,D25,D26
       REF  D27,D28,D29,D30,D31,D32,D33,D34
       REF  D35,D36,D37,D38,D39,D40,D48

* MISCELLANEOUS VALUES

       REF  LRGSET,SMLSET,REGSET
       REF  SCRTAB,COLTAB,CHRTAB,SCRDTH
       REF  SAVEXT,KEYHIT,NOKEY
       REF  LT,GT,EQ,NE
       REF  KEYADR,KEYVAL,STATUS
       REF  FAC,ARG,FACADR,ARGADR
       REF  DATYPE,DATLEN,DATTP1
```

```
       REF  INTDAT,FLTDAT,STRDAT

       LIST

* WORKSPACE AND ENTRANCE

       DEF  RUN
WRKSPC BSS  >0020

RUN    LWPI WRKSPC
       MOV  R11,@SAVEXT
```

Disk 42. Universal Disassembler

Version: 2.2 *Author*: Rene LeBlanc
Requires: EA *Language*: Forth *Updated*: 01/11/86

The most versatile fairware disassembler. Allows disassembly of programs from disk in any format, in memory, and even off the cards in your expansion box. Very complete with some unique features. Load with E/A3, DSK1.UNIVERSAL.

```
dskdir. v2.0. 12-dec-96
Disk name               = DASSM
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >003 FORTHSAVE    39 PROGRAM    Y >027 038
002 >004 SYS-SCRNS   313 DIS/FIX128 Y >04d 283 >005 029
003 >002 UNIVERSAL     6 DIS/FIX 80 Y >022 005
```

Disk 43. Disk Utilities II

Version:
Author:
Requires: XB, EA
Language: AL, XB
Updated: 08/17/86

Two track copiers, Disk Information Manager, SuperDisk very fast assembly disk librarian, File Reader — will display any file type (except program) and convert to Display/Variable 80.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-43
Sectors total           = 360
Sectors used            = 218
Sectors available       = 140
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >009 -README       5 DIS/VAR 80   >0f0 004
002 >002 DIM          33 PROGRAM      >022 032
003 >003 DIM/DOC      24 DIS/VAR 80   >042 023
004 >004 FILEREADER   63 INT/VAR254   >059 062
005 >005 LOAD          3 PROGRAM      >097 002
006 >006 SUPERDISK    57 DIS/FIX 80   >099 056
007 >007 TRACK1       18 DIS/FIX 80   >0d1 017
008 >008 TRACK2       15 DIS/FIX 80   >0e2 014
```

Disk 43. Contents of file -README

Boston Computer Society
TI-99/4A Use Group
One Center Plaza
Boston, MA 02108

Public domain software disk #43
Disk Utilities II

DIM: Disk Information Manager. Run from Editor/Assembler option 5.

DIM/DOC: Documentation for Disk Information Manager.

FILEREADER: Runs in Extended BASIC.

LOAD: Load program for FILEREADER. Menu driven.

SUPERDISK: Load with Editor/Assembler option 3. Program name is SFIRST. Menu driven.

TRACK1: Track copier. Load with Editor/Assembler option 3. Program name is BACKUP. Requires 2 disk drives.

TRACK2: Another track copier. Load with Editor/Assembler option 3. Will automatically start. Requires 2 disk drives.

Most of these programs are freeware. If you use one of them please send the author a donation for his work. Thank you.

Disk 43. Contents of file DIM/DOC

Disk Information Manager

Operating Instructions

Once the drive number (1, 2 or 3) is selected, the five options available are listed below.

CTRL B	Back to drive select.
CTRL I	Initialize disk in drive selected.
CTRL L	Look at sectors in drive selected.
CTRL S	Save the disk name at NAME=.
ENTER	Catalog the disk in drive selected.

Look Options

CTRL 0	Looks at the first 128 bytes in the sector selected.
CTRL 8	Looks at the last 128 bytes in the sector selected.
CTRL S	Sector selection. Note that sector number is a hex value. **FCTN E** and **FCTN X** may be used to increase or decrease the sector number.
CTRL B	Back to drive select.
CTRL O	Send sector information to a printer or disk file. **CTRL Append** in this option allows you to add the information to the bottom of the file name on the cursor line. e.g. DSK2.SECTOR
CTRL H	Modify HEX values in the sector. Use **CTRL W** option to save the new values.
CTRL A	Modify ASCII values in the sector.
CTRL R	Sets **ENTER** option to sector read.
CTRL W	Sets **ENTER** option to sector write.
ENTER	Reads or writes disk sector.

Catalogue Options

FCTN E	Scrolls file names up.
FCTN X	Scrolls file names down.
CTRL A	Sets or resets all files less than 46 sectors long for copying or recording. a small "c" to the left of the file name indicates which files have been selected.
CTRL B	Back to drive select.
CTRL C	Copy file (max. 45 sectors) to a second drive if the **CTRL A** or **CTRL 0** option has been used previously ; or a selected file if those options haven't been used. The copy disk must be inserted in the drive selected before you enter the copy file name. If the **CTRL A** or **CTRL 0** options were used it will copy from drive 1 to drive 2 or vise-versa.
CTRL D	Deletes file on cursor line. Actual file deletion takes place with **CTRL S** option.
CTRL L	A listing of the disk sectors used by the file is displayed. **FCTN E** or **FCTN X** may be used to change the sector. **ENTER** will read and display sector values. Use **0-8** to change the range displayed.
CTRL Out	option lists the file sectors used by ALL files on a printer or disk file. **CTRL Modify** allows you to change sector values (refer to Look Options).
CTRL N	Catalogues new disk in drive selected.
CTRL O	Outputs catalogue information to a disk or printer. The Save option in this mode stores the catalogue in a file named *CATALOGUE. This information may be retrieved using the Load option.
CTRL P	If a file is write protected, this option unprotects it or vise-versa.
CTRL R	Records files selected by **CTRL 0** or **CTRL A** options on CS1 and/or CS2.
CTRL S	Saves information from the **CTRL P** or **CTRL D** or **ENTER** options on the disk. A ">" symbol to the left of the file name indicates there is information to be saved on the disk.
CTRL T	Tabs over to the 1.INFO, 2.USE or 3.DATE section to allow modification of that data. Use **CTRL T** to tab back to the file name.
CTRL 0	Selects or de-selects file on cursor for the Cpy or Rec options.
CTRL 1	Selects the 1.INFO data for display. This information may be any 8 character file extension information.
CTRL 2	Selects the 2.USE data for display. One of 6 file types may be selected.
CTRL 3	Selects the 3.DATE data for display. The date must be between JAN01/80 and DEC31/95 and entered in that format to be accepted.
CTRL 4	Resets to the top of the catalogue.
CTRL 5	Moves cursor line half way up the file list.
ENTER	A new file name may be typed in and entered. Once entered, the file sequence is resorted. To save the new file name use the **CTRL Save** option.

Disk 44. Sorgan

Version: *Author*:
Requires: EA *Language*: AL *Updated*: 08/17/86

An assembly program that turns your keyboard into an organ. Sound effects, different instruments, various chord forms, color graphics, complete control of all sound parameters.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-44
Sectors total           = 360
Sectors used            = 139
Sectors available       = 219
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 SORGAN       33 PROGRAM      >022 032
002 >003 SORGAN/DOC   49 DIS/VAR 80   >042 048
003 >004 SORGAO        3 PROGRAM      >072 002
004 >005 SORGAP       54 PROGRAM      >074 053
```

Disk 44. Contents of file SORGAN/DOC

SORGAN II

Program By Jerome Trinkl

TI-99/4A

This is my first attempt at assembly language programming. I am sure it could be improved, so feel free to modify it in anyway. Be sure your children give it a try. To run this program with E/A cartridge use E/A option five (run program file). The program will execute automatically. A detailed description of all key options follows. This program is public domain so feel free to pass it along to others.

Jerome Trinkl

Sorgan II will allow you to use your computer keyboard to play voiced notes and chord formations. You have control from the keyboard to change some of the variables that affect those voices and chords.

The Sorgan screen has three main sections and was created in full color in the bit-map graphics mode. The first and most obvious section is the keyboard. It corresponds to a piano keyboard showing two octaves of keys (twenty-four keys). The twenty-fifth key on the extreme right is depressed only for special effects. The keys are accurate with that of a piano and not the computer keyboard. Uppercase letters on the computer play the notes and lowercase letters play the chord formations. When a note is played the piano key that corresponds on screen is shown depressed. Similarly when a chord is played the three notes making up the chord have their keys shown depressed. The key will stay depressed as long as the computer key is held down.

In the upper section the triangular mountain is the graphical representation of notes and chords played. Any one of the twelve bars generated are of random colors and correspond in height on the mountain to the frequency of the single note or the higher chord note played. Although two octaves may be played from the keyboard only one octave is displayed in the triangle. The second or higher octave is repeated using the same locations as the first. The color bars stay on screen as long as the note is played. The bars for chords stay on screen until they are deleted by playing a note of the same value or using the equal key to clear the graphics.

The lower section gives the status of all the key options except that of defaults and ♯ or ♭. Those key options are discussed in detail following this text.

Now is your chance to use one of those extra plastic strips for write-in key options that came with your computer when it was new. I have setup mine up like this:

```
-------------------------------------------------------------------
OCT | OCT | CHRD | CHRD | VOL | VOIC | CMOD | NMOD | # | DFAULT | SOM
-------------------------------------------------------------------
UP | DN | UP | DN | SELT | SELT | SELT | SELT | b | ON/OFF | CLR
-------------------------------------------------------------------
```

Note: This strip represents the top row of keys 1 through = only and not the function or control keys normally associated with the strip.

Detailed Description of Sorgan II Key Options

MAIN KEYS

KEYS	*FUNCTION*	*DESCRIPTION*
Z-M,A-L,Q-I	Notes	24 Keys or two octaves of chromatic notes
z-m,a-l,q-i	Chords	24 Keys or two octaves of chromatic chords *Note*: Release **ALPHA LOCK** key and use the **SHIFT** key to play notes with chords. Also the sustain volume should be greater than zero in order to hear both together.
1 & 2	Octave Up/Down	There are 6 oct. for notes. You play two octaves from the keyboard, the octave shown and the next higher octave. (6 plays 6 & 1)
3 & 4	Chord Formation Up/Down	Select formations: Major, Minor, Major7th, Minor7th, Augmented, and Diminished.
(5)	Volume Select	Set individual volumes for Notes, Chords, Sustain, and Bass. (use + and – keys too)
(6)	Voice Select	Select the voices of Organ, Piano, Space, Pulse, and Funny for Notes only.
(7)	Chord Mode Select	Provides different effects for chords. The Y factor will alter some of these effects.

(8)	Note Mode Select	Provides different effects for notes. The X factor will alter some of these effects.
(9)	♯ or ♭	Select sharps or flats. Note values will be consistent with music theory. This key gives an audible indication only.
(0)	Defaults On/Off	Certain default parameters were set up for all voice selections and certain chord mode selections. Gives an audible indication only. More detail following.
=	Clear/S.O.M.	Pressing this key briefly stops all sounds and clears the triangle graphics. Holding the key depressed will produce what I call sound of memory. (S.O.M.) You actually hear the machine code of this program. The X factor affects the time delay between each sound of S.O.M.

SORGAN STATUS SCREEN

```
              note
Chord-Major \  / Organ-Note
     /Sustn  A#  Norml\
Modes- 100   Cb   100 -Mode
     \Chord /  \ Notes/
Octave     chords      Bass
 \01/ Note - 15 -Volume Off
```

SPECIAL KEYS

KEYS	*FUNCTION*	*DESCRIPTION*
P	Special Effect	Special effect sound heard as long as key is pressed and is modified by both the X and Y factors. Graphics are a solid color randomly chosen in the triangle.
+ or **-**	Volume	Increases or decreases the volume you selected with the number 6 key(Vol select). Values range from 0 (off) to 15 (loudest)
;	Random X Value	Chooses a random value for X in the range of from ten to one thousand.
:	Random Y Value	Same as X.
,	X Decrease	Decreases X by one.
.	X Increase	Increases X by one.
>	Y Increase	Increases Y by one.
<	Y Decrease	Decreases Y by one.
SPACE BAR	Bass on/off	Once turned on bass stays on until = is pressed or space bar is pressed again.
FCTN BACK	Key Options	On screen review of these key options.
FCTN =	Quit	Same as most programs.

Note: Key options in () signify that one key is used to select from a list of options. The list is read from beginning to the end and then back to the beginning again.

What are these X and Y factors? They are simply a means by which you can control the various note and chord mode characteristics. If you are into programming they are a for-next delay loop for X or Y number of iterations. X affects the voices of Piano, Pulse, and Funny. It affects the note modes of Echo, Vibro, and Decay. Y factor affects the chord modes of Arpeg, Vibro, and Invrt. The idea is to experiment with different X and Y values for different sound effects. Use the random keys to get close to the number range you want for X and Y, then use the increase or decrease by one keys for an exact number.

X AND Y PARAMETERS

Note Echo Mode

> Plays a normal note three times after key is released and the delay between each note is governed by the X factor.

Note Vibro Mode

> Plays a distorted note with the X factor governing the amount of distortion.

Note Decay Mode

> As the key plays the note its volume decreases to zero as a function of how long the key is held depressed and the X factor.

Arpeggio Chord Mode

> Plays the individual notes of the chord from lowest to highest with a delay between each note of Y factor.

Inverted Chord Mode

> Same as an arpeggiated chord but from highest to lowest notes.

Special Effect (P Key)

> Both X and Y in combination affect this special sound.

DEFAULT PARAMETERS

Notes

Organ	X=100	Octave=1
Piano	X=600	Octave=3
Space	X=600	Octave=2
Pulse	X=500	Octave=1
Funny	X=100	Octave=2

Chords

Arpeggio and Inverted	Y=500
All Other Modes	Y=100

Note: The **0** key is used to turn on and off these defaults.

Disk 45. c99 Library

Version: 2.0 *Author*: Clint Pulley
Requires: EA *Language*: c99 *Updated*: 5/15/86

Includes a new graphics library which allows access to all 99/4A graphics capabilities, a text formatter with c99 source code similar to the TI-Writer formatter, speech routines, sound routines and string functions. Requires the c99 compiler on BCS #27.

```
dskdir. v2.0. 12-dec-96
Disk name               = C99REL4B
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README2      8 DIS/VAR 80   >022 007
002 >003 BITDOC       11 DIS/VAR 80   >029 010
003 >004 BITRTN       16 DIS/VAR 80   >033 015
004 >005 BITWRT        8 DIS/VAR 80   >042 007
005 >006 C99PFF        2 DIS/FIX 80   >049 001
006 >007 C99PFI        3 DIS/FIX 80   >04a 002
007 >008 CONV;C        5 DIS/VAR 80   >04c 004
008 >009 DIMTST;C      5 DIS/VAR 80   >050 004
009 >00a FCOPY;C       6 DIS/VAR 80   >054 005
010 >00b FLOATC       31 DIS/VAR 80   >059 030
011 >00c FLOATDOC     30 DIS/VAR 80   >077 029
012 >00d FLOATI        2 DIS/VAR 80   >094 001
013 >00e FMTIODOC      7 DIS/VAR 80   >095 006
014 >00f GRF1         14 DIS/FIX 80   >09b 013
015 >010 GRF1DOCS     19 DIS/VAR 80   >0a8 018
016 >011 GRF1RF        3 DIS/VAR 80   >0ba 002
017 >012 OPT;C        16 DIS/VAR 80   >0bc 015
018 >013 PRSET;C       9 DIS/VAR 80   >0cb 008
019 >014 RANDOM;C      4 DIS/VAR 80   >0d3 003
020 >015 RNDTST;C      4 DIS/VAR 80   >0d6 003
021 >016 RUNOFF1      31 PROGRAM      >0d9 030
022 >017 RUNOFF;C     45 DIS/VAR 80   >0f7 044
023 >018 RUNOFFDOC    16 DIS/VAR 80   >123 015
024 >019 SOUNDS;C      2 DIS/VAR 80   >132 001
025 >01a STRINGFNS    11 DIS/VAR 80   >133 010
026 >01b TCIOC        17 DIS/VAR 80   >13d 016
```

```
027 >01c TCIODOC      28 DIS/VAR 80   >14d 027
028 >01d TCIOI         5 DIS/VAR 80   >01e 004
```

Disk 45. Contents of file -README2

C99REL4B RELEASE NOTES 88/01/07

BY Clint Pulley

This disk contains programs and function libraries which should provide examples of useful programs in c99 and expand the capabilities of the language.

I am pleased to be able to include function libraries written by other c99 enthusiasts. I have not tested all of their contributions as I assume that they have done so.

Contents of this diskette:

From me:

C99PFF	
C99PFI	Obj files for pgm file generation (cf manual)
CONV;C	String <-> int conversion fcns
DIMTST;C	Test pgm for 2-dim arrays
FCOPY;C	Simple file copy program
FMTIODOC	Docs for printf/scanf et al
GRF1	Graphics function library
GRF1DOCS	Documentation for GRF1
GRF1RF	Include file for GRF1
OPT;C	Program to optimize output file from c99 compiler, producing a shorter, faster program. Only works if inline push code option was NOT selected.
PRSET;C	Printer setup program for an Epson compatible printer
RANDOM;C	Random number generators
RNDTST;C	Test program for RANDOM
RUNOFF1	Text formatter pgm file
RUNOFF;C	Source of RUNOFF1
RUNOFFDOC	Documentation for RUNOFF
SOUNDS;C	Beep and honk fcns

By Jay Holovacs:

BIT...	Bitmap graphics functions/docs

By Tom Bentley:

FLOAT...	Floating point functions/docs
TCIO...	Alternate file I/O library (supports all modes)

By Tom Wible and Don Mahler:

STRINGFNS Library of string functions

All doc files except BITDOC and FMTIODOC are set up for printing by RUNOFF1.

Clint

Disk 45. Contents of file BITDOC

BITMAP GRAPHICS COMMANDS FOR C99 COMPILER

12/18/85

This material is written by Jay Holovacs, 74756,413 (New Jersey Users Group) as a support for Clint Pulley's C compiler. I believe the commands are compatible with various versions of his compiler. I would appreciated any comments etc. on this material, hope you can use it. (Address any comments/questions via Email rather than forum messages as I get to check that more often).

The bitmap graphics extensions to c are divided into two "#include" files on this disk. The first, BITRTN, contains the tools for basic graphics manipulation. The second, BITWRT contains additional commands to mix text with bit mapped graphics.

Once included in your program, the commands can be called as any c function.

VALID RANGES:

color:	1 to 16
row:	1-24
column:	1-32
ascii:	32-127
x (bitmap location):	1-256
y (bitmap location):	1-192
str:	string pointer; string literal; base pointer of character array

Note: Internal labels in the assembly language portions all include '$' to avoid conflict with normally named global variables in your program.

INCLUDED IN THE BITRTN FILE:

`bitmap(fore,back)`
Must be called first to change the screen configuration from text mode (used by c) to bitmapped mode. The arguments are respectively the fore and background colors selected for the screen default.

`bitclr()`
Clears the entire screen, but does not change any color defaults. If you wish to modify screen color defaults, the bitmap() call can be repeated.

`plot(x,y,color)`
Will turn 'on' a single pixel at the screen location indicated.

`line(x1,y1,x2,y2,color)`
Draws a line of the specified color between the points indicated.

`rect(x1,y1,x2,y2,color)`
Draws a rectangle with opposite corners at the points indicated.

INCLUDED IN THE BITWRT FILE:

`bitxt()`
Call this function once, before calling bitmap(); this function copies all ASCII character definitions into a buffer in CPU ram.

`bputch(ascii,row,column,color)`
Places the character at location indicated

`bputs(row,column,color,str)`
Places text on the screen in 24x32 format (works similar to 'puts()' in normal c).

Disk 45. Contents of file BITRTN

```
/* BITR;CS CONTAINS FULL COMMENTS, USE BITRTN
 FOR INCLUDE PURPOSES */
/* REL VERSION 1.0 1/4/86
   Jay Holovacs  (New Jersey Users Group)
   95 King George Rd.
   Warren, NJ 07060
   CIS: 74756,413

   For Use With C99C Compiler by Clint Pulley */


/* Contents:

   BITMAP V1.1B
   LINE V2.1B
   PLOT V2.2A
   RECT V1.0  */

 bitmap(fore,back)
 /* V1.1B 12/15/85 JAY HOLOVACS
  MUST BE CALLED BEFORE ANY BITMAP COMMANDS
 ARE INVOKED
 * * USES TI BASIC COLOR DEFINITIONS */

  int fore,back;
  #asm
 REF VWTR,VSBW,VMBW,VSBR
 MOV @4(14),0
 DEC 0
 SLA 0,4
 A @2(14),0
 DEC 0
 SWPB 0
 BLWP @BITMP$
 B @B$MP2
BITMP$  DATA PLW$P,B$MP1 USES PLOT WORKSPACE
B$MP1  MOV *13,2
 LIMI 0
 LI 0,>0002
 BLWP @VWTR
 LI 0,>01E0
 MOV 0,@>83D4
 BLWP @VWTR
 LI 0,>0206
 BLWP @VWTR
 LI 0,>03FF
 BLWP @VWTR
 LI 0,>0403
```

```
 BLWP @VWTR
 LI 0,>0570
 BLWP @VWTR
 LI 0,>0607
 BLWP @VWTR
 LI 0,>3800
 LI 1,>D000
 BLWP @VSBW
 MOV 2,0
 SWPB 0
 ANDI 0,>000F
 ORI 0,>0700
 BLWP @VWTR
 LI 0,>1800
 CLR 1
SILP$  BLWP @VSBW
 INC 0
 AI 1,>0100
 CI 0,>1B00
 JNE SILP$
 BLWP @B$CL1
 LI 0,>2000
 MOV 2,1
CLP$  BLWP @VSBW
 INC 0
 CI 0,>3800
 JNE CLP$
 RTWP
B$CL1  DATA B$CL2,B$CL3
B$CL2  BSS 32
B$CL3  LIMI 0
 CLR 0
 CLR 1
CLP2$  BLWP @VSBW
 INC 0
 CI 0,>1800
 JNE CLP2$
 RTWP
B$MP2
 #endasm


bitclr()
/* V1.0B BY JAY HOLOVACS CLEARS THE SCREEN */

 #asm
 BLWP @B$CL1
 #endasm
```

```
line(x1,y1,x2,y2,color)
/* V2.1B 12/15/85 JAY HOLOVACS */
/* USE TI BASIC COLOR AND LOCATION FORMATS
  COLOR=1-16
  X LOC=1-256
  Y LOC=1-192 */

 int x1,y1,x2,y2,color;

 #asm
 MOV @2(14),4
 DEC 4
 MOV @4(14),3
 DEC 3
 MOV @6(14),2
 DEC 2
 MOV @8(14),1
 DEC 1
 MOV @10(14),0
 DEC 0
 BLWP @L$NE14
 B @L$NE15
L$NE14  DATA L$WSP,L$NE1
L$WSP  BSS 32
X1$  BSS 2
X2$  BSS 2
Y1$  BSS 2
Y2$  BSS 2
INCX$  BSS 2
INCY$  BSS 2
XP$T  BSS 2
YP$T  BSS 2
L$NE1  MOV 13,12
 MOV *12+,@X1$
 MOV *12+,@Y1$
 MOV *12+,@X2$
 MOV *12+,@Y2$
 MOV *12,0
 MOV @X1$,3
 MOV @X2$,5
 S 3,5
 MOV 5,11
 ANDI 11,>8000
 MOV @Y1$,6
 MOV @Y2$,8
 S 6,8
 MOV 8,10
 ABS 10
 MOV 5,9
 ABS 9
 C 9,10
```

```
 JLT YGRT$
 MOV 5,1
 ANDI 1,>8000
 JEQ L$NE5
 LI 1,>FF00
 JMP L$NE6
L$NE5  LI 1,>0100
L$NE6  MOV 1,@INCX$
 MOV 8,11
 ABS 8
 SLA 8,8
 CLR 7
 DIV 9,7
 MOV 11,11
 JGT L$NE12
 NEG 7
L$NE12  MOV 7,@INCY$
 JMP L$NE10
YGRT$  MOV 8,1
 ANDI 1,>8000
 JEQ L$NE7
 LI 1,>FF00
 JMP L$NE8
L$NE7  LI 1,>0100
L$NE8  MOV 1,@INCY$
 CLR 4
 MOV 5,11
 ABS 5
 SLA 5,8
 DIV 10,4
 MOV 11,11
 JGT L$NE11
 NEG 4
L$NE11  MOV 4,@INCX$
L$NE10  SLA 3,8
 MOV 3,@XP$T
 SLA 6,8
 MOV 6,@YP$T
L$NE9  MOV @XP$T,2
 SRL 2,8
 MOV @YP$T,1
 SRL 1,8
 C 2,@X2$
 JNE L$NE4
 MOV 9,9
 JNE L$NE13
 C 1,@Y2$
 JNE L$NE4
L$NE13  RTWP END
L$NE4  BLWP @PL$T3
```

```
 A @INCX$,@XP$T
 A @INCY$,@YP$T
 JMP L$NE9
L$NE15
 #endasm


plot(x,y,color)
 int x,y,color;
/* v 2.2A 12/27/85 */
/* USES TI BASIC COLORS AND LOCATIONS
   COLOR=1-16
   X LOCATION=1-256
   Y LOCATION=1-192

NOTE: USES WKS IN CPU PAD--IF YOU NEED THIS SPACE
 FOR OTHER PURPOSES, CHANGE:
PLW$P EQU >8330
 to:
PLW$P BSS 32 */

 #asm
 MOV @2(14),0
 DEC 0
 MOV @4(14),1
 DEC 1
 MOV @6(14),2
 DEC 2
 BLWP @PL$T3
 B @PL$T4
PL$T3  DATA PLW$P,PL$T1 PLOT ENTRY POINT
PLW$P  EQU >8330
PL$T1  MOV 13,12
 LIMI 0
 MOV *12+,10
 SLA 10,12
 MOV *12+,2
 MOV *12,1
 CI 1,255
 JH PL$T2
 CI 2,192
 JH PL$T2
 MOV 1,8
 MOV 2,6
 SRA 1,3
 SRA 2,3
 MOV 1,4
 SLA 4,3
 S 8,4
 NEG 4
 INC 4
```

```
 MOV 2,5
 SLA 5,3
 S 6,5
 NEG 5
 MOV 2,3
 SLA 3,5
 A 1,3
 SLA 3,3
 A 5,3
 MOV 4,0
 LI 7,>0100
 SRA 7,0
 MOV 3,0
 BLWP @VSBR
 SWPB 7
 SOCB 7,1
 BLWP @VSBW
 AI 0,>2000
 BLWP @VSBR
 ANDI 1,>0F00
 SOC 10,1
 BLWP @VSBW
PL$T2  RTWP
PL$T4
 #endasm

rect(x1,y1,x2,y2,color)
/* v1.0 12/16/85 */
 int x1,y1,x2,y2,color;
{
 line(x1,y1,x1,y2,color);
 line(x1,y2,x2,y2,color);
 line(x2,y2,x2,y1,color);
 line(x2,y1,x1,y1,color);
}
```

Disk 45. Contents of file BITWRT

```
/* REL 1.0 1/4/86 ADDS TEXT CAPABILITY
CONTENTS:
  bitxt 1.1A
  bputch 2.1A
  bputs 1.0
  btblnk 1.0




TO BITRTN
  By:  Jay Holovacs (New Jersey Users Group)
       95 King George Rd
       Warren, NJ 07060
       CIS 74756,413

Required files:

       C99C compiler by Clint Pulley
       BITRTN support                        */


bitxt()
/* v1.1A 12/8/85 must be called before
bitmap()
copies letter definitions into
storage buffer for use in bitmap
mode */

 #asm
 REF VMBR,VMBW
 B @T$T2
T$T  BSS 768
T$T1  BSS 8
T$T2  LI 0,>900
 LI 1,T$T
 LI 2,>300
 BLWP @VMBR
 #endasm


bputch(ascii,row,colmn,color)
 int ascii,row,colmn,color;


/* v2.1A 12/8/85 or's char def onto
screen image ROW=1-24, COL=1-32 */
/* COLORS TIBASIC FORMAT */
```

```
 #asm
 MOV @6(14),0
 DEC 0
 SLA 0,5
 A @4(14),0
 DEC 0
 SLA 0,3
 LI 1,T$T1
 LI 2,8
 BLWP @VMBR
 MOV @8(14),3
 AI 3,-32
 SLA 3,3
 AI 3,T$T
 LI 5,T$T1
 LI 4,6
B$LP  SOC *3+,*5+
 DECT 4
 JOC B$LP
 BLWP @VMBW
 MOV @2(14),6
 DEC 6
 MOV 6,7
 SLA 6,12
 SLA 7,4
 SOC 7,6
 AI 0,>2000
 BLWP @VMBR
 LI 5,T$T1
 LI 4,6
 LI 3,>F0F0
B$LP2  SZC 3,*5
 SOC 6,*5+
 DECT 4
 JOC B$LP2
 BLWP @VMBW
 #endasm

bputs(row,colmn,color,str)
 int row,colmn,color;
 char *str;
/* v1.0 */

 {
 int ascii;
 while (*str!=0)
   {
    ascii=*str++;
    bputch(ascii,row,colmn++,color);
   }
```

```
 }


 btblnk(row,colmn)
 /* v1.0 12/11/85 blanks a character space */

 int row,colmn;

 #asm
 B @BL$1
B$BF  DATA 0,0,0,0
BL$1  MOV @4(14),0
 DEC 0
 SLA 0,5
 A @2(14),0
 DEC 0
 SLA 0,3
 LI 1,B$BF
 LI 2,8
 BLWP @VMBW
 #endasm
```

Disk 45. Contents of file CONV;C

```
/* simple conversion functions */
/*
** n=atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
/*
** s=itod(nbr,str,sz) -
**
** convert nbr to signed decimal string of width sz
** right justified, blank filled, result in str[].
** sz includes 0-byte string terminator. returns str.
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ int sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
}
```

Disk 45. Contents of file DIMTST;C

```
/* Test program for two dimsional arrays
*/
extern printf();
char st[5][10]={"Line # 01",
                "Line # 02",
                "Line # 03",
                "Line # 04",
                "Line # 05"};
char ss[5][40];
int a[4][3]={1,2,3, 4,5,6, 7,8,9, 10,11,12};
int i,j;

main()
{ printf("Test of two dimensional arrays\n\n");
  pmat(a);
  printf("\n");
  for(i=0;i<4;++i)
    for(j=0;j<3;++j)a[i][j]=10*(i+1)+(j+1);
  pmat(a);
  printf("\n");
  getchar();
  for(i=0;i<5;++i)
    printf("%s\n",&st[i][0]);
  printf("\nInput 5 lines\n");
  for(i=0;i<5;++i)
    gets(&ss[i][0]);
  printf("\nYou entered:\n");
  for(i=0;i<5;++i)
    printf("%s\n",&ss[i][0]);
  printf("\nThe 3rd line was\n%s\n",&ss[2][0]);
}
/* print a 4x3 matrix neatly
*/
pmat(m) int m[][3];
{ int n;
  for(n=0;n<4;++n)
    printf(" %6d%6d%6d\n",m[n][0],m[n][1],m[n][2]);
}
```

Disk 45. Contents of file FCOPY;C

```
/* c99 file copy utility
**
** this utility program copies a display/variable 80
** file. Any legal filename may be used.
** responding to a filename prompt by pressing ENTER
** will result in that file being mapped to the
** console. keyboard input is terminated with CTRL-Z.
** during execution, pressing a key will pause the
** program, CLEAR (FCTN-4) will abort.
** Load with CFIO
*/
#include DSK1.STDIO
#define LEN 81
char fn[40],line[LEN];
int c;
FILE inp,out;
main()
{ while(1)
  { putchar(FF);
    puts("c99 file copy utility\n\n");
    inp=getfn("Input","r");
    out=getfn("Output","w");
    putchar('\n');
    while(fgets(line,LEN,inp))
    { fputs(line,out);
      if(!out)putchar(EOL);
      poll(YES);
    }
    fclose(inp);
    fclose(out);
    puts("\nmore copies?");
    c=getchar();
    if((c=='N')|(c=='n'))break;
  }
  exit(7);
}
/* getfn returns unit # or 0 if null name entered
*/
getfn(text,m) char *text,*m;
{ int unit;
  unit=0;
  while(1)
  { puts(text);
    puts(" filename?");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
```

```
  }
  return(unit);
}
```

Disk 45. Contents of file FLOATC

```
/*
 * FLOATING POINT ROUTINES  V1.1
 *
 * Written by :
 *
 *  Tom Bentley
 *  P.O. Box 346
 *  Osgoode, Ontario
 *  Canada  K0A 2W0
 *
 *  Date Changed: May 6, 1986
 *
 */

#asm
 REF C$GPLL,XMLLNK,VMBW
 DEF ITOF,FTOI,STOF,FTOS,FEXP,FCOM,FPGET,FPPUT,FINT,FCPY

FAC    EQU >834A
ARG    EQU >835C
STATUS EQU >837C
VSPTR  EQU >836E
VDPWRK EQU >24CA
EQ     DATA >2000
GT     DATA >4000
GTEQ   DATA >6000
ONE    DATA 1
SAVREG BSS 12
#endasm

#define FLOATLEN 8
#define float    char


float ftemp[FLOATLEN];
int   itemp;

fpget(s,f)
  char  *s;
  float *f;
  {
    gets(s);
    return(stof(s,f));
  }

fpput(f,s)
  float *f;
  char  *s;
```

```
  {
    puts(ftos(f,s,0,0,0));
  }

/*
 * name: itof - integer to floating point
 *
 * itof(i,f)
 *
 * i = integer value
 * f = float pointer
 * returns pointer to float number
 *
 */

itof(i,f)
  int i;
  float *f;
  {

#asm
       MOV  @4(14),@FAC  MOVE INTEGER TO FAC
       BLWP @XMLLNK
       DATA >2300
       LI   2,8
       MOV  @2(14),0     GET FLOAT POINTER
       LI   1,FAC
ITOFL  MOVB *1+,*0+
       DEC  2
       JNE  ITOFL
#endasm
    return(f);
    }

/*
 * ftoi - floating point to integer
 *
 * i = ftoi(f)
 *
 * i = integer value
 * f = float pointer
 *
 */

ftoi(f)
  float *f;
  {
#asm
       LI   2,8
       MOV  @2(14),0     GET FLOAT POINTER
       LI   1,FAC
```

```
FTOIL  MOVB *0+,*1+
       DEC  2
       JNE  FTOIL
       BLWP @XMLLNK
       DATA >1200
       MOV  @FAC,@ITEMP
#endasm
    return(itemp);
    }

/*
 * name: stof - string to floating point
 *
 * stof(s,f)
 *
 * s = string pointer
 * f = float pointer
 * returns pointer to float number
 *
 */

stof(s,f)
  char *s;
  float *f;
  {
#asm
       LI   0,VDPWRK     MOVE STRING NUMBER TO VDP
       MOV  @4(14),1
       CLR  2            USED FOR COUNTER
STOFL2 INC  2
       CB   *1+,2        IS IT A NULL?
       JEQ  STOFL1
       CI   2,21         UP TO 20 DIGITS
       JLT  STOFL2
       DEC  1
       MOVB 2,*1         NULL TERMINATE
STOFL1 MOV  @4(14),1
       BLWP @VMBW        WRITE TO VDP
       MOV  0,@FAC+12    ADDRESS OF STRING IN VDP
       BLWP @XMLLNK
       DATA >1000
       MOV  @2(14),0
       LI   1,FAC
       LI   2,8
STOFL  MOVB *1+,*0+
       DEC  2
       JNE  STOFL
#endasm
    return(f);
    }
```

```
/*
 * name: ftos - floating point to string
 *
 * ftos(f,s,mode,signif,decimal)
 *
 * s = string pointer
 * f = float pointer
 * mode = 0 - basic mode, 1 - fix mode
 *  if fixed mode then the following must
 *  be specified.
 *
 * signif  = number of significant digits
 * decimal = indicates the number of digits to
 *           the right of the decimal point
 *
 * returns pointer to string
 *
 */

ftos(f,s,mode,signif,decimal)
  float *f;
  char  *s;
  int   mode,signif,decimal;
  {
#asm
       MOV  @6(14),0     MODE
       SWPB 0
       MOVB 0,@FAC+11
       MOV  @4(14),0     SIGNIF
       SWPB 0
       MOVB 0,@FAC+12
       MOV  @2(14),0     DECIMAL
       SWPB 0
       MOVB 0,@FAC+13
       LI   2,8
       MOV  @10(14),0
       LI   1,FAC
FTOSL  MOVB *0+,*1+
       DEC  2
       JNE  FTOSL
       LI   3,>8320      SAVE C WORKSPACE
       LI   4,SAVREG
       LI   5,6
X1     MOV  *3+,*4+
       DEC  5
       JNE  X1
       CLR  @STATUS
       BL   @C$GPLL
       DATA >0014
       CLR  0
       MOVB @FAC+11,0
```

```
       SWPB 0
       AI   0,>8300
       CLR  2
       MOVB @FAC+12,2
       SWPB 2
       LI   3,SAVREG     RESTORE C WORKSPACE
       LI   4,>8320
       LI   5,6
X2     MOV  *3+,*4+
       DEC  5
       JNE  X2
       MOV  @8(14),1
FTOSL1 MOVB *0+,*1+
       DEC  2
       JNE  FTOSL1
       CLR  0
       MOVB 0,*1         TERMINATE STRING WITH NULL
#endasm
      return(s);
      }

/*
 * name: fexp - execute an expression
 *
 *  fexp(f1,op,f2,res)
 *
 *  f1 = floating point number
 *  f2 = second floating point number
 *  op = op code (*,/,+,/)
 *  res = result in floating point format
 *  returns pointer to float result
 *
 */

fexp(f1,op,f2,res)
  float *f1,*f2,*res;
  char  *op;
  {
#asm
       MOV  @6(14),0     GET OP CODE
       CLR  3
       MOVB *0,3
       LI   2,8          MOVE F1 TO ARG
       MOV  @8(14),0
       LI   1,ARG
FEXPL1 MOVB *0+,*1+
       DEC  2
       JNE  FEXPL1
       LI   2,8          MOVE F2 TO FAC
       MOV  @4(14),0
```

```
       LI   1,FAC
FEXPL2 MOVB *0+,*1+
       DEC  2
       JNE  FEXPL2
FADD   CI   3,>2B00      ADD OP CODE
       JNE  FSUB
       BLWP @XMLLNK
       DATA >0600
       JMP  FRES
FSUB   CI   3,>2D00      SUBTRACT OP CODE
       JNE  FMULT
       BLWP @XMLLNK
       DATA >0700
       JMP  FRES
FMULT  CI   3,>2A00      MULTIPLY OP CODE
       JNE  FDIV
       BLWP @XMLLNK
       DATA >0800
       JMP  FRES
FDIV   CI   3,>2F00      DIVIDE OP CODE
       JNE  FRES
       BLWP @XMLLNK
       DATA >0900
FRES   MOV  @2(14),0
       LI   1,FAC
       LI   2,8
FEXPL3 MOVB *1+,*0+
       DEC  2
       JNE  FEXPL3
#endasm
    return(res);
  }

/*
 * name: fint - greatest integer function
 *
 * fint(f1,f2)
 *
 *  f1 = floating point value
 *  f2 = int floating point value
 *  return address of f
 *
 */

fint(f1,f2)
  float *f1,*f2;
  {
#asm
       MOV  @4(14),0
       LI   1,FAC
       LI   2,8
```

```
FINTL1 MOVB *0+,*1+
       DEC  2
       JNE  FINTL1
       LI   3,>8320      SAVE C WORKSPACE
       LI   4,SAVREG
       LI   5,6
X3     MOV  *3+,*4+
       DEC  5
       JNE  X3
       CLR  @STATUS
       BL   @C$GPLL
       DATA >0022
       LI   3,SAVREG     RESTORE C WORKSPACE
       LI   4,>8320
       LI   5,6
X4     MOV  *3+,*4+
       DEC  5
       JNE  X4
       MOV  @2(14),0
       LI   1,FAC
       LI   2,8
FINTL2 MOVB *1+,*0+
       DEC  2
       JNE  FINTL2
#endasm
    return(f2);
  }

/*
 * name: fcom - logical compare
 *
 * fcom(f1,rel,f2)
 *
 * f1 = floating point value
 * f2 = floating point value
 * rel = relation (==,!=,<,<=,>,>=)
 *
 * returns 1 for true 0 for false
 *
 */

fcom(f1,rel,f2)
  float *f1,*f2;
  char  *rel;
  {
#asm
       MOV  @4(14),3     GET REL CODE
       MOV  @6(14),0     MOVE F1 TO ARG
       LI   1,ARG
       LI   2,8
```

```
FCOML1 MOVB *0+,*1+
       DEC  2
       JNE  FCOML1
       MOV  @2(14),0     MOVE F2 TO FAC
       LI   1,FAC
       LI   2,8
FCOML2 MOVB *0+,*1+
       DEC  2
       JNE  FCOML2
       CLR  @ITEMP
       BLWP @XMLLNK
       DATA >0A00
       CLR  5
       MOVB @STATUS,5
       CLR  4
       MOVB *3+,4
       SWPB 4
       MOVB *3,4
       SWPB 4
EQUAL  CI   4,>3D3D      ==
       JNE  NEQUAL
       COC  @EQ,5
       JNE  FCOMR1
       JMP  FCOMR9
NEQUAL CI   4,>213D      !=
       JNE  LTHAN
       COC  @EQ,5
       JEQ  FCOMR1
       JMP  FCOMR9
LTHAN  CI   4,>3C00      <
       JNE  GTHAN
       CZC  @GTEQ,5
       JNE  FCOMR1
       JMP  FCOMR9
GTHAN  CI   4,>3E00      >
       JNE  LTHNEQ
       COC  @GT,5
       JNE  FCOMR1
       JMP  FCOMR9
LTHNEQ CI   4,>3C3D      <=
       JNE  GTHNEQ
       COC  @GT,5
       JEQ  FCOMR1
       JMP  FCOMR9
GTHNEQ CI   4,>3E3D      >=
       JNE  FCOMR1
       COC  @GT,5
       JEQ  FCOMR9
       COC  @EQ,5
       JNE  FCOMR1
FCOMR9 MOV  @ONE,@ITEMP
```

```
FCOMR1 NOP
#endasm
    return(itemp);
  }

fcpy(f1,f2)
  float *f1,*f2;
  {
#asm
       MOV  @4(14),0
       MOV  @2(14),1
       LI   2,8
FCPYL  MOVB *0+,*1+
       DEC  2
       JNE  FCPYL
#endasm
    return(f2);
  }
```

Disk 45. Contents of file FLOATDOC

'C' Floating Point Library (by Tom Bentley 860201)

Tom Bentley
P.O. Box 346
Osgoode, Ont.
Canada K0A 2W0

The following is a description of the floating point library that may be used by Clint Pulley's c99 program.

Library and Include Files

a) FLOAT — The floating point library. This contains all of the floating point routines.

b) FLOATI — Include file containing the externs and defines required by the floating point library.

How to define a FLOAT data type:

To define a float data type use 'float' as the type and FLOATLEN as the size of the data type. A float number on the TI is always stored as an 8 byte number, keep this in mind if you wish to manipulate it with your own routines.

Example.

```
/* define float type */
float number[FLOATLEN];
```

Functions in FLOAT:

- Prompt for floating point number

```
float f[FLOATLEN];
char  *c, s[input size];

c=fpget(s,f);
```

Prompts for a floating point string and converts is to a floating point number. s is a character array which receives the string, f is the float array which receives the floating point number and c is a character pointer that contains the pointer to the float array.

Example 1.

```
    /* prompt for a float number */

    puts("Enter a number: ");
    fpget(s,f);
```

- Display floating point number to screen

```
float f[FLOATLEN];
char  s[display size];

fpput(f,s);
```

Display at floating point number to the screen. Locate(r,c) may be used to position to the start of where to display from. f is the float array that contains the floating point number and s is the char array that contains the string representation of the floating point number. s is available to you even after the call to fpput so you may use it for other purposes.

Example 1.

```
    /* display floating point number */

    puts("The number is: ");
    fpput(f,s);
```

- Integer to floating point

```
int   i;
float f[FLOATLEN];
char  *c;

c=itof(i,f);
```

Converts an integer value to a floating point number. i is any integer value, f is the float array which receives the floating point number and c is a character pointer that contains the pointer to the float array.

Example 1.

```
    /* convert integer to float */

    i = 100;
    itof(i,f);
    puts("i contains ");
    fpput(f,s);
```

- Floating point to integer

```
float f[FLOATLEN];
int   i;

i=ftoi(f);
```

Converts a floating point number to integer. f is the floating point array to be converted and i is the integer. The floating point array must contain a valid integer number in the range of +32767 and -32768.

Example 1.

```
    /* convert float to integer */

    i=ftoi(f);
    puts("i contains ");
    itod(i,s,4);
    puts(s);
```

- String to floating point

```
float f[FLOATLEN];
char  *c, s[string size];

c=stof(s,f);
```

Converts a numeric string to a floating point number. s is a character array that contains the number to be converted, f is the float array and c is a character pointer that contains the pointer to the float array.

Example 1.

```
    /* convert string to float */

    puts("Enter a number: ");
    gets(s);
    stof(s,f);
```

- Floating point to string

```
float f[FLOATLEN];
char  *c, s[string size];
int   mode, signif, decimal;

c=ftos(f,s,mode,signif,decimal);
```

Converts a float array to a string array. f is a float array that contains the float number to be converted, s is a character array that contains the converted number, mode is a value that specifies the conversion mode (0 = basic mode, displayed as in basic, 1 = fix mode, if fixed mode then signif and decimal must be specified.), signif contains the number of significant digits, decimal indicates the number of digits to the right of the decimal point and c is a character pointer that contains the pointer to the converted string.

Example 1.

```
    /* convert float to string */

    /* use basic mode */
    ftos(f,s,0,0,0);
    puts("Value is ");
    puts(s);

    /* use fix mode, scientific notation */
    ftos(f,s,1,5,2);
    puts("\nValue is ");
    puts(s);

    /* if f = 2355 then s = 2.355E+03    */
    /* if f =   25 then s = 2.5E+01      */
    /* if f = 1234.56 then s = 1.2346E+03*/
```

- Execute float expression

```
float f1[FLOATLEN],f2[FLOATLEN],res[FLOATLEN];
char  *c, op[2];

c=fexp(f1,op,f2,res);
```

Executes and expression between f1 and f2 using the provided op code. f1 is a float array containing the first operand, f2 is a float array containing the second operand, op is a character array containing the op code (one of *,/,+,-), res is a float array that contains the result of the expression and c is a character pointer that points the result.

Example 1.

```
    /* perform expression */

    /* add two numbers */
    fexp(f1,"+",f2,res);

    /* subtract two numbers */
    fexp(f1,"-",f2,res);

    /* multiply two numbers */
    fexp(f1,"*",f2,res);

    /* divide two numbers */
    fexp(f1,"/",f2,res);
```

- Compare float numbers

```
float f1[FLOATLEN], f2[FLOATLEN];
char  rel[3];
int   true;

true=fcom(f1,rel,f2);
```

Compares two floating point numbers using the relation supplied (one of ==,!=,<,<=,>,>=). f1 is a float array containing first number, f2 is a float array containing second number and rel is a character array containing the compare relation.

Example 1.

```
    /* compare two float numbers */

    /* are they equal (==) ? */
    if(fcom(f1,"==",f2))
      /* true */

    /* are they not equal (!=) ? */
    if(fcom(f1,"!=",f2))
      /* true */

    /* is f1 < f2 ? */
    if(fcom(f1,"<",f2))
      /* true */

    /* is f1 <= f2 ? */
    if(fcom(f1,"<=",f2))
      /* true */

    /* is f1 > f2 ? */
    if(fcom(f1,">",f2))
```

```
      /* true */

    /* is f1 >= f2 ? */
    if(fcom(f1,">=",f2))
      /* true */
```

- Float greatest integer function

```
float f1[FLOATLEN], f2[FLOATLEN];
char  *c;

c=fint(f1,f2);
```

Takes a float value and returns it's greatest integer value. f1 is a float array containing a float value, f2 is a float array that will contain the greatest integer value and c is a character pointer that points to f2.

Example 1.

```
    /* take the greatest integer value */

    f= some float value
    fint(f,f);
```

- Copy float number

```
float f1[FLOATLEN], f2[FLOATLEN];
char  *c;

c=fcpy(f1,f2);
```

Copy one float array to another float array, c is a character pointer to f2.

Example 1.

```
    /* copy one float to another float */

    fcpy(f1,f2);
```

Disk 45. Contents of file FLOATI

```
/* C Float Include file */

extern itof(),ftoi(),stof(),ftos(),fexp();
extern fcom(),fpget(),fpput(),fint(),fcpy();

#define float    char /* float data type */
#define FLOATLEN 8    /* size of float data type */
```

Disk 45. Contents of file FMTIODOC

Formatted Input/Output functions (86/10/28)

This c99 release includes the standard C formatted I/O functions in object form. printf, fprintf, sprintf, scanf, fscanf, and sscanf perform as in standard C. Supported conversion modes are:

c	single character
d	signed decimal
o	octal
s	character string
u	unsigned decimal
x	hexadecimal

An extern specifier similar to:

```
extern printf(),scanf();
```

must be provided for each function used.

The object file requirements for these functions are:

for printf	load PRINTF
for fprintf	load FPRINTF,PRINTF,CFIO
for sprintf	load SPRINTF,PRINTF
for scanf	load SCANF
for fscanf	load FSCANF,SCANF,CFIO
for sscanf	load SSCANF,SCANF

In order to provide in-line editing for keyboard input, scanf was implemented using gets. As a result, no conversions occur until the input line is terminated with the **ENTER** key. scanf has an internal char [81] line buffer which is only reset on the first call to scanf after a load. If more data items are entered than are scanned, the remainder of the input line is available to the next scanf. If the program is rerun (without reloading), this will also be the case.

Disk 45. Contents of file GRF1DOCS

GRF1 : c99 graphics mode 1 function library

Usage Documentation (86/07/27)

This library provides functions analogous to those in Extended Basic for character graphics and sprite control. Except as noted, all functions perform the same operation as the equivalent Ex Basic call although names may differ.

All arguments are of type int except for the character definition string in chrdef. Arguments preceded by & are used to return values. Omission of & (address of) will result in highly unpredictable results!

Numbering conventions:

Character positions:	row 1-24, column 1-32
Colors:	1-16
Characters:	0-255 (full ASCII set available)
Character sets	: 0-31 (add 3 to Ext. Basic set number)
Sprites:	0-31 (32 sprites available)
Sprite characters:	0-255 (coincide with normal characters)
Sprite positions:	row 0-255 (invisible above 191)
	col 0-255

Functions available in GRF1 :

`grf1();`
Set to graphics 1 mode (24x32 characters). Load standard character patterns. Character colors set to black/transparent, backdrop set to cyan.

`text();`
Set to text mode (24x40 characters, no sprites). Load standard character patterns. Screen set to black/cyan.

`screen(c);`
Set screen (backdrop) color to c.

`color(cs,f,b);`
Change colors for char set cs to f/b.

`chrdef(ch,str);`
Define character pattern(s) starting at character ch. Up to 4 characters may be defined with one call. str is either a quoted string or a pointer to a string consisting of 1-64 hex (0-9, A-F) digits. Incomplete patterns are zero-filled.

`chrset();`
Load standard character patterns.

`patcpy(a,b);`
Copy the pattern for character a to the pattern space for character b. This function is provided instead of Ext. Basic's CHARPAT.

`clear();`
Clear the screen.

`hchar(r,c,ch,n);`
Place character ch at row r, col c and repeat n times horizontally.

`vchar(r,c,ch,n);`
Place character ch at row r, col c and repeat n times vertically.

`c=gchar(r,c);`
Return the value (int) of the character at row r, col c.

`s=joyst(u,&x,&y);`
Read joystick u (1 or 2) and return the x and y values (0 , +4, or -4). s is true if x or y != 0.

`c=key(u,&s);`
Read keyboard u (0-5) and return char value c and status s (-1, 0, or 1).

`sprite(spn,ch,col,dr,dc);`
Define sprite number spn with char ch, color col, located at dr,dc.

`spdel(spn);`
Delete sprite spn.

`spdall();`
Delete all sprites and clear automotion table.

`spcolr(spn,col);`
Set sprite spn to color col.

`sppat(spn,ch);`
Set pattern for sprite spn to char ch.

`sploct(spn,dr,dc);`
Locate sprite spn at row dr, col dc.

`spmag(f);`
Set sprite magnification to f (1-4).

`spmotn(spn,rv,cv);`
Set row velocity rv and col velocity cv for sprite spn.

`spmct(n);`
Enable automotion for the first n sprites (numbers 0 to n-1).

`spposn(spn,&rp,&cp);`
Return row position rp and col position cp for sprite spn.

`dsq=spdist(spn1,spn2);`
Return the square of the distance between sprites spn1 and spn2.

`dsq=spdrc(spn,dr,dc);`
Return the square of the distance between sprite spn and location dr,dc.

`flg=spcnc(spn1,spn2,tol);`
Returns true if the distance between sprites spn1 and spn2 is <= tol.

`flg=spcrc(spn,dr,dc,tol);`
Returns true if the distance between sprite spn and location dr,dc is <= tol.

`flg=spcall();`
Returns true if any sprites are in coincidence.

Notes :

For sprite automotion to occur, interrupts must be enabled at least 60 times per second. The functions poll, key, and joyst enable interrupts when called. If they are not called often enough, a function such as the following may be used :

```
inton()
{
#asm
 LIMI 2
 LIMI 0
#endasm
}
```

inton should be invoked at a suitable place in the program to ensure smooth sprite motion.

It is necessary to include REFs for each function referenced. An include file (GRF1RF) containing REFs for all functions in this library is on this diskette. It is accessed with:

```
#include "dsk1.grf1rf"
```

The following functions may also be used in text mode :

```
chrdef  chrset  patcpy  hchar vchar gchar joyst key
```

Disk 45. Contents of file GRF1RF

```
/* grf1 references
*/
extern grf1(),text(),screen(),color(),chrdef(),chrset(),patcpy(),clear();
extern hchar(),vchar(),gchar(),joyst(),key(),sprite(),spdel(),spdall();
extern spcolr(),appat(),sploct(),spmag(),spmotn(),spmct(),spposn();
extern spdist(),spdrc(),spcnc(),spcrc(),spcall();
```

Disk 45. Contents of file OPT;C

```
/* c99 code optimizer v1.3 by Clint Pulley
**
** Last Edit 88/01/09 0820
**
** Load with CSUP, CFIO
*/
#include dsk1.stdio
#define MAXLIN 81
#define LINES  10
char lbuf[810];
char *line1,*line2,*line3,*line4,*line5;
char *pat1,cn[6];
int inp,otf,ni,nb;
int out,in,rptr,eoflg;
/*
*/
main()
{ puts("c99 optimizer v1.2\n\n");
  pat1=" MOV *8,8";
  ni=nb=in=eoflg=0;
  out=-LINES;
  while(1)
  { inp=getfn("Input","r");
    if(inp)break;
    puts("Illegal filename\n");
  }
  roll(LINES);
  otf=getfn("Output","w");
  fputs("*c99opt v1.2",otf);
/* main loop */
  while(read(&line1))
  { poll(1);
    if(match(line1," MOV 14,8"))
    { if(type1()) continue;
    }
    if(match(line1," BL 15"))
    { if(type2()) continue;
    }
    fputs(line1,otf);
    roll(1);
  }
  fclose(inp);
  fclose(otf);
  puts("\nInstructions optimized =");
  puts(itod(ni,cn,6));
  puts("\nBytes of memory saved  =");
  puts(itod(nb,cn,6));
}
```

```
/* get filename and open file
*/
getfn(text,m) char *text,*m;
{ int unit;
  char fn[20];
  unit=0;
  while(1)
  { puts(text);
    puts(" filename? ");
    gets(fn);
    if(!*fn)break; /* return 0 if null name */
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
type1()
{ read(&line2);
  if(match(line2,pat1))
  { fputs(" MOV *14,8",otf);
    ++ni;
    nb=nb+2;
    roll(2);
    return 1;
  }
  if(match(line2," INCT 8"))
  { return(type1a());
  }
  if(nmatch(line2," AI 8,",6))
  { return(type1b());
  }
  reset();
  return 0;
}
type1a()
{ read(&line3);
  if(match(line3,pat1))
  { fputs(" MOV @2(14),8",otf);
    ++ni;
    nb=nb+2;
    roll(3);
    return 1;
  }
  reset();
  return 0;
}
type1b()
{ read(&line3);
  if(match(line3,pat1))
  { strcpy(line1," MOV @");
    strcat(line1,line2+6);
```

```
    strcat(line1,"(14),8");
    fputs(line1,otf);
    ++ni;
    nb=nb+4;
    roll(3);
    return 1;
  }
  reset();
  return 0;
}
type2()
{ read(&line2);
  if(nmatch(line2," LI 8,",6) | match(line2," S 8,8"))
  { read(&line3);
    if(match(line3," MOV *14+,9"))
    { fputs(" MOV 8,9",otf);
      fputs(line2,otf);
      ++ni;
      nb=nb+2;
      roll(3);
      return 1;
    }
  }
  reset();
  return 0;
}
/* match entire string
*/
match(s,t) char *s,*t;
{ while(*s==*t)
  { if(!*s)return(1);
    ++s; ++t;
  }
  return(0);
}
/* match partial strings
*/
nmatch(s,t,n) char *s,*t; int n;
{ while(n--)
  { if(*s++==*t++)continue;
    return(0);
  }
  return(1);
}
/* concatenate t to end of s
*/
strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
```

```
}
/* copy t to s
*/
strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
/*
** itod(nbr,str,sz) -
**   convert nbr to signed decimal string of width sz
**   right justified, blank filled, result in str[].
**   sz includes 0-byte string terminator.
**   returns str.
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
}
/*
** circular lookahead buffer handling
*/
/* roll n lines in circular buffer
*/
roll(n) int n;
{ char *cptr;
  rptr=out=(out+n)%LINES;
  if(eoflg)return;
  while(n--)
  { cptr=bufloc(in);
    if(!fgets(cptr,MAXLIN,inp))
    { *cptr=127;
      return;
    }
    in=++in%LINES;
  }
}
/* read a line by setting ptr to the buffer loc
** return 0 if end-of-file
*/
read(ptr) int *ptr;
{ char *cptr;
  *ptr=cptr=bufloc(rptr);
  rptr=++rptr%LINES;
```

```
  if(*cptr==127)return 0;
  return 1;
}
/* reset rptr to starting line
*/
reset()
{ rptr=out;
}
/* return buffer location for line n
*/
bufloc(n) int n;
{  return(lbuf+n*MAXLIN);
}
```

Disk 45. Contents of file PRSET;C

```
/* PRSET - program to set print modes for
**         Epson compatible printers
** Load with CFIO
*/
#include "dsk1.stdio"

int pr,sel;

main()
{ pr=fopen("PIO.LF","w");
  putml("Set Printer Modes :");
  putchar('\n');
  putml("A  Reset Printer");
  putml("B  Pica  (10 char/inch)");
  putml("C  Elite (12 char/inch)");
  putml("D  Italic");
  putml("E  Non-italic");
  putml("F  Emphasized");
  putml("G  Double-strike");
  putml("H  Enlongated");
  putml("I  Compressed");
  putml("J  Subscript");
  putml("K  Superscript");
  putml("L  Normal");
  putml("M  6 lines/inch");
  putml("N  8 lines/inch");
  putml("O  Indent 8 spaces");
  putml("P  Double space");
  putml("Q  Skip perforations");
  putml("R  Disable paper-out check");
  putml("S  Print tiny");
  putml("X  Exit program");
  locate(24,2);
  puts("Selection? ");

  do
  {
    switch( sel=getchar()&95 )
    {
  case 'A' : putfmt("\033@");   break;
  case 'B' : putfmt("\033P");   break;
  case 'C' : putfmt("\033M");   break;
  case 'D' : putfmt("\0334");   break;
  case 'E' : putfmt("\0335");   break;
  case 'F' : putfmt("\033E");   break;
  case 'G' : putfmt("\033G");   break;
  case 'H' : putfmt("\033W\1"); break;
  case 'I' : putfmt("\017");    break;
```

```
  case 'J' : putfmt("\033S\1"); break;
  case 'K' : putfmt("\033S\0"); break;
  case 'L' : putfmt("\033T");   break;
  case 'M' : putfmt("\0332");   break;
  case 'N' : putfmt("\0330");   break;
  case 'O' : putfmt("\033l\10");break;
  case 'P' : putfmt("\033A\30");break;
  case 'Q' : putfmt("\033N\4"); break;
  case 'R' : putfmt("\033\10"); break;
  case 'S' : putfmt("\033@\33A\6\33N\10\33S\1");
    }
  }
  while(sel!='X');
  putchar('\f');
  fclose(pr);
}
putml(s) char *s;
{ puts("      ");
  puts(s);
  putchar('\n');
}
putfmt(s) char *s;
{ fputs(s,pr);
}
```

Disk 45. Contents of file RANDOM;C

```
/*
** c99 random number functions
**
** by Clint Pulley, based on TI-FORTH
** last edit 85/11/16 0810
*/
/*
** initialize the random seed
*/
randomize()
{
#asm
 MOVB @>8802,0  RESET VDPSTA
 CLR  1
R#1
 INC  1         COUNT
 MOVB @>8802,0  READ VDPSTA
 ANDI 0,>8000
 JEQ  R#1       IF NOT VDP INT
 MOV  1,@>83C0  STORE SEED
#endasm
}
/* generate a 16-bit random number
*/
rndnum()
{
#asm
 MOV  @>83C0,0  GET SEED
 MPY  @R#C1,0
 A    @R#C2,1
 SRC  1,5
 MOV  1,8       RETURN NUMBER
 MOV  1,@>83C0  NEXT SEED
#endasm
}
#asm
R#C1  DATA >6FE5
R#C2  DATA >7AB9
#endasm
/* generate a random number between
** 0 and n-1
*/
rnd(n) int n;
{
#asm
 BL   *12      CALL rndnum
 DATA RNDNUM
 ABS  8
```

```
 CLR  7
 MOV  @2(14),9 GET n
 DIV  9,7      REMAINDER IN R8
#endasm
}
```

Disk 45. Contents of file RNDTST;C

```
/* random function test
*/
int i,avg,rn;
char cb[6];
main()
{ randomize();
  i=avg=0;
  while(++i<160)
  { rn=rnd(100);
    avg=avg+rn;
    puts(itod(rn,cb,6));
  }
  puts("\nAverage =");
  puts(itod(avg/160,cb,6));
}
#include dsk1.random;c
/*
** itod(nbr,str,sz) -
**        convert nbr to signed decimal string of width sz
**        right justified, blank filled, result in str[].
**        sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
}
```

Disk 45. Contents of file RUNOFF;C

```
/*
** text formatter transcribed from the book Software Tools
**
** Converted to c99 by Clint Pulley
**
** Last edit 86/03/23 0900
**
** Load with CFIO
*/
#include dsk1.stdio
#define PAGEWID   60 /* default page width */
#define PAGELEN   66 /* default page length */
#define MAXLINE  134
#define PAGECHAR '#' /* page number escape char */
#define HUGE   32700
/*
** global storage
*/
int fill  = YES,    /* in fill mode if YES */
    lsval = 1,      /* current line spacing */
    inval = 0,      /* current indent; >= 0 */
    rmval = PAGEWID,/* current right margin */
    tival = 0,      /* current temporary indent */
    ceval = 0,      /* number of lines to center */
    curpag= 0,      /* current output page number */
    newpag= 1,      /* next output page number */
    lineno= 0,      /* next line to be printed */
    plval = PAGELEN,/* page length in lines */
    m1val = 2,      /* top margin, including header */
    m2val = 3,      /* margin after header */
    m3val = 2,      /* margin after last text line */
    m4val = 3,      /* bottom margin, including footer */
    inp   = stdin,  /* input unit */
    inpsv = 0,      /* input unit save loc */
    bottom,/* last live line on page: */
           /*  = plval - m3val - m4val */
    outp,  /* index into outbuf */
    outw,  /* width of text in outbuf */
    outwds,/* number of words in outbuf */
    dir,   /* directions flag */
    out;   /* output unit*/
/*
** buffers in low RAM
*/
char *header, /* top of page title */
     *footer, /* bottom of page title */
     *outbuf, /* lines to be filled go here */
     *inbuf ; /* input line buffer */
```

```
/*
*/
main()
{ bottom = plval - m3val - m4val; /* invariant */
  header=9850;
  footer=10000;
  outbuf=10150;
  inbuf=10300;
  *header=*footer=*outbuf=*inbuf=0;
  puts("RUNOFF text formatter v1.4\n");
  out=getfn("Output","w132");
  puts("\nKeyboard input? (y/n) ");
  gets(inbuf);
  putchar('\n');
  if((*inbuf&95)=='Y')
  { puts("Enter data, terminate with CTRL-Z\n\n");
    roff();
  }
  while(inp=getfn("Input","r"))
  {
    roff();
    fclose(inp);
  }
  if (lineno > 0)
    space(HUGE);
  fclose(out);
}
/* get filename and open file
*/
getfn(text,m) char *text,*m;
{ int unit;
  char fn[20];
  unit=0;
  while(1)
  { puts(text);
    puts(" filename? ");
    gets(fn);
    if(!*fn)break; /* return 0 if null name */
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
/* format current file
*/
roff()
{ while(1)
  { while (fgets(inbuf,MAXLINE-1,inp))
    { strip(inbuf);
      if (*inbuf == '.')
        command();
```

```
      else
        text();
    }
    if(inpsv)
    { fclose(inp);
      inp=inpsv;
      inpsv=0;
      continue;
    }
    else break;
  }
}
/* strip out TI-Writer control characters
*/
strip(b) char *b;
{ if(*b==FF)
  { strcpy(b,".bp");
    return;
  }
  if(*b>127) /* TIW tab data */
  { strcpy(b,".co");
    return;
  }
  --b;
  while(*++b);
  if(*--b==13)*b=NULL;
}
/* perform formatting command
*/
command()
{ int val, spval,argtyp;
  val = getval(&argtyp);
  switch( (inbuf[1]<<8) + inbuf[2] )
  {
  case 'af' :
    if(inpsv)
    { puts("\nNested .af ignored\n");
      break;
    }
    appopen();
    break;
  case 'bp' :
    if (lineno > 0)
      space(HUGE);
    set(&curpag,val,argtyp,curpag+1,-HUGE,HUGE);
    newpag = curpag;
    break;
  case 'br' :
    brk();
    break;
```

```
  case 'ce' :
    brk();
    set(&ceval,val,argtyp,1,0,HUGE);
    break;
  case 'fi' :
    brk();
    fill = YES;
    break;
  case 'fo' :
    strcpy(footer,inbuf+3);
    break;
  case 'he' :
    strcpy(header,inbuf+3);
    break;
  case 'in' :
    set(&inval,val,argtyp,0,0,rmval-1);
    tival = inval;
    break;
  case 'ls' :
    set(&lsval,val,argtyp,1,1,HUGE);
    break;
  case 'nf' :
    brk();
    fill = NO;
    break;
  case 'pl' :
    set(&plval,val,argtyp,PAGELEN,m1val+m2val+m3val+m4val+1,HUGE);
    bottom=plval-m3val-m4val;
    break;
  case 'rm' :
    set(&rmval,val,argtyp,PAGEWID,tival+1,HUGE);
    break;
  case 'sp' :
    set(&spval,val,argtyp,1,0,HUGE);
    space(spval);
    break;
  case 'ti' :
    brk();
    set(&tival,val,argtyp,0,0,rmval);
  }
  return; /* ignore unknown commands */
}
/* evaluate optional numeric argument
*/
getval(argtyp) int *argtyp;
{ int i;
  i = 3;
  /* ..find argument.. */
  while (inbuf[i] == ' ')
    ++i;
  *argtyp = inbuf[i];
```

```
  if (*argtyp == '+' | *argtyp=='-')
    i++;
  return(atoi(inbuf+i));
}
appopen()
{ char *iptr;
  iptr=inbuf+3;
  while(*iptr==' ')++iptr;
  inpsv=inp;
  if(inp=fopen(iptr,"r"))return;
  puts("Can't open <");
  puts(iptr);
  puts(">\n");
  inp=inpsv;
  inpsv=0;
}
/* set parameter and check range
*/
set(param,val,argtyp,defval,minval,maxval)
int *param,val,argtyp,defval,minval,maxval;
{ int t;
  t=*param;
  switch(argtyp)
  {
  case NULL :
    *param = defval;
    break;
  case '+' :
    *param = t+val;
    break;
  case '-' :
    *param = t-val;
    break;
  default :
    *param = val;
  }
  *param = min(*param,maxval);
  *param = max(*param,minval);
}
/* process text lines
*/
text()
{ char wrdbuf[MAXLINE];
  int i;
  if (*inbuf==' ')
    leadbl(inbuf);  /* go left. set tival */
  if (ceval > 0)
  { center(inbuf);
    put(inbuf);
    ceval--;
```

```
  }
  else if (!*inbuf)
  { brk();
    put(" ");
  }
  else if (fill == NO)
    put(inbuf);
  else
  { i = 0;
    while (getwrd(inbuf,&i,wrdbuf) > 0)
      putwrd(wrdbuf);
  }
}
/* delete leading blanks.  Set tival
*/
leadbl(buf) char *buf;
{ int i,j;
  brk();
  /* find first non blank */
  i = 0;
  while (buf[i] == ' ')
    i++;
  if (buf[i] != NULL)
    tival = tival+i;
  else ++tival;
  /* move line to left */
  j = 0;
  while ((buf[j++] = buf[i++]) != NULL) ;
}
/* put out line with proper spacing and indenting
*/
put(buf) char *buf;
{ int i;
  if ((lineno == 0) | (lineno > bottom))
    phead();
  i = 1;
  while (i++ <= tival)
    putc(' ',out);
  tival = inval;  /* tival good for one line only */
  while(*buf)
  { putc(*buf++,out);
  }
  putc('\n',out);
  skip(min(lsval-1,bottom-lineno));
  lineno = lineno+lsval;
  if (lineno > bottom)
    pfoot();
}
/* put out page header
*/
phead()
```

```
{ curpag = newpag++;
  if (m1val > 0)
  {
    skip(m1val-1);
    puttl(header,curpag);
  }
  skip(m2val);
  lineno = m1val+m2val+1;
}
/* put out page footer
*/
pfoot()
{ skip(m3val);
  if (m4val > 0)
  {
    puttl(footer,curpag);
    skip(m4val-1);
  }
}
/* put out title line with optional page number
*/
puttl(buf,pageno) char *buf; int pageno;
{ while(*buf)
  { if (*buf == PAGECHAR)
      numout(pageno);
    else
      putc(*buf,out);
    ++buf;
  }
  putc('\n',out);
}
/* print page number
*/
numout(num) int num;
{ char nbuf[6],*ptr;
  itoa(num,nbuf);
  ptr=nbuf;
  while(*ptr)putc(*ptr++,out);
}
/* space n lines or to bottom of page
*/
space(n) int n;
{ brk();
  if (lineno > bottom)
    return;
  if (lineno == 0)
    phead();
  skip(min(n,bottom+1-lineno));
  lineno = lineno+n;
  if (lineno > bottom)
```

```
    pfoot();
}
/* output n blank lines
*/
skip(n) int n;
{ while ((n--) > 0)
  { putc(' ',out);
    putc('\n',out);
  }
}
/* get non-blank word from in[i] into out.
** increment *i.
*/
getwrd(in,ii,out) char *in,*out; int *ii;
{ int i, j;
  i = *ii;
  while (in[i]  == ' ') i++;
  j = 0;
  while((in[i]) & (in[i] != ' '))
    out[j++] = in[i++];
  out[j] = NULL;
  *ii = i;    /* return index in ii */
  return(j);  /* return length of word */
}
/* put a word in outbuf; includes margin justification
*/
putwrd(wrdbuf) char *wrdbuf;
{ int last, llval, w, nextra;
  w = width(wrdbuf);
  /* new end of wrdbuf */
  last = strlen(wrdbuf)+outp+1;
  llval = rmval-tival;
  if ( (outp > 0) & ((outw+w>llval) | (last>=MAXLINE)) )
  {
    /* too big */
    last = last-outp; /* remember end of wrdbuf */
    nextra = llval-outw+1;
    spread(outbuf,outp-1,nextra,outwds);
    if (nextra > 0 & outwds > 1)
      outp = outp+nextra;
    brk();  /* flush previous line */
  }
  strcpy(outbuf+outp,wrdbuf);
  outp = last;
  outbuf[outp-1] = ' '; /* blank between words */
  outw = outw+w+1;      /* 1 extra for blank */
  outwds++;
}
/* spread words to justify right margin
*/
spread(buf,outp,ne,outwds) char *buf; int outp,ne,outwds;
```

```
{ int nholes, i, j, nb;
  if (ne <= 0 | outwds <= 1)
    return;
  dir = 1-dir;  /* reverse direction */
  nholes = outwds-1;
  i = outp-1;
  /* leave room for EOS */
  j = min(MAXLINE-2,i+ne);
  while (i < j)
  {
    buf[j] = buf[i];
    if (buf[i] == ' ')
    {
      /* compute # of blanks */
      if (dir == 0)
        nb = (ne-1)/nholes+1;
      else
        nb = ne/nholes;
      ne = ne-nb;
      nholes--;
      /* put blanks in buffer */
      while (nb-- > 0)
        buf[--j] = ' ';
    }
    i--;
    j--;
  }
}
/* compute width of character string
*/
width(buf) char *buf;
{ int val;
  val = 0;
  while(*buf++)
    val++;
  return(val);
}
/* end current filled line
*/
brk()
{ if (outp > 0)
  {
    outbuf[outp-1] = NULL;
    put(outbuf);
  }
  outp = outw = outwds = 0;
}
/* center a line by setting tival
*/
center(buf) char *buf;
```

```
{ tival = max((rmval+tival-width(buf))/2,0);
}
max(a,b) int a,b;
{ if(b<a)return(a); else return(b);
}
min(a,b) int a,b;
{ if(b>a)return(a); else return(b);
}
/* concatenate t to end of s
*/
strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
}
/* copy t to s
*/
strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
/* return length of s
*/
strlen(s) int *s;
{ char *t;
  t=s-1;
  while(*++t);
  return(t-s);
}
/*
** n=atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
/*
** convert signed n to characters in s
*/
itoa(n,s) int n; char *s;
{ if(n<0)
  { *s++='-';
    n=-n;
  }
  itoneb(n,s,10);
}
```

```
/*
** convert unsigned n to chars in s using base b
**
** itoneb(n,s,b) int n,b; char *s;
*/
#asm
ITONEB MOV  @2(14),7     BASE
       MOV  @6(14),2     N
       LI   6,BUF#       WORK BUFFER
       CLR  5            DIGIT COUNTER
ITONB1 CLR  1
       DIV  7,1          N/BASE, REM IN 2
       AI   2,48         MAKE CHAR
       CI   2,58
       JL   ITONB2       IF REM<2
       AI   2,7
ITONB2 SWPB 2
       MOVB 2,*6+        TO BUF
       INC  5            COUNT
       MOV  1,2          QUOTIENT
       JNE  ITONB1       IF MORE DIGITS
       MOV  @4(14),7     S
ITONB3 DEC  6            STORE IN REVERSE
       MOVB *6,*7+
       DEC  5
       JNE  ITONB3       IF MORE
       SB   *7,*7        ZERO BYTE
       B    *13
*
BUF#   BSS  8
#endasm
```

Disk 45. Contents of file RUNOFFDOC

RUNOFF text formatting program — documentation

This is a translation of the text formatting program from Kernighan and Plauger's book *Software Tools*, modified to take advantage of the features of C. If the following command summary is not sufficient, please refer to the book.

command	*break?*	*default*	*result*
.af fn	no		append input file named fn
.bp n	yes	+1	begin page numbered n
.br	yes		cause a paragraph break
.ce n	yes	1	center the next n lines
.co	no		includes a comment in input
.fi	yes		begins fill mode
.fo s	no	empty	sets footer text
.he s	no	empty	sets header (page top) text
.in n	no	0	indent lines by n spaces
.ls n	no	1	sets line spacing
.nf	yes		stops filling
.pl n	no	66	sets lines per page
.rm n	no	60	sets right margin
.sp n	yes	1	spaces down n lines
.ti n	yes	0	temporary indent of n spaces

Run Instructions:

After RUNOFF has been loaded in the usual manner, it will identify itself and prompt for an output filename. This will normally be the same printer filename (PIO, RS232.BA=4800, etc.) that is used for listing files with the Editor. Next, the program will request keyboard input. This allows titles, margins, etc. to be input. Press **CTRL Z** to end keyboard input. The program will then prompt for an input filename and proceed to print it. When the input file has been printed, the program will prompt for another filename. When all files have been printed, it is essential to respond to the prompt by pressing **ENTER**. This causes the fill buffer to be flushed and the last footer to be printed before program exit.

If the c99 rerun option is selected, all program buffers are reset, but all other formatting parameters (including page number) remain set from the previous execution.

Notes:

1. In the above table, n designates a numeric constant. If it unsigned, it is an absolute value. If the number is preceded by + or – it is a relative value and is added to or subtracted from the present value of that parameter.

2. In the table, s represents a string of characters used for a title. If the string includes a #, the current page number is substituted for the #.

3. Filling consists placing as many words as possible on each line and right-justifying them. Paragraph breaks are generated by .br, an empty line, or a line beginning with space(s). Paragraph indentation is preserved.

4. The output device is opened with a maximum record length of 132 characters, so if the printer has been set for condensed mode it is possible to print lines up to 132 characters wide by setting .in and .rm.

5. The book *Software Tools* by B.W. Kernighan and P.J. Plauger (Addison-Wesley 1976) is a standard reference which is well worth reading.

Disk 45. Contents of file SOUNDS;C

```
/* Test of GPL linkage */
int c;
main()
{ while((c=getchar())>0)
  { if(c<='A')
      beep();
    else honk();
  }
}
#asm
 REF C$GPLL
BEEP BL @C$GPLL
 DATA >34
 B *13
HONK BL @C$GPLL
 DATA >36
 B *13
#endasm
```

Disk 45. Contents of file STRINGFNS

```
/*
** string function library
**
** contributed by :
**
** Tom Wible
** 203 Cardinal Glen Circle
** Sterling, VA
** USA 22170
*/
strlen(s) /* returns string length*/
char *s;
{
     int n;
     n = 0;
     while (*s++)
          ++n;
     return (n);
}

strcmp(s1, s2) /* compares s1 to s2, returns n<0 if s1 before s2, etc*/
char *s1, *s2;
{
     int r12;
     while ((*s1) | (*s2)) {
          r12 = (*s1) - (*s2);
          if (r12)
               return (r12);
          else {
               ++s1;
               ++s2;
          }
     }
     return (0);
}

stncmp(s1, s2, n) /* compares n chars of s1 to s2, returns n<0 if s1 before s2*/
char *s1, *s2;
int n;
{
     int r12;
     while ((*s1 | *s2) & (n--)) {
          r12 = (*s1) - (*s2);
          if (r12)
               return (r12);
          else {
               ++s1;
               ++s2;
```

```
          }
     }
     return (0);
}

index(s, c) /* returns location of c in s */
char *s, c;
{
     int n;
     n = 1;
     while (*s) {
          if (*s == c)
               return (n);
          else {
               ++*s;
               ++n;
          }
     }
     return (0);
}

rindex(s, c) /* returns location of c in s from right*/
char *s, c;
{
     int n;
     n = 0;
     while (*s++)
          ++n;           /*get length of string*/
     --*s;               /*point to last char  */
     while (n) {
          if (*s == c)
               return (n);
          else {
               --*s;
               --n;
          }
     }
     return (0);
}

strcat(s1, s2) /* concatenate s2 to end of s1*/
char *s1, *s2;
{
     char *p;
     p = s1;   /*point to s1*/
     while (*p++)
          ;    /*point to end of s1*/
     --p;
     while ((*p++) = (*s2++))
          ;    /*move s2 to end of s1*/
     return;
```

```
}

stncat(s1, s2, n) /* concatenate at most n chars of s2 to end of s1*/
char *s1, *s2;
int n;
{
     char *p;
     p = s1;   /*point to s1*/
     while (*p++)
          ;    /*point to end of s1*/
     --p;
     while (((*p++) = (*s2++)) & (n--))
          ;    /*move s2 to end of s1*/
     if (!(n))
          *p = NULL;
     return;
}

strcpy(s1, s2) /*copy s2 into s1*/
char *s1, *s2;
{
     while ((*s1++) = (*s2++))
          ;
     return;
}

stncpy(s1, s2, n) /*copy at most n chars of s2 into s1*/
char *s1, *s2;
int n;
{
     while (((*s1++) = (*s2++)) & (n--))
          ;
     if (!(n))
          *s1 = NULL;
     else
          while (n--)
               *s1++ = NULL;
     return;
}
```

Disk 45. Contents of file TCIOC

```
/*
 * Written by:
 *
 *  Tom Bentley
 *  P.O. Box 346
 *  Osgoode, Ont.
 *  Canada  K0A 2W0
 */

#asm
 REF DSRLNK,VMBW,VSBW,VSBR,VMBR,GRMRA,GRMWA
 DEF TOPEN,TCLOSE,TREAD,TWRITE
#endasm

#define PAB1      8192
#define PAB2      8498
#define PAB3      8804
#define PAB4      9110
#define PABLEN      50
#define NUMFIL       4
#define DSRCLR      31
#define WS       33536


char *status;
int  *wsp;
int  dsrerr,saveg;
int  filalloc[NUMFIL]={PAB1,PAB2,PAB3,PAB4};
int  filstk[NUMFIL]={0,0,0,0};
char pab[PABLEN];

topen(name,access,size)
  char *name;
  char access,size;
  {
  int  *pb,lp,fp;

  wsp = WS;    /* >8300 */


  /* set pab up for open */
  pab[0] = 0;     /* op code */
  pab[1] = access;
  /* data buffer address */
  fp=0;
  dsrerr=4; /* if file table is full */
  while(fp<NUMFIL-1) {
    if(filstk[fp]==0) {
```

```
      filstk[fp]=filalloc[fp];
      pb=&pab[2];
      *pb=PABLEN+filstk[fp++];
      dsrerr=0;
      break;
      }
    else
      fp++;
    }
  if(dsrerr==0) {
    pab[4] = size;  /* record length */
    pab[5] = size;  /* character count */
    pab[6] = 0;     /* record number */
    pab[7] = 0;
    pab[8] = 0;     /* screen offset */
    /* name length */
    pab[9] = tlen(name);
    /* device name... */
    lp=10;
    while(*name)
      pab[lp++]=*name++;
    if(linkdsr(fp)!=-1)
      fp=-dsrerr;
    }
  else fp=-dsrerr;
  return(fp);
  }

tclose(fp)
  int fp;
  {
  int eof;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp=WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=1;
  eof=linkdsr(fp);
  if(eof!=-1&eof!=7)
    return(-2);
  filstk[fp-1]=0;
  fp=0;
  }
```

```
tread(buff,rec,fp,size)
  char buff[];
  int  rec,fp,*size;
  {
  int  *pb,eof;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp = WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=2; /* read op code */
  pab[1]=pab[1]&DSRCLR;
  /* if fixed and relative */
  if((pab[1]&17)==1 & rec>=0) {
    pb=&pab[6];
    *pb=rec;
    }
  eof=linkdsr(fp);
  if(eof==5)
    return(-1);
  if(eof!=-1)
    return(-2);
  wsp=WS;
  pb=&pab[2];
  *wsp++=*pb;
  *wsp++=buff;
  *wsp=pab[4];
  #asm
   MOV  1,6
   BLWP @VMBR * MOVE DATA TO RAM
  #endasm
  wsp = WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR * MOVE PAB TO RAM
   MOV  1,4
   AI   4,5   * NUM OF CHAR READ
   CLR  5
   MOVB *4,5
   SWPB 5
   MOV  5,2   * SIZE OF BUFFER
   A    5,6
   MOVB 5,*6  * TERMINATE WITH NULL
  #endasm
```

```
  *size=*wsp;
  return(0);
  }

twrite(buff,rec,fp,size)
  char buff[];
  int  rec,fp,size;
  {
  int  *pb,fr;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp=WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=3; /* write op code */
  pab[1]=pab[1]&DSRCLR;
  /* if fixed and relative */
  fr=pab[1]&17;
  if(fr==1) {
    pb=&pab[6];
    *pb=rec;
    }
  if(size)
    pab[5]=size;
  else pab[5]=tlen(buff);
  wsp=WS;
  pb=&pab[2];
  *wsp++=*pb;
  *wsp++=buff;
  *wsp=pab[5];
  #asm
   BLWP @VMBW * MOVE BUFF TO VDP
  #endasm
  if(linkdsr(fp)!=-1)
    return(-2);

  return(0);
  }

linkdsr(fp)
  int fp;
  {
  wsp = WS;    /* >8300 */
  status = 33660; /* >837C */
```

```
  *status = 0;

  /* set up workspace registers */
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=10+pab[9];
  #asm
   BLWP @VMBW
   MOV  0,4
   AI   4,9
   MOV  4,@>8356 * DEVICE POINTER
   SETO @DSRERR
   MOVB @GRMRA,@SAVEG
   NOP
   MOVB @GRMRA,@SAVEG+1
   DEC  @SAVEG
   BLWP @DSRLNK
   DATA 8
   JEQ  DE
   JMP  DA
DE SRL  0,8
   MOV  0,@DSRERR
DA MOVB @SAVEG,@GRMWA
   NOP
   MOVB @SAVEG+1,@GRMWA
  #endasm
  return(dsrerr);
  }


tlen(s)
  char *s;
  {
  char *t;
  t=s-1;
  while(*++t);
  return(t-s);
  }
```

Disk 45. Contents of file TCIODOC

Enhanced I/O Library (by Tom Bentley 860201)

The following is a description of a new I/O library that may be used by c99 users to perform I/O functions on any file type on the TI. *Note*: This library was written using Clint Pulley's c99 program.

I. Library and Include Files

a) TCIO — The file input/output library. This contains the file tables and all file I/O functions.

b) TCIOI — Include file containing I/O definitions for TCIO file functions as well as extern directives. This file will not conflict with the I/O include files that come with the compiler.

Functions in TCIO:

- Open a file

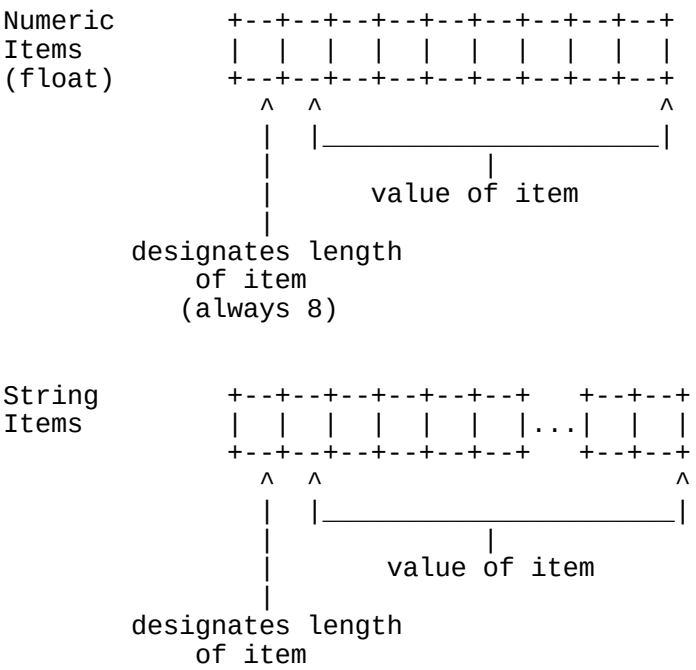
```
int filptr;
char name[],access,fsize;

filptr=topen(name,access,fsize);
```

Name contains a valid filename in lower or upper case, access contains all the access information for the file (see include file for access information) and fsize contains the size in bytes of the file. If the file is opened for input the fsize parm may be set to zero (undefined) to tell the system to determine the size of the file. If the open is successful filptr will contain a value greater than 0, otherwise it will contain the error code (see the include file for open errors).

DATA TYPE INFORMATION

DISPLAY-type data has the same form as data entered from the keyboard. If you wish to use this file between 'C' and Basic then the computer knows the length of each data item in a DISPLAY-type record by the comma separators placed between items (basic will locate the items automatically but in 'C' you have to locate the items yourself).

INTERNAL-type data has the following form:



The computer knows the length of each INTERNAL-type item by interpreting the one-position length indicator at the beginning of each item (Basic will interpret the item automatically when inputting or printing but you must do the interpretation of items yourself when in 'C').

No validation of INTERNAL-type data-items is performed. All numeric items must be 9 positions long (8 digits plus one position which specifies the length) and must be valid representations of float-point number (see floating point library).

For more information refer to pages II-126 and II-132 of the TI-99/4A *User's Reference Guide*. Just keep in mind that Basic will do all of the translation effort while you must do the translation if in 'C'.

Example 1.

```
filptr=topen("DSK1.MYFILE",UPDATE+INTERNAL+RELATIVE+FIXED,100)
if(filptr<1) {
  /* error */
  }
```

Example 2. (open as an undefined file size)

```
filptr=topen("DSK1.MYFILE",INPUT+DISPLAY+SEQUENTIAL+FIXED,0)
if(filptr<1) {
  /* error */
  }
```

- Read a file

```
int eof,rec,filptr,size ;
char buff[];

eof=tread(buff,rec,filptr,&size);
```

Reads the file specified by filptr. If eof returns -1 then it is a valid end of file condition, if it returns -2 then it is an error. Buff is a character buffer, this buffer should be one byte more than the size of the file, for example if you open the file with a size of 80 then the definition of buff should say buff[81]. If you've opened the file with an undefined size you should ensure the buffer size will be big enough to hold the longest record, to be safe use buff[256]. Rec is used to perform direct record reads, if the file is open with access relative and rec is -1 then the file will be read sequentially, otherwise the record specified in rec is read. Size will contain the actual number of bytes read from the disk.

Example 1.

```
/* Read the file sequentially */

while(1) {
  eof=tread(buff,RELSEQ,filptr,&size);
  if(eof==TEOF)
    break;
  else if(eof==TERR) {
    puts("File Error");
    break;
    }
  /* Display Buffer */
  puts(buff);
  }
```

Example 2.

```
/* Read using directed reads */
rec = 0;
while(1) {
  eof=tread(buff,rec++,filptr,&size);
  if(eof==TEOF)
    break;
  else if(eof==TERR) {
    puts("File Error");
```

```
    break;
    }
  /* Display Buffer */
  puts(buff);
  }
```

- Write to a file

```
int eof,rec,filptr,size;
char buff[];

eof=twrite(buff,rec,filptr,size);
```

Writes to the file specified by filptr. If eof returns -2 then there is a file error. Buff is a character buffer. If the file is a relative file then rec will contain the record number of where to write the buffer to. If size is not equal to zero the write will perform a write of size bytes to the disk, otherwise it will write up to the null terminator of buff.

Example 1.

```
/* Write to sequential file */

while(there is data) {
  eof=twrite(buff,0,filptr,size);
  if(eof) {
    puts("File Error");
    break;
    }
  }
```

Example 2.

```
/* Write to relative file */

while(there is data) {
  rec = record to update
  eof=twrite(buff,rec,filptr,size);
  if(eof) {
    puts("File Error");
    break;
    }
  }
```

- Close a file

```
int eof,filptr;

eof=tclose(filptr);
```

Closes the specified file, if an error occurs eof is set to -2.

Example 1.

```
/* Close a file */

eof=tclose(filptr);
if(eof)
  puts("File Error");
```

Disk 45. Contents of file TCIOI

```
/* Include file for TCIO file library */

extern topen(),tclose(),tread(),twrite();

/* File Type  */
#define SEQUENTIAL 0
#define RELATIVE   1

/* Record Type */
#define FIXED      0
#define VARIABLE   16

/* Data Type */
#define DISPLAY    0
#define INTERNAL   8

/* Mode of Operation */
#define UPDATE     0
#define OUTPUT     2
#define INPUT      4
#define APPEND     6

/* read relative file*/
/* sequentially. use */
/* this in place of  */
/* the record number */
#define RELSEQ     -1

/* Error & Eof Codes */
#define TERR       -2
#define TEOF       -1

/* Open Errors */
/* for more info see */
/* page 299 in E/A.  */
#define BADDEV     0
#define PROTECTD   -1
#define BADATTR    -2
#define ILGALOP    -3
#define NOSPACE    -4
#define READEOF    -5
#define DEVERR     -6
#define NOFILE     -7
```

Disk 46A. Funnelweb 4.12

Version: 4.12 *Author*: Will and Tony McGovern
Requires: XB or EA or TIW *Language*: AL, XB *Updated*: 10/28/88

Eliminates the need for the TI-Writer and Editor/Assembler cartridges while supplying many additional features. A truly integrated environment for the TI including an editor, formatter, disk manager, assembler, disk sector editor, Forth loader, c99 compiler, terminal emulator loader, and the ability to add features and customize. Perhaps the most important TI disk ever. Requires BCS disk 61. (10/22/87)

```
dskdir. v2.0. 12-dec-96
Disk name               = FUNL4*1A
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 AS           33 PROGRAM      >022 032
002 >003 AT           22 PROGRAM      >042 021
003 >004 CHARA1        5 PROGRAM      >057 004
004 >005 CHARA2        5 PROGRAM      >05b 004
005 >006 DP           17 PROGRAM      >05f 016
006 >007 DS            4 PROGRAM      >06f 003
007 >008 EA            9 PROGRAM      >072 008
008 >009 ED           33 PROGRAM      >07a 032
009 >00a EE           19 PROGRAM      >09a 018
010 >00b FO           33 PROGRAM      >0ac 032
011 >00c FP           15 PROGRAM      >0cc 014
012 >00d LL           10 PROGRAM      >0da 009
013 >00e LOAD         31 PROGRAM      >0e3 030
014 >00f MG           33 PROGRAM      >101 032
015 >010 MH           23 PROGRAM      >121 022
016 >011 QD           12 PROGRAM      >137 011
017 >012 SL           10 PROGRAM      >142 009
018 >013 SYSCON        6 PROGRAM      >14b 005
019 >014 UL            4 PROGRAM      >150 003
020 >015 UTIL1        33 PROGRAM      >153 021 >016 011
```

Disk 46B. Funnelweb 4.40

Version: 4.40 *Author*: Will and Tony McGovern
Requires: XB or EA or TIW *Language*: AL, XB *Updated*:

```
dskdir. v2.0. 12-dec-96
Disk name               = FWB4*40-1
Sectors total           = 720
Sectors used            = 716
Sectors available       = 2
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 2
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 AR           33 PROGRAM      >022 032
002 >003 AS           33 PROGRAM      >042 032
003 >004 AT           23 PROGRAM      >062 022
004 >005 C1            5 PROGRAM      >078 004
005 >006 C2            5 PROGRAM      >07c 004
006 >007 C99PFI;O      2 DIS/FIX 80   >080 001
007 >008 CF           32 PROGRAM      >081 031
008 >009 CG           26 PROGRAM      >0a0 025
009 >00a CHARA1        5 PROGRAM      >0b9 004
010 >00b CP            4 PROGRAM      >0bd 003
011 >00c CT8K/O       17 DIS/FIX 80   >0c0 016
012 >00d D1            4 PROGRAM      >0d0 003
013 >00e DR           33 PROGRAM      >0d3 032
014 >00f DR80         40 PROGRAM      >0f3 039
015 >010 DR81         34 PROGRAM      >11a 033
016 >011 DS           32 PROGRAM      >13b 031
017 >012 DU           33 PROGRAM    Y >15a 032
018 >013 DV           33 PROGRAM    Y >17a 032
019 >014 DW           30 PROGRAM    Y >19a 029
020 >015 EA            9 PROGRAM      >1b7 008
021 >016 ED           33 PROGRAM      >1bf 032
022 >017 ED80         33 PROGRAM      >1df 032
023 >018 ED81         21 PROGRAM      >1ff 020
024 >019 EE           18 PROGRAM      >213 017
025 >01a FO           33 PROGRAM      >224 032
026 >01b FP           16 PROGRAM      >244 015
027 >01c FW           33 PROGRAM      >253 032
028 >01d LL           10 PROGRAM      >273 009
029 >01e LOAD         32 PROGRAM      >27c 031
030 >01f ML            4 PROGRAM      >29b 003
031 >020 ML80          4 PROGRAM      >29e 003
```

```
032 >021 QD           12 PROGRAM      >2a1 011
033 >183 QF           11 PROGRAM      >2ad 010
034 >17c SL           13 PROGRAM      >2b8 012
035 >177 SYSCON        6 PROGRAM      >2c5 005
036 >16b UL            4 PROGRAM      >2cb 003
```

Disk 47. Disk One

Version:
Requires: XB
Author: John Bonavia
Language: XB, AL
Updated: 08/17/86

Previously available commercially for over $20. Programs to automatically create menued documentation for your programs, to compile a catalog of all your disks, to edit files, a text formatter, and much more. Excellent code to use, learn from, and customize.

```
dskdir. v2.0. 12-dec-96
Disk name               = DISK_ONE
Sectors total           = 360
Sectors used            = 246
Sectors available       = 112
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >014 -README       8 DIS/VAR 80   >0fe 007
002 >002 CAT           4 PROGRAM      >022 003
003 >003 DISPL         4 DIS/FIX 80   >025 003
004 >004 DISPLS        8 DIS/VAR 80   >028 007
005 >005 DOCUM         3 PROGRAM      >02f 002
006 >006 EDIT         19 PROGRAM      >031 018
007 >007 EDITINFO     28 DIS/VAR 80   >043 027
008 >008 INFO          8 PROGRAM      >05e 007
009 >009 INTRO         7 DIS/VAR 80   >065 006
010 >00a LIB          22 PROGRAM      >06b 021
011 >00b LIBINFO      17 DIS/VAR 80   >080 016
012 >00c LIBSCAN      17 PROGRAM      >090 016
013 >00d LOAD         11 PROGRAM      >0a0 010
014 >00e LOADINFO     28 DIS/VAR 80   >0aa 027
015 >00f NULL          2 PROGRAM      >0c5 001
016 >010 SHELL         3 DIS/VAR163   >0c6 002
017 >011 TEXT         16 PROGRAM      >0c8 015
018 >012 TEXTINFO     13 DIS/VAR 80   >0d7 012
019 >013 XBLIST       28 PROGRAM      >0e3 027
```

Disk 47. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain Disk #47

DISK_ONE

Disk_one is a collection of programs written by John Bonavia. It was previously available commercially for $22.95. John has since consented to allow us to distribute this program. He has NOT released it into the public domain. He still maintains his Copyright. This disk is being made available by agreement between Bonavia and the Boston Computer Society TI User Group. Other user groups may distribute this program through their regular software library channels. It may not, however, be distributed by any commercial ventures in any way without the expressed consent of John Bonavia.

With the exception of the addition of this file, this disk is exactly as it was when it was commercially available. Thus you will see references to its price and a request to send $11.95 to Bonavia if you obtained this program through pirate channels. If you find the disk useful (and there is some exceptional code here) please consider sending John a donation for his work.

The documentation for the various parts of this disk are contained in the DIS/VAR 80 files on the disk. They can be read using some of the utilities on the disk. The best way to do this is to just put this disk in drive one and start up Extended BASIC.

Please do not break up this disk. That is, if you give a copy to someone, please include ALL files from the original.

Thank you

J. Peter Hoddie
Co-director
Boston Computer Society
TI-99/4A User Group
April 14, 1986

Disk 47. Contents of file DISPLS

```
*************************************************************
*    ROUTINE TO DISPLAY UP TO 255 BYTES. LIKE DISPLAY AT BUT *
*    GIVES THE FULL 32 BYTES WIDTH AND DOES MULTIPLE LINES.  *
*    CALL IT WITH "CALL LINK("DISPL",A$,R,C) WHERE A$ IS     *
*    THE TEXT TO DISPLAY AND R AND C ARE ROW AND COLUMN.     *
*************************************************************
       DEF  DISPL
*  XBASIC EQUATES FOLLOW
       RORG
NUMREF EQU  >200C        UTILITY
STRREF EQU  >2014
VMBW   EQU  >2024        UTILITY
PARM   EQU  >834B        2ND BYTE OF FAC
STATUS EQU  >837C        ERROR BYTE
DISPL  LI   R1,1         FOR FIRST PARM1
       CLR  R0
       LI   R2,BUFFL
       MOVB @K255,@BUFFL
       BLWP @STRREF      GET STRING
       LI   R1,2         FOR SECOND PARM
       CLR  R0
       BLWP @NUMREF      GET THE ROW
       CLR  R2
       MOVB @PARM,R2
       SWPB R2
       DEC  R2       MAKE ROW RELATIVE TO ZERO NOT 1
       SLA  R2,5     MULT BY 32
       LI   R1,3     NOW FOR COLUMN
       CLR  R0
       BLWP @NUMREF
       CLR  R3
       MOVB @PARM,R3 GET COLUMN
       SWPB R3
       DEC  R3       MAKE RELATIVE TO ZERO
       MOV  R3,R0    SCREEN POSITION IN 0
       A    R2,R0
       CLR  R5
       MOVB @BUFFL,R5 LENGTH FOR OFFSET LOOP
       SWPB R5
       LI   R6,BUFF
LOOP   AB   @K60,*R6+
       DEC  R5
       JNE  LOOP
       CLR  R2       FOR NUMBER OF BYTES
       MOVB @BUFFL,R2
       SWPB R2
       LI   R1,BUFF  DATA ADDR
       BLWP @VMBW
```

```
       MOVB @ZERO,@STATUS
       RT
ZERO   DATA 0
K255   BYTE 255
K60    BYTE >60
BUFFL  BYTE 0
BUFF   BSS  255
       END
```

Disk 47. Contents of file EDITINFO

ABOUT THE EDITOR

This editor is designed to appear similar to the editing facilities in BASIC: i.e. when you key in text with this, it is quite like keying in a program. For instance, you enter data with line numbers: you have a "LIST" command which works the same way: and you use SAVE, OLD, and MERGE to store and recall your text files on disk.

However, this program has some nifty things that BASIC doesn't have. For instance, you can delete a range of lines. You can copy one or a block of lines to another location. You can move lines. And you can make text changes either to a single line, or to all lines in the file — that is, you can say "Change JONES wherever it appears, to SMITH". (Would be nice to have these in BASIC, wouldn't it?) On the other hand, this program doesn't have the line recall/edit facility that BASIC has. You can change any line at any time, but you must either retype the line or enter a change command (text replacement: see section on Changing Data).

Other headings of this guide will show you how to use the editor in detail. Here are a few key facts:

- The text you are working on is held all in memory. You can get maybe 200 long lines (up to 80 characters) or 500 shorter ones. Program has a max of 700. You always use line numbers when editing, but they are not stored with the data on the disk. You can put any keyboard characters into the text.

- The disk file is always stored as "Display, Variable 80", so it is compatible with files produced by the Assembler Editor. It is also compatible with the "INFO" program on this disk — in fact, it is an ideal way to prepare data for that program.

ENTERING DATA

You key in each line of text in the format: Line number, space, your text. Line number is from 1 to 700. Lines don't have to be entered consecutively, or in any particular order.

There is no RES command as in BASIC, but you can achieve the same effect by doing a SAVE to disk, and then reloading the file with OLD. OLD lets you specify the line numbers to be assigned when loading the data. Remember that line numbers are not saved with the data. When you reload a file from disk, the editor will number the lines starting at 1. but you can override this to any start and increment you want — see the section on SAVE and OLD.

To change a line, you can retype it (with its line number, of course) or use the change command — see that section. The "CLEAR" command will clear all data from memory (for when you just have to start over!). Another use is when you have just finished and saved one document, and want to start another.

A final word about lower-case: The Editor happily accepts any text you key in in lower-case (actually TI's small caps). Lowercase versions of commands are also accepted, except for SAVE, OLD and MERGE.

The general response to any incorrect entry is "?". The lack of wordy messages leaves more room for text.

COPY, MOVE, AND DELETE

You can copy a group of lines from one place to another with this command: C A-B,L

> A = starting number of lines to be copied
> B = ending number of lines to be copied
> L = number of line after which the copied lines are to be inserted. Any existing lines in that place will be moved on.

To copy a single line, just make A and B the same, e.g.:

```
C 104-104,120
```

This puts a copy of line 104 after line 120. You can copy to places after or before the source lines.

Moving lines is similar. It is done with the M command (same format as "C") and the only difference is that the moved lines are deleted after being moved. A typical MOVE would be:

```
M 15-25,30
```

Useful Trick: When you want to insert some blank lines between some existing lines, just "MOVE" some unused lines from the high end of the text (up to line 700) to the place you want.

Deleting lines is done with DEL A-B where a and B are the start and finish numbers of lines to be deleted.

You can also delete a line as in Basic, by typing the line number by itself. If you type one or more spaces, the line is kept as a blank line.

CHANGING TEXT

There are 3 ways to change your text:

1. Re-key the line
2. Specify a change for the line.
3. Specify a GLOBAL change, i.e. one which takes effect through all your text.

The command for changes is: /old/new/ line-number, i.e., you put the text to be changed between two slashes, and the replacement text right after, closing with another slash. Then, if you want the change to be done just on one particular line, you give that line number. Otherwise, you can leave off the line number, and the Editor will scan all your lines of data and change that text wherever it is found. You can include spaces and punctuation in the text to be changed, or to be inserted. You can remove a word, too: Suppose you typed "Let's go to Paris in the the spring". Take out the first "the" with:

```
/the//  (line-number)
```

Be careful of unplanned effects. Suppose you typed in "ber" when you meant to put "bar". You did this several times, so you do a global change:

```
/ber/bar/
```

You will change "number" to "numbar", and "berth" to "barth". You would need

```
/ ber / bar /
```

in this case. Of course, then if "ber" was followed by a comma or period, you wouldn't pick it up.

The FIND command - F text

This is another useful aid. It lets you find any string of characters, scanning your whole text for a match. When it finds a match, it displays that line and stops. If you press **ENTER**, it will go on and look for more.

Examples:

```
F Jones
F Class of '84
```

Note that upper and lower case are not equivalent. If you want to find "however", but it may sometimes be "However", you should try "F owever ".

STORING TEXT ON DISK

To save your text on disk, just type SAVE filename. If the first 3 letters of filename aren't "DSK", the program will put "DSK1." in front of the name for you.

To load a file from disk, type OLD filename. This will bring in the text numbered from 1 by 1. To change that, give a start and increment with the OLD, e.g:

```
OLD PAYROLL 100 10
```

This loads DSK1.PAYROLL numbered 100,110,120. . .

You can also say MERGE, and you can give numbers for that too.

As you may expect, merge text combines with text in memory by line-number.

Note: You don't have to give a "CLEAR" command before you load a file with "OLD". "OLD" will clear any old data from memory anyway. ("MERGE" will not, of course).

Disk 47. Contents of file INTRO

DISK_ONE

Bona-Via,
39 Bradford St.,
Needham MA 02192

Preface

This introduction was printed by the "TEXT" program from "INTRO", which is included on the disk as an example of "TEXT" control codes.

The main text of the documentation was printed from the disk tutorial files with the "DOCUM" program.

General notes.

The programs are fairly well debugged. (We don't claim 100% — who does?). They are good enough that we suggest this approach. If you have a problem, consider first the question — "Am I doing it right?" Recheck the instructions. Next — are you sure this isn't simply a limitation of the program? The programs usually keep all their data in memory. Memory is a scarce resource on the TI, so some neat program ideas are left out to leave more space for data.

These programs were developed on a system with SS/SD disk, serial RS232 printer, and X-Basic 110. Will be glad to get comments on use in other setups. Since the programs are not protected, they can be easily modified to suit your needs.

Finally: please back up the disk and use the copy, not the master!

We make this easy for you by not copy-protecting, so we will charge for replacements.

GENERAL GUIDE

THE EDITOR

THE LIBRARIAN

PROGRAM LISTER

TEXT PRINTER

Disk 47. Contents of file LIBINFO

GENERAL DESCRIPTION

There are two programs,

```
DSK1.LIB
DSK1.LIBSCAN
```

LIB is used to create and maintain your catalog while LIBSCAN is used to look at the catalog on the screen, to search for data and to produce printouts.

You can run either program at any time: when running one program, you can transfer to the other with a single keystroke.

Note that if you have been updating or sorting the catalog using LIB, you must save it on disk before you switch over to get a print or to scan it. Each program comes into memory like a new program and loads its file from disk.

Both programs are controlled by simple selection from a menu.

When each task is done, the program returns to the menu for the next task. The programs assume that all your disks have unique names. If you have two disks called "MISC', for instance, and you try to put them both in the same catalog, you can do it, but you won't know which is which, and when you next update the catalog, only the last one you put in will stay there. Unique names are really the only way to go, anyway.

THE LIB PROGRAM

Here is the normal sequence of operations to use this program.

You run "DSK1.LIB". You would begin your disk library catalog by choosing option 1 from the menu, "CREATE NEW FILE". Then the program lets you specify up to three common filenames which you don't want to keep in the catalog (perhaps files that you keep copies of on almost every disk). After that, the program will ask you to put a disk for cataloging into the drive and press **ENTER**. The screen will show the filenames as they are processed. As you catalog more disks, the system will take a little longer to do each one.

The capacity of the catalog is about 350-400 files: it depends mainly on how long your disk volume names are (not the filenames). With single-letter disk names, you can get almost 500 files cataloged. Of course, there is nothing to stop you from having several catalogs, perhaps one for business, one for games, etc.

When a disk has been read, you are asked for another. When you have cataloged all the disks you want, press **M** to return to menu. Note that in general you can press **M** in response to any prompt to cancel that action and return to menu.

Now your catalog has been built in memory. You will want to sort it — usually by filename. This takes 3-4 minutes for 250 files. Then you save it on disk.

Now you may wish to look at it or print it, so you choose option 7, "RUN THE SCAN/PRINT PROGRAM". See the next menu section for details of this. For now, we'll carry on with the main Librarian Program.

To keep your file updated, all you do is choose option 2 to load in the catalog you made before. Then take option 3 and put in only the disks that have been changed. The LIB program replaces all entries for the updated disks. You should then sort the file.

Note that you can sort the file by diskname. It can be useful to have this print as well as one in filename sequence.

THE LIBSCAN PROGRAM

LIBSCAN has options to print the file, scan the file, do a keyword search or return to LIB For further work.

The Print Option is very simple. Just give it the same printer specification you would give in a program — e.g. RS232.BA=9600, or whatever you use. If you want, you can put the print on a disk file to be run off later.

Viewing the file is done a page at a time. You press **ENTER** to go forwards and **BACK** to go back. However, there is a neat twist as well: if you key a digit (1-9) before pressing **ENTER** or **BACK**, you will skip forward or back that number of pages. This is a great time-saver when you are looking through a large catalog.

The Keyword Option scans file and/or disk names (your option) for a match. It goes right through the file, since it does not assume you want just names that begin with the key.

Disk 47. Contents of file TEXTINFO

GENERAL RULES

This is program "DSK1.TEXT".

The Text Printer reads a disk file (which must be stored as DISPLAY, VARIABLE 80) and prints the lines. Every 57 lines it gives a page-throw command to start up a new page. Also, it takes note of any control codes it finds in the text. Control codes are put in by you when you are writing the text. They are on a line by themselves (with one exception) and they consist of a period in position 1 of the line and a brief code immediately following. For instance .PG means "start a new page right now". For details of all the codes, select "CONTROL CODES" from the menu

PAGE HEADINGS

You can get a running heading on every page of your printed document in one of two ways:

1. Wait until you are printing the document, then type in the heading you want when the program asks you. This allows you to print the same text with different headings at different times.

2. Or you can put the headings in the document text, this way: Main heading begins .H1, then a space, then the heading Second line, if required, begins .H2, then space, then the text.

The program will center the heading(s) and put them at the top of every page.

In addition, you can have page numbering and dating. Date must be specified when you run off the print. Page numbering can be done then, too, or can be put into the document.

At run-off time, you just give a number at the prompt that asks "First page number?". If you press ENTER instead, then the page-numbering in the text is used (if any) — if none, page numbers are not printed.

Page numbers in the text are specified as a line with the control code .PG nnn (nnn being the page number to take effect at that point).

Remember that .PG without a following number doesn't affect page numbers: it just starts a new page.

CONTROL CODES

To get condensed, emphasized, or double-strike print (using Epson control characters) you use these codes. *Note*: they act as toggle switches, i.e. the first occurrence sets the option on, the next turns it off, and so on.

.EM	Emphasized
.EN	Enlarged
.CO	Condensed
.DP	Double Print
.DS	Double-Space

You can have combinations of these in effect at one time. The effects can get a bit complicated, especially with page headings, so do some experimenting.

One point: to make all these work, the program does depend on the printer not inserting auto linefeeds. (It puts a linefeed at the end of each line). So be sure your printer spec is of the style "RS232.BA=9600.CR" The speed is optional, but the .CR is required. Also, keep your text lines less than 78 characters long, to allow for the generated "CR" at the end and possible control characters at the beginning.

Disk 48. Disk Manager 99

Version: 2.1 *Author*: Mike Dodd
Requires: XB *Language*: AL *Updated*: 01/21/87

A set of assembly language routines callable from XB or TI BASIC which give you access to most features of the Disk Manager cartridge. Complete source code included.

```
dskdir. v2.0. 12-dec-96
Disk name               = DM99-V2/1
Sectors total           = 360
Sectors used            = 354
Sectors available       = 4
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >017 -README       5 DIS/VAR 80   >018 004
002 >007 DM99-1      103 DIS/VAR 80 Y >0d8 102
003 >008 DM99-2       57 DIS/VAR 80 Y >13e 042 >009 014
004 >006 DM99/DOC     54 DIS/VAR 80 Y >0a3 053
005 >002 LOAD         28 PROGRAM    Y >020 027
006 >004 PRINTDOCS     4 PROGRAM    Y >08e 003
007 >005 XBDM99       19 DIS/VAR 80 Y >091 018
008 >003 XBDM99/O     84 DIS/FIX 80 Y >03b 083
```

Disk 48. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain Software Disk #48

This disk contains a set of assembly routines that allow you to use most of the features of a Disk Manager through CALL LINK statements in TI BASIC or Extended BASIC.

Documentation may be found in the file DM99/DOC and printed by PRINTDOCS.

This is a very impressive piece of work (especially if you go through the source code, the author makes use of next to none of the built in GPL routines for disk access) and is very well documented and commented. I would recommend sending the author a donation of at least $10 (he doesn't specify an amount) if you find it useful. Thank you.

J. Peter Hoddie

Disk 48. Contents of file DM99-1

```
* DISK MANAGER 99 PART 1. BY MIKE DODD
* THIS IS A FAIRWARE PROGRAM. IF YOU LIKE THIS PROGRAM,
* PLEASE SEND WHATEVER YOU THINK IT IS WORTH TO:
* MIKE DODD, 116 RICHARDS DRIVE, OLIVER SPRINGS
* TENNESSEE 37840 USA (615)435-1667
* VERSION 2.1
*
* IMPORTANT---SAVE AS DM99-1
VDPRD  EQU  >8800
VDPWA  EQU  >8C02
VDPWD  EQU  >8C00
KEYNUM EQU  >8374        KEYBOARD NUMBER
KEY    EQU  >8375        KEY CODE
PNTR   EQU  >8356        POINTER FOR DSRLNK
FAC    EQU  >834A
PAB1   EQU  >1E02        SPACE FOR PRINTER PAB
VDPBUF EQU  >1D02        SPACE FOR VDP BUFFER
SECLEN EQU  >100         LENGTH OF ONE SECTOR
GPLWS  EQU  >83E0        ADDRESS OF GPL WORKSPACE
STATUS EQU  >837C        ADDRESS OF GPL STATUS BYTE
NEXT   EQU  >0070        ADDRESS OF GPL NEXT ADDRESS
SCREEN BYTE 0            >FF IF SCREEN ONLY
STRLEN BYTE 0            STRING BUFFER LENGTH BYTE
STRBUF BSS  10           SPACE FOR STRING
* DBUF1 IS USED TO STORE SECTOR 1 FOR FIND SUBROUTINE
DBUF1  BSS  SECLEN
       DATA 0            TO INDICATE NO MORE FILES
* SECTOR 0 BUFFER FOR INITIALIZE DISK ROUTINE
INTBUF BSS  13           SPACE FOR NAME AND #SECTORS
       TEXT 'DSK '
       BSS  3            SPACE FOR#TRACKS,SIDE,DENSITY
       DATA 0,0,0,0,0,0,0,0,0 *9 0'S
       DATA 0,0,0,0,0,0,0,0,0 *9 0'S
       BSS  218          SPACE FOR SECTOR BITMAP
SUBWS  BSS  >20          SUBROUTINE WORKSPACE
SUBWS1 BSS  >20          SUBROUTINE WORKSPACE
CATBUF BSS  28           SPACE FOR DATA IN CAT ROUTINE
SCRBUF BSS  >20          SCREEN BUFFER
SAVBUF BSS  316          SPACE TO SAVE USED VDP MEMORY
MYWS   BSS  >20          MAIN WORKSPACES
DELPAB DATA >0700,PAB+26,0,0
DELLEN DATA 0
       TEXT 'DSK'
DELDRV BYTE 0
       TEXT '.'
DELNAM BSS  15
       EVEN
DAT0   DATA 0
```

```
GROMSV DATA >0000        SPACE TO STORE GROM ADDRESS
FLETAB DATA DF,DV,IF,IV  TABLE FOR FILE TYPES
SECSUB DATA >0110        SECTOR READ/WRITE SUB
INITSB DATA >0111        INITIALIZE DISK SUB
PROTSB DATA >0112        CHANGE FILE PROTECTION SUB
RENSUB DATA >0113        RENAM FILE SUB
* THE FOLLOWING 4 LINES ARE USED BY THE CATALOG ROUTINE
HEAD0  TEXT 'DSKX-Diskname='
HEAD1  TEXT 'Free=     Used=     Numb=       '
HEAD2  TEXT ' Filename  Size    Type   P     '
HEAD3  TEXT '---------- ---- --------- -     '
DSKNAM TEXT 'New diskname?'
FILET  TEXT 'Filename?'
DRIVET TEXT 'Drive#?'
* THE FOLLOWING 5 LINES INDICATE THE TYPE OF FILE
DF     TEXT 'Ds/Fx'
IF     TEXT 'In/Fx'
DV     TEXT 'Ds/Vr'
IV     TEXT 'In/Vr'
PROG   TEXT 'Program'
TOTERR TEXT 'Total errors: XXXX'   TOTAL#OF ERRORS
TESTXT TEXT 'Reading sector: XXXX' READING SECTOR
ERRTXT TEXT 'Bad sector: XXXX'     BAD SECTOR
DNAMT  TEXT 'Disk name?' 10 CHARS
* NEXT 3 ARE 8 CHARS EACH
SIDT   TEXT '#Sides? '
DENT   TEXT 'Density?'
VERT   TEXT 'Verify? '
* THE FOLLOWING 10 LINES ARE PRINTED WHEN THE PROGRAM
*  IS FIRST LOADED
TITLET TEXT '  DISK MANAGER 99. FAIRWARE by  '
       TEXT '  Mike Dodd. Version 2.1 Date:  '
       TEXT '  86.1202 Copr. 1985, 1986  by  '
       TEXT '  Mike Dodd.  Please copy this  '
       TEXT '  program for your friends. If  '
       TEXT '  you use this program, please  '
       TEXT '  send  whatever you thing  it  '
       TEXT '  is worth to: Mike Dodd,  116  '
       TEXT '  Richards Dr,  Oliver Springs  '
       TEXT '  TENN 37840 USA  615/435-1667  '
       TEXT '                                '
       TEXT '  Press <CTRL>D to pull up the  '
       TEXT '  main menu.  Please  send any  '
       TEXT '  comments,  good  or bad,  to  '
       TEXT '  the above address. Thank you  '
       TEXT '                                '
       TEXT '  For  complete  instructions,  '
       TEXT '  type RUN "DSK1.PRINTDOCS"     '
KEYN   TEXT 'N'
KEYD   TEXT 'D'
KEYU   TEXT 'U'
```

```
KEYP   TEXT 'P'
NUM1   TEXT '1'
NUM9   TEXT '9'
H2E    BYTE >2E          PERIOD
BACKEY BYTE >0F          FCTN 9 KEY
H05    BYTE >05
H02    BYTE >02          FCTN 4 KEY
UARROW BYTE >0B          UP ARROW ASCII
H08    BYTE >08
OUTPUT BYTE 0            OUTPUT CODE
INPUT  BYTE 1            INPUT CODE
HGRT   BYTE >80          LOGICAL GREATER THAN MASK
HEQ    BYTE >20          EQUAL BIT MASK
EOFVAL BYTE >05          CODE FOR EOF ERROR
HFF    BYTE >FF
DRIVE  BYTE 0            USED TO INDICATE DRIVE#
       EVEN              MAKE SURE PC IS EVEN
PDATA1 BYTE >00          OPERATION (OPEN)
       BYTE >12          FILE TYPE (OUTPUT, DIS/VAR)
       DATA VDPBUF       ADDRESS OF VDP BUFFER
       BYTE >00          RECORD LENGTH-DSR DETERMINED
       BYTE >00          CHARACTER COUNT
       DATA >00          RELATIVE RECORDS NUMBER (N/A)
       BYTE >00          SCREEN OFFSET (N/A)
       BYTE >00          LENGTH OF FILE NAME
       BSS  >20          SPACE FOR  "    "
       EVEN
* START UP ROUTINE THAT PRINTS THE TITLE MESSAGE
PTITLE CLR  @>83C4       CLEAR ISR HOOKUP
       LWPI MYWS         LOAD MAIN WORKSPACE
       BL   @CLS
       LI   R0,>00A0     R5,C0
       LI   R1,TITLET    TITLE SCREEN TEXT
       LI   R2,>240      18 LINES
       BLWP @PBASIC      WRITE WITH BASIC OFFSET
       LI   R0,CHKKEY
       MOV  R0,@>83C4
       LWPI GPLWS        GPL WORKSPACE
       RT                RETURN
* SCREEN INPUT SUBROUTINE. INPUT: R0=SCREEN ADDRESS,
*  R1=CPU ADDRESS, R2=MAXIMUM LENGTH.
*  OUTPUT: R2=ACTUAL LENGTH.
*  REQUIRES SUBWS1 BSS >20
*  ACCESS WITH A BLWP @INPUT
ANYKEY
SPACE  BYTE >20
LARROW BYTE >08
RARROW BYTE >09
ERASE  BYTE >07
ENTER  BYTE >0D
```

```
CURSOR BYTE >7E
A127   BYTE 127
       EVEN
RESCHR CB   R1,@CURSOR
       JNE  RST1
       SWPB R1
       BLWP @VSBW
RST1   SWPB R1
       LI   R9,>100
       RT
INPUT0 DATA SUBWS1,INPUT1
INPUT1 MOV  *R13,R4      R4=START SCREEN
       SZCB @HEQ,R15     ZERO EQ BIT
       MOV  @4(R13),R5   R5=LEN
       MOV  @2(R13),R6   R6=ADDR TO PUT DATA
       A    R4,R5        R5=LAST SCREEN ADDR.
       DEC  R5           CORRECT
       MOV  R4,R0        COPY R4 INTO R0
       MOV  R6,R7
       MOVB @H05,@KEYNUM
       LI   R9,>0100
       CLR  @STATUS
* REGISTER LAYOUT:
*  R0=CURRENT VDP
*  R1=CURSOR & SCREEN CHAR
*  R2=KEY PRESSED
*  R3=N/U
*  R4=START VDP
*  R5=MAXIMUM LENGTH
*  R6=START CPU
*  R7=CURRENT CPU
*  R8=N/U
*  R9=COUNTER FOR CURSOR FLASH
*  R10=N/U
*  R11=RETURN ADDRESS FOR BL'S
*  R12=N/U
*  R13,R14,R15=RTWP VECTORS
INPUT2 BLWP @VSBR
INPUT3 SWPB R1
       MOVB @CURSOR,R1
       BLWP @VSBW
INPUT4 DEC  R9
       JNE  INPUT5
       SWPB R1
       BLWP @VSBW
       LI   R9,>100
INPUT5 BLWP @KSCAN
       CB   @ANYKEY,@STATUS
       JNE  INPUT4
       MOVB @KEY,R2
       CB   R2,@BACKEY
```

```
       JEQ  INPUTB
       CB   R2,@ENTER
       JNE  INPUT6
       BL   @RESCHR
       MOV  R5,R0         LAST VDP ADDR
       INC  R0
FNDBLK DEC  R0
       C    R0,R4
       JL   INPT5A
       CLR  R1
       BLWP @VSBR
       CI   R1,>8000
       JEQ  FNDBLK
INPT5A S    R4,R0
       INC  R0
       MOV  R0,R2
       MOV  R0,R9
       MOV  @4(R13),R0   MAX LENGTH
       S    R2,R0        R0=R0-R2
       JEQ  INPUTR
       MOV  R6,R1
       A    R2,R1
       LI   R2,>2000
CLRNAM MOVB R2,*R1+
       DEC  R0
       JNE  CLRNAM
       MOV  R9,@4(R13)
INPUTR RTWP
INPUT6 CB   R2,@RARROW
       JNE  INPUT7
       C    R0,R5
       JEQ  INPUT4
       BL   @RESCHR
       INC  R0
       INC  R7
       JMP  INPUT2
INPUT7 CB   R2,@LARROW
       JNE  INPUT8
       C    R0,R4
       JEQ  INPUT4
       BL   @RESCHR
       DEC  R0
       DEC  R7
       JMP  INPUT2
INPUT8 CB   R2,@ERASE
       JNE  INPUTA
       MOV  R4,R0
       LI   R1,>8000
       BLWP @VSBW
       MOV  R0,R2
```

```
INPUT9 MOVB R1,@VDPWD
       INC  R2
       C    R2,R5
       JNE  INPUT9
       MOV  R6,R7
       JMP  INPUT2
INPUTA CB   R2,@SPACE
       JL   INPUT5
       CB   R2,@A127
       JH   INPUT5
       LI   R1,>001E
       MOVB R2,R1
       AI   R1,>6060
       BLWP @VSBW
       MOVB R2,*R7
       MOVB @RARROW,R2
       JMP  INPUT6
* ABORT DM99
INPUTB SOCB @HEQ,R15     SET EQ BIT
       RTWP
* PRINT WITH BASIC OFFSET. R0=VDP ADDRESS. R1=CPU ADDR.
* R2=COUNT
PBASIC DATA SUBWS1,PBAS1
PBAS1  MOV  *R13,R0      R0=VDP SCREEN ADDRESS
       MOV  @2(R13),R5   R5=CPU ADDRESS
       MOV  @4(R13),R6   R6=CHARACTER COUNT
       BL   @VDPWRT
       LI   R2,>6000     MSBY OF R2=CHAR. OFFSET (>60)
       LI   R0,VDPWD     R0=ADDR. OF VDP WRITE DATA ADD
PBAS2  MOVB *R5+,R1      MOVE NEXT CHAR. TO MSBY OF R1
       AB   R2,R1        ADD OFFSET
       MOVB R1,*R0       VDP WRITE DATA ADDRESS
       DEC  R6           DECREMENT CHAR. COUNT
       JNE  PBAS2        IF <> 0, THEN GOTO PBAS2
       RTWP              RETURN
* CONVERT NUMBER TO STRING ROUTINE. INPUT:R0=INTEGER
* NUMBER TO CONVERT. R1=CPU LOCATION TO PUT STRING
CNUM   DATA 1000,100,10,1     PLACE NUMBERS
CNS    DATA SUBWS,CNS1
CNS1   MOV  *R13,R5      R5=NUMBER TO CONVERT
       MOV  @2(R13),R1   R1=SPACE TO PUT STRING
       CLR  R6           R6=0
* R7 INDICATES WHETHER OR NOT A NUMBER OTHER THAN 0 HAS
* BEEN FOUND
       CLR  R7           R7=0.
CNS2   CLR  R4           R4=0
       DIV  @CNUM(R6),R4 R4 AND R5=R4 AND R5 / CNUM(R6)
       CI   R6,6         IF AT LAST DIGIT THEN PRINT,
       JEQ  CNS4              EVEN IF A ZERO
       MOV  R7,R7        HAVE WE FOUND A NON-0 YET?
       JNE  CNS4         YES. GOTO CNS4
```

```
       MOV  R4,R4        SEE IF # = 0
       JEQ  CNS3         YES. GOTO CNS3
       SETO R7           SET R7 TO >FFFF
       JMP  CNS4         GOTO CNS4
CNS3   LI   R4,>20       SPACE CHARACTER
       JMP  CNS5
CNS4   AI   R4,>30       CONVERT TO ASCII
CNS5   SWPB R4           PUT IN MSBY
       MOVB R4,*R1+      MOVE TO CPU LOCATION
       INCT R6
       CI   R6,8         ARE WE DONE YET?
       JNE  CNS2         IF NOT, THEN GOTO CNS2
       RTWP              RETURN
* WRITES A LINE TO THE SELECTED OUTPUT DEVICE. INPUT:
* R0=CPU LOCATION OF MESSAGE. R1=LENGTH. IF BYTE AT
* SCREEN=0 THE WILL PRINT TO PRINTER AND SCREEN, ELSE
* JUST SCREEN. PRINTS AT R22,C2. SCROLL DOES NOT SCROLL
* BOTTOM LINE
WRTLIN DATA SUBWS,WRLIN0
WRLIN0 MOVB @SCREEN,R0   TO SCREEN ONLY?
       JNE  WRTSCR       YES, GOTO WRTSCR
       LI   R0,VDPBUF    R0=VDPBUF
       MOV  *R13,R1      R1=LOCATION OF LINE TO PRINT
       MOV  @2(R13),R2   R2=LENGTH
       BLWP @VMBW        WRITE TO VDP
       MOVB @3(R13),R1   MOVE LENGTH TO MSBY R1
       LI   R0,PAB1+5    R0=LENGTH BYTE OF PRINTER PAB
       BLWP @VSBW        WRITE TO VDP
       LI   R0,PAB1+9
       MOV  R0,@PNTR
       BLWP @DSRLNK      GOTO DSR ROUTINE
       DATA 8            CODE FOR NORMAL FILE I/O
       JNE  WRTSCR       IF NO ERROR, THEN GOTO WRTSCR
       B    @IOERR       GOTO ERROR ROUTINE
WRTSCR LI   R0,>2C2      R0=R22,C2
       MOV  *R13,R1      R1=LOCATION OF LINE TO PRINT
       MOV  @2(R13),R2   R2=LENGTH
       BLWP @PBASIC      PRINT WITH BASIC OFFSET
       BL   @SCROLL      SCROLL SCREEN
       RTWP              RETURN
* FIND SUBROUTINE. INTPUT:STRBUF=STRING TO SEARCH FOR.
* OUTPUT:R0=SECTOR# R1=POSITION IN SECTOR 1 RELATIVE TO
* DBUF1. IF NOT FOUND, EQUAL BIT WILL BE SET (JEQ)
FIND   DATA SUBWS,FIND0
FIND0  SZCB @HEQ,R15     ZERO EQUAL BIT
       LI   R0,1
       BL   @READ        READ SECTOR 1
       LI   R0,VDPBUF
       LI   R1,DBUF1
       LI   R2,SECLEN
```

```
       BLWP @VMBR        READ SECTOR INTO DBUF1
       MOV  R1,R9        R9=DBUF1
       CLR  R10          R10=0
FIND1  MOV  *R9+,R0      MOVE FILE LOC. TO R0
       JEQ  NOFILE       IF 0, THEN GOTO NOFILE
       MOV  R0,R7        SAVE SECTOR#
       BL   @READ        READ SECTOR
       BLWP @FCOMP       COMPARE FILE NAMES
       JNE  FIND2        NOT EQUAL. GOTO FIND2
       MOV  R7,*R13      EQUAL. R0=SECTOR#
       MOV  R10,@2(R13)  R1=LOCATION RELATIVE TO DBUF1
       RTWP              RETURN
FIND2  INCT R10          R10=R10+2
       JMP  FIND1        GOTO FIND1
NOFILE SOCB @HEQ,R15     FILE NOT FOUND. SET EQUAL BIT
       RTWP              RETURN
* FILE NAME COMPARE ROUTINE. INPUT:STRBUF=STR1. VDPBUF=
* STR2. OUTPUT:JH IF STR2>STR1. JL IF STR2<STR1. JEQ IF
* STR2=STR1
FCOMP  DATA SUBWS1,FCOMP0
FCOMP0 CLR  R15          ZERO ALL STATUS BITS
       LI   R0,VDPBUF
       BL   @VDPRED      SET UP VDP READ ADDRESS
       LI   R0,STRBUF
       LI   R1,VDPRD
       LI   R2,10        R2=LENGTH OF FILE NAME
FCOMP1 CB   *R0+,*R1     COMPARE BYTES
       JL   FCL          IF R0<R1 THEN GOTO FCL
       JH   FCH          IF R0>R1 THEN GOTO FCH
       DEC  R2           LENGTH=LENGTH-1
       JNE  FCOMP1       IF NOT 0, THEN GOTO FCOMP1
       SOCB @HEQ,R15     SET EQUAL BIT
FCL    RTWP              RETURN
FCH    SOCB @HGRT,R15    SET LOGICAL GREATER THAN BIT
       RTWP              RETURN
* GET DISK DRIVE NUMBER
GETDRV MOV  R11,R12
       BL   @CLS
       LI   R0,2
       LI   R1,DRIVET
       LI   R2,7
       BLWP @PBASIC
       LI   R0,10
       BL   @GETONE
       TEXT '123456789'
       BYTE 0
       EVEN
       SWPB R1
       MOVB R1,@DRIVE
       B    *R12
* FILE COMMANDS
```

```
FIL    LWPI MYWS
       BL   @SETUP
FILLNK BL   @GETDRV
       LI   R0,>22
       LI   R1,FILET
       LI   R2,9
       BLWP @PBASIC
       LI   R0,>2D
       LI   R1,STRBUF
       LI   R2,10
       BLWP @INPUT0
       JNE  FIL1
FIL01  B    @RETURN      ABORT
FIL1   BL   @LEGAL       IS IT A LEGAL FILENAME?
       BLWP @FIND
       JNE  FIL2
       B    @IOERR
FIL2   MOV  R0,@DAT0
       LI   R0,VDPBUF
       LI   R1,CATBUF
       LI   R2,28
       BLWP @VMBR
       BL   @FILE
       LI   R0,>A4
       LI   R1,SCRBUF
       LI   R2,27
       BLWP @PBASIC
       LI   R0,>64
       LI   R1,HEAD2
       BLWP @PBASIC
       DECT R0
       LI   R1,>A300     "C"
       BLWP @VSBW
       LI   R0,>84
       LI   R1,HEAD3
       BLWP @PBASIC
       DECT R0
       LI   R1,>8D00     "-"
       BLWP @VSBW
       LI   R0,>A2
       LI   R1,>AE00     "N"
       BLWP @VSBW
       LI   R1,>7EAE     CURSOR&"N"
       CLR  R2           FIELD
       CLR  R3
       CLR  @KEYNUM
       CLR  @STATUS
* COMMAND - >A2       (C2)
* FNAM - >A4 THRU >AD (C4-C13)
* PROT - >BE          (C30)
```

```
FIL3   BL   @RESCHR
       A    R3,R0
       BLWP @VSBR
       SRL  R1,8
       MOVB @CURSOR,R1
       BLWP @VSBW
FIL5   DEC  R9
       JNE  FIL5X
       SWPB R1
       BLWP @VSBW
       LI   R9,>100
FIL5X  BLWP @KSCAN
       CB   @STATUS,@ANYKEY
       JNE  FIL5
       CLR  R3
       MOVB @KEY,R3
       CB   R3,@BACKEY
       JEQ  FIL01
       CB   R3,@RARROW
       JNE  FIL6
FILRI  MOV  R2,R2
       JNE  FIL5A
       LI   R3,2
       INC  R2
       JMP  FIL3
FIL5A  CI   R2,2
       JNE  FIL5B
       LI   R3,-28
       DECT R2
       JMP  FIL3
* THEN WE ARE IN FNAM FIELD
FIL5B  CI   R0,>AD
       JNE  FIL5C
       LI   R3,17
       INC  R2
       JMP  FIL3
FIL5C  LI   R3,1
       JMP  FIL3
FIL6   CB   R3,@LARROW
       JNE  FIL7
       MOV  R2,R2
       JNE  FIL6A
       LI   R3,28
       INCT R2
       JMP  FIL3
FIL6A  CI   R2,2
       JNE  FIL6B
       DEC  R2
       LI   R3,-17
       JMP  FIL3
* THEN WE ARE IN FNAM FIELD
```

```
FIL6B  CI   R0,>A4
       JNE  FIL6C
       DEC  R2
       LI   R3,-2
       JMP  FIL3
FIL6C  LI   R3,-1
       JMP  FIL3
FIL7   CB   R3,@ENTER
       JEQ  FIL9
       CB   R3,@A127
       JH   FIL5
       CB   R3,@SPACE
       JL   FIL5
       MOV  R2,R2
       JNE  FIL7A
       CB   R3,@KEYN
       JEQ  FIL8
       CB   R3,@KEYD
       JEQ  FIL8
       JMP  FIL5
FIL7A  CI   R2,2
       JNE  FIL7B
       CB   R3,@KEYU
       JEQ  FIL8
       CB   R3,@KEYP
       JEQ  FIL8
       JMP  FIL5
FIL7B  CB   R3,@SPACE
       JL   FIL5
       CB   R3,@A127
       JH   FIL5
FIL8   MOV  R3,R1
       AI   R1,>6000
       SWPB R1
       MOVB @CURSOR,R1
       JMP  FILRI
* PROCESS COMMANDS
FIL9   BL   @RESCHR
       LI   R0,>A4
       LI   R1,STRBUF
       LI   R2,10
       BLWP @VMBR
* FIX FOR BASIC OFFSET
       LI   R0,>6060
       S    R0,@STRBUF+0
       S    R0,@STRBUF+2
       S    R0,@STRBUF+4
       S    R0,@STRBUF+6
       S    R0,@STRBUF+8
       BL   @LEGAL       IS IT A LEGAL FILENAME?
```

```
* DELETE FILE?
       LI   R0,>A2
       BLWP @VSBR
       AI   R1,->6000
       CB   R1,@KEYD
       JNE  FIL9X
       LI   R0,>BE       PROTECTION
       BLWP @VSBR
       AI   R1,->6000
       CB   R1,@KEYP     IF PROTECTED,
       JEQ  DEL3          THEN GIVE ERROR
* SCREEN SAYS "U"NPROTECTED
       CB   R1,@SCRBUF+26     SEE IF FILE UNPROTECTED
       JEQ  DEL0         YUP
       LI   R0,VDPBUF+12 TYPE BYTE
       BLWP @VSBR
       SZCB @H08,R1
       BLWP @VSBW
       MOV  @DAT0,R0     SECTOR#
       BL   @WRITE       WRITE TO DISK
* DELETE FILE
DEL0   LI   R0,VDPBUF
       LI   R1,DELNAM
       LI   R2,10
       BLWP @VMBR
       MOVB @DRIVE,R0
       AI   R0,>3000
       MOVB R0,@DELDRV
       LI   R3,5
DEL1   CB   *R1+,@SPACE
       JEQ  DEL2
       INC  R3
       DEC  R2
       JNE  DEL1
DEL2   MOV  R3,@DELLEN
       LI   R0,PAB
       LI   R1,DELPAB
       LI   R2,25
       BLWP @VMBW
       AI   R0,9
       MOV  R0,@PNTR
       BLWP @DSRLNK
       DATA 8
       JNE  DEL4
DEL3   B    @IOERR
DEL4   B    @RETURN
* READ NEW FNAM
FIL9X  LI   R3,VDPBUF
       BLWP @FCOMP       COMPARE FNAMS
       JEQ  FILA         IF SAME THEN FILA
* RENAME FILE
```

```
*  WRITE NEW FNAM TO VDP BUFFER
FIL9A  LI   R0,VDPBUF+10
       MOV  R0,R3
       LI   R1,STRBUF
       LI   R2,10
       BLWP @VMBW
       MOV  R0,@FAC+4
       LI   R0,VDPBUF
       MOV  R0,@FAC+6
       LI   R0,PAB
       LI   R1,RENSUB
       LI   R2,2
       BLWP @VMBW
       BL   @GODSR
FILA   LI   R0,>BE
       BLWP @VSBR
       AI   R1,->6000
       CB   @SCRBUF+26,R1
       JEQ  FILB
       MOV  R3,@FAC+4
       CB   R1,@KEYU
       JEQ  UNPROT
       SETO @FAC+2
       JMP  PROTEC
UNPROT CLR  @FAC+2
PROTEC MOVB @DRIVE,@FAC+2
       LI   R0,PAB
       LI   R1,PROTSB
       LI   R2,2
       BLWP @VMBW
       BL   @GODSR
FILB   B    @RETURN
* DISK RENAME
DRENAM LWPI MYWS
       BL   @SETUP       SETUP MEMORY
DRELNK BL   @GETDRV
       LI   R0,>22
       LI   R1,DSKNAM
       LI   R2,13
       BLWP @PBASIC
       LI   R0,>30
       LI   R1,STRBUF
       LI   R2,10
       BLWP @INPUT0
       JEQ  DRENM1
       CLR  R0
       BL   @READ        READ SECTOR 0
       LI   R0,VDPBUF
       LI   R1,STRBUF
       LI   R2,10
```

```
       BLWP @VMBW        WRITE DISK NAME TO VPBUF
       CLR  R0
       BL   @WRITE       WRITE TO SECTOR 0
DRENM1 B    @RETURN
* DISK TEST
TEST   LWPI MYWS
       BL   @SETUP       SET UP MEMORY
TSTLNK BL   @DISPRT
       BL   @GETDRV
* GET TOTAL # SECTORS IN R10
       CLR  R0
       BL   @READ        READ SECTOR 0
       LI   R0,VDPBUF+10
       LI   R1,MYWS+20   R10
       LI   R2,2
       BLWP @VMBR
       BL   @OPNPAB
       CLR  R9           CURRENT SECTOR
       CLR  R15          #BAD SECTORS
       MOVB @INPUT,R13
       SWPB R13
       MOVB @DRIVE,R13
       LI   R14,VDPBUF
       LI   R12,PAB
TEST1  MOV  R9,R0        R0=SECTOR#
       LI   R1,TESTXT+16
       BLWP @CNS
       LI   R0,>2E2
       LI   R1,TESTXT
       LI   R2,20
       BLWP @PBASIC
       MOV  R9,@FAC+6    SECTOR#
       MOV  R12,@PNTR    PAB ADDR
       MOV  R13,@FAC+2   DRIVE&INPUT CODE
       MOV  R14,@FAC+4   VDP BUFFER
       BLWP @DSRLNK
       DATA >A           SUBROUTINE
       MOVB @FAC+6,R0    ERROR CODE
       JEQ  TEST2        NO ERROR
       INC  R15          # BAD SECTORS
       MOV  R9,R0
       LI   R1,ERRTXT+12
       BLWP @CNS
       LI   R0,ERRTXT
       LI   R1,16
       BLWP @WRTLIN      WRITE TO SCREEN/PRINTER
TEST2  INC  R9           SECTOR#
       C    R9,R10       AT END?
       JEQ  TEST4        NO
       CLR  @KEYNUM
       BLWP @KSCAN
```

```
       CB   @KEY,@H02    FCTN 4?
       JNE  TEST1        NO. KEEP READING SECTORS
       MOVB @SCREEN,R0
       JNE  TEST4
TEST3  BLWP @KSCAN
       CB   @KEY,@HFF
       JNE  TEST3
TEST4  MOV  R15,R0       # BAD SECTORS
       LI   R1,TOTERR+14
       BLWP @CNS
       LI   R0,TOTERR
       LI   R1,18
       BLWP @WRTLIN      WRITE TO SCREEN/PRINTER
       BL   @CLSPRN      CLOSE PRINTER?
       B    @RETURN      RETURN TO BASIC
* DISK CATALOG
CAT    LWPI MYWS
       BL   @SETUP       SET UP MEMORY
CATLNK BL   @DISPRT
       BL   @GETDRV
       MOVB @DRIVE,R0
       AI   R0,>3000     CONVERT TO ASCII
       MOVB R0,@HEAD0+3
       BL   @OPNPAB      OPEN PRINTER?
       LI   R0,1
       BL   @READ
       LI   R0,VDPBUF
       BL   @VDPRED
       CLR  R0
       LI   R1,127
CAT1   MOVB @VDPRD,R3
       SWPB R3
       MOVB @VDPRD,R3
       MOV  R3,R3
       JEQ  CAT2
       INC  R0
       DEC  R1
       JNE  CAT1
CAT2   LI   R1,HEAD1+25
       BLWP @CNS
       BL   @HEAD
       LI   R0,1
       BL   @READ        SECTOR 1
       LI   R0,VDPBUF
       LI   R1,DBUF1
       LI   R2,SECLEN
       BLWP @VMBR
       BL   @FILES
       BL   @CLSPRN      CLOSE PRINTER?
CATR   B    @RETURN      RETURN TO BASIC
```

```
* INITIALIZE DISK
INIT   LWPI MYWS
       BL   @SETUP       SET UP MEMORY
INTLNK
INIT1  BL   @GETDRV
       LI   R0,>22
       LI   R1,DNAMT
       LI   R2,10
       BLWP @PBASIC
       LI   R0,>2D
       LI   R1,INTBUF
       LI   R2,10
       BLWP @INPUT0
       JEQ  CATR
       LI   R0,>42
       LI   R1,SIDT
       LI   R2,8
       LI   R3,3
INIT2  BLWP @PBASIC
       AI   R0,>20
       A    R2,R1
       DEC  R3
       JNE  INIT2
INIT3  LI   R0,>4B
       BL   @GETONE
       TEXT '12'
       BYTE 0
       EVEN
       CB   R0,@UARROW
       JEQ  INIT1
       MOV  R1,R8
INIT4  LI   R0,>6B
       BL   @GETONE
       TEXT 'SD'
       BYTE 0
       EVEN
       CB   R0,@UARROW
       JEQ  INIT3
       SWPB R1
       MOV  R1,R9
       LI   R0,>8B
       BL   @GETONE
       TEXT 'YN'
       BYTE 0
       EVEN
       CB   R0,@UARROW
       JEQ  INIT4
       CLR  R15
       CI   R1,1
       JNE  INIT6
       LI   R15,>0156
```

```
INIT6  LI   R10,40       40 TRACKS
       MOV  R10,R0
       MOVB @DRIVE,R0
       MOV  R0,@FAC+2
       LI   R0,DNTBUF
       MOV  @>831A,R1    START OF BASIC'S USE
       CI   R9,>0200     DOUBLE DENSITY?
       JEQ  DENS2        YES
       LI   R2,3300
       JMP  SPACCH
DENS2  LI   R2,6700
SPACCH S    R0,R1
       C    R1,R2
       JH   SPACOK       ENOUGH SPACE
       LI   R0,ERRMF     MEMORY FULL ERROR
       B    @IOERR1
SPACOK MOV  R0,@FAC+4
       MOVB R9,R8
       MOV  R8,@FAC+6
       LI   R0,PAB
       LI   R1,INITSB
       LI   R2,2
       BLWP @VMBW
       MOV  R0,@PNTR
       BLWP @DSRLNK
       DATA >A           SUBROUTINE
       MOVB @FAC+6,R0    ERROR?
       JEQ  WRTS0        NO
       B    @IOERR       I/O ERROR
WRTS0  SWPB R8
       MOV  R8,@INTBUF+18
       MOV  @FAC,R1      TOTAL # SECTORS
       MOV  R1,@INTBUF+10
       MOV  R1,R12
       SWPB R0
       LI   R2,>0900     SECTORS/TRACK IN SIGLE DENSITY
       SLA  R2,0
       SRL  R2,1
       MOVB R2,@INTBUF+12
       MOV  R10,R0       #TRACKS
       SWPB R0
       MOVB R0,@INTBUF+17
       SRL  R1,3         /8
       LI   R0,INTBUF+56
       LI   R3,INTBUF+SECLEN
       CLR  R2
* WRITE >00'S TO SECTOR BITMAP
WRT00  MOVB R2,*R0+
       DEC  R1
       JNE  WRT00
```

```
       SETO R2
* WRITE >FF'S TO SECTOR BITMAP
WRTFF  MOVB R2,*R0+
       C    R0,R3
       JNE  WRTFF
       LI   R0,>0300
       MOVB R0,@INTBUF+56
       LI   R0,PAB
       LI   R1,SECSUB
       LI   R2,2
       BLWP @VMBW
       LI   R0,VDPBUF
       LI   R1,INTBUF
       LI   R2,SECLEN
       BLWP @VMBW
       CLR  R0
       BL   @WRITE       WRITE TO SECTOR 0
       LI   R0,VDPBUF
       BL   @VDPWRT
       LI   R0,SECLEN
* WRITE 0'S
CLRS1  CLR  @VDPWD
       DEC  R0
       JNE  CLRS1
       INC  R0           MAKE IT A 1
       BL   @WRITE       WRITE TO SECTOR 1
       CI   R15,>0156    VERIFY? (1 CHAR - "V")
       JEQ  VERIFY       YES
       B    @RETURN      RETURN TO BASIC
VERIFY CLR  R0
       BL   @READ        READ SECTOR 0
       LI   R0,VDPBUF
       LI   R1,INTBUF
       LI   R2,SECLEN
       BLWP @VMBR
       LI   R13,PAB
       MOVB @INPUT,R0
       SWPB R0
       MOVB @DRIVE,R0
       MOV  R0,@FAC+3
       LI   R0,VDPBUF
       MOV  R0,@FAC+4
       LI   R15,2
INTCHK MOV  R15,R0
       LI   R1,TESTXT+16
       BLWP @CNS
       LI   R0,>2E2
       LI   R1,TESTXT
       LI   R2,20
       BLWP @PBASIC
       MOV  R15,@FAC+6
```

```
       MOV  R13,@PNTR
       BLWP @DSRLNK
       DATA >A
       MOVB @FAC+6,R0
       JEQ  SECOK
       MOV  R15,R5
       SRL  R5,3
       MOV  R15,R0
       ANDI R0,7
       LI   R1,>80
       INC  R0
       SLA  R1,0
       SOCB R1,@INTBUF+56(R5)
SECOK  INC  R15
       C    R15,R12
       JNE  INTCHK
       LI   R0,VDPBUF
       LI   R1,INTBUF
       LI   R2,SECLEN
       BLWP @VMBW
       CLR  R0
       BL   @WRITE
       B    @RETURN
* NEXT FILE IS DM99-2
```

Disk 48. Contents of file DM99-2

```
* DISK MANAGER 99 PART 2. BY MIKE DODD, K-TOWN 99'ERS
* THIS IS A FAIRWARE PROGRAM. IF YOU LIKE THIS PROGRAM,
* PLEASE SEND WHATEVER YOU THINK IT IS WORTH TO:
* MIKE DODD, 116 RICHARDS DRIVE, OLIVER SPRINGS
* TENNESSEE 37840 USA (615)435-1667
* VERSION 2.1
*
* IMPORTANT---SAVE AS DM99-2
* SET VDP WRITE ADDRESS
PRNT   LWPI MYWS
       BL   @SETUP
PRNLNK CLR  R0
       LI   R0,>21
       LI   R1,PRINTX
       LI   R2,30
       BLWP @PBASIC
       LI   R0,>42
       LI   R1,PDATA1+10
       MOV  @PDATA1+8,R2
       JEQ  PRNT1
       BLWP @PBASIC
PRNT1  LI   R2,>20
       CLR  @PDATA1+8
       BLWP @INPUT0
       MOV  R2,R2
       JEQ  PRNT2
       MOV  R2,@PDATA1+8
PRNT2  B    @RETURN
DISPRT LI   R1,PDATA1+8
       MOV  *R1+,R2
       JEQ  DSPRT1
       LI   R0,>21
       BLWP @PBASIC
DSPRT1 RT
VDPWRT ORI  R0,>4000     SET WRITE FLAG
* SET VDP READ ADDRESS
VDPRED SWPB R0           WRITE LSBY FIRST
       MOVB R0,@VDPWA    WRITE
       SWPB R0           GET MSBY
       MOVB R0,@VDPWA    WRITE
       ANDI R0,>3FFF     ZERO WRITE FLAG
       RT                RETURN
* R/W SECTOR ROUTINES
READ   MOVB @INPUT,R1    READ FLAG
       JMP  DODSR
WRITE  MOVB @OUTPUT,R1   WRITE FLAG
DODSR  SWPB R1
       MOVB @DRIVE,R1    GET DRIVE NUMBER
```

```
       MOV  R1,@FAC+2
       LI   R1,VDPBUF    VDP BUFFER
       MOV  R1,@FAC+4
       MOV  R0,@FAC+6    SECTOR NUMBER
       LI   R0,PAB
GODSR  MOV  R0,@PNTR
       BLWP @DSRLNK      GO TO DSR ROUTINE
       DATA >A           CODE FOR SUBROUTINE
       MOVB @FAC+6,R0    GET ERROR CODE
       JNE  IOERR
       RT                RETURN
* CHECKS TO SEE IF FILE NAME IN STRBUF IS LEGAL
LEGAL  LI   R0,STRBUF
       LI   R1,10        LENGTH
       CB   *R0,@SPACE   IS THE FIRST CHAR A SPACE?
       JEQ  IOERR        YES. ILLEGAL
LEGAL1 CB   *R0+,@H2E    IS IT A PERIOD
       JEQ  IOERR        YES. ILLEGAL
       DEC  R1
       JNE  LEGAL1
       LI   R0,STRBUF
       LI   R1,10
* FIND FIRST SPACE CHARACTER
LEGAL2 CB   *R0+,@SPACE
       JEQ  LEGAL3
       DEC  R1
       JNE  LEGAL2
       RT
* MAKE SURE NOTHING ELSE IS A SPACE
LEGAL3 DEC  R1
       JEQ  LEGAL5
LEGAL4 CB   *R0+,@SPACE
       JNE  IOERR
       DEC  R1
       JNE  LEGAL4
LEGAL5 RT
IOERR  LI   R0,ERRFE     FILE ERROR
IOERR1 MOV  R0,R3
       BL   @RESSPC      RESTORE VDP MEMORY
       B    @IOERR2
* RESTORE VDP MEMORY
RESSPC MOV  R11,R10
       LI   R0,PAB       START OF VDP MEMORY USED
       LI   R1,SAVBUF    BUFFER WHERE STORED
       LI   R2,316       316 BYTES
       BLWP @VMBW        WRITE IT OUT
       LI   R0,FACBUF
       LI   R1,FAC
       LI   R2,34
       BL   @MOVE
```

```
       B    *R10         RETURN
* RETURN TO BASIC
RETURN BL   @RESSPC      RESTORE VDP MEMORY
       LI   R3,>300
       BL   @SCROL1
       MOVB @SCREEN,R0   PRINTER?
       JNE  RETRN1       NO. SCREEN ONLY
* RESTORE GROM ADDRESS
       MOVB @GROMSV,@>9C02    GRMWA
       NOP
       MOVB @GROMSV+1,@>9C02  GRMWA
RETRN1 LWPI GPLWS        GPL WORKSPACE
       CLR  @STATUS      CLEAR STATUS BYTE
       B    @NEXT        BACK TO THE GPL INTERPRETER
FACBUF BSS  34
* SETUP MEMORY ROUTINE
SETUP  MOV  R11,@SETUPR+2     SAVE RETURN ADDRESS
       MOVB @HFF,@SCREEN SCREEN ONLY
* SAVE VDP MEMORY USED
       LI   R0,PAB       START OF USED VDP MEMORY
       LI   R1,SAVBUF    BUFFER TO PUT IT
       LI   R2,316       316 BYTES
       BLWP @VMBR        READ IT
* WRITE SECTOR SUBROUTINE CODE
*  R0 ALREADY SET FROM ABOVE
       LI   R1,SECSUB    >0110
       LI   R2,2         2 BYTES
       BLWP @VMBW        WRITE
       LI   R0,FAC
       LI   R1,FACBUF
       LI   R2,34
       BL   @MOVE
SETUPR B    @0           RETURN
* GET ONE CHARACTER FROM KEYBOARD
* INPUT: R0=SCREEN ADDRESS
* ACCESS WITH BL   @GETONE
*             TEXT 'chars'
*             BYTE 0
GETONE CLR  @KEYNUM
       MOV  R0,R3        SAVE SCREEN ADDR
       LI   R1,>7E00
       BLWP @VSBW
       CLR  @STATUS
       MOV  R11,R10      SAVE RETURN ADDR
GETON0 MOV  R10,R11      SAVE RETURN ADDR
GETON1 BLWP @KSCAN
       CB   @ANYKEY,@STATUS
       JNE  GETON1
       CLR  R0
       MOVB @KEY,R0
       CB   @KEY,@BACKEY
```

```
       JNE  GETON4
       B    @RETURN
GETON4 CB   R0,@UARROW
       JEQ  GETON3
       CLR  R1
GETON2 INC  R1
       MOVB *R11+,R2
       JEQ  GETON0
       CB   R2,R0
       JNE  GETON2
       MOVB R0,R1
       AI   R1,>6000
       MOV  R3,R0
       BLWP @VSBW
       MOVB R1,R0
       AI   R0,->6000
       ANDI R1,>FF
GETON3 MOVB *R11+,R2
       JNE  GETON3
       INC  R11
       ANDI R11,>FFFE
       RT
* MOVE BYTES
MOVE   MOVB *R0+,*R1+    COPY BYTE
       DEC  R2           DONE?
       JNE  MOVE         NO
       RT                RETURN
* MOVE BYTES, BACKWARDS
BMOVE  A    R2,R0        R0=R0+R2
       A    R2,R1        R1=R1+R2
BMOVE1 DEC  R0           -1
       DEC  R1           -1
       MOVB *R0,*R1      COPY BYTE
       DEC  R2           DONE?
       JNE  BMOVE1       NO
       RT                RETURN
* SCROLL SCREEN. DOES NOT SCROLL BOTTOM LINE
SCROLL LI   R3,>2E0
SCROL1 MOV  R11,@SCROLR+2     SAVE RETURN ADDRESS
       LI   R0,>20       R1,C0
       LI   R1,SCRBUF    SCREEN BUFFER
       MOV  R0,R2        32 BYTES/LINE
SCROL2 BLWP @VMBR        READ LINE
       S    R2,R0        GO BACK A LINE
       BLWP @VMBW        WRITE IT
       AI   R0,>40       GO FORWARD 2 LINES
       C    R0,R3        SEE IF AT END
       JNE  SCROL2       NO
       BL   @CLRBUF      CLEAR SCRBUF
       MOV  R3,R0
```

```
       LI   R1,SCRBUF    BLANKS
       LI   R2,>20       32 BYTES
       S    R2,R0        CORRECT
       BLWP @PBASIC      WRITE WITH BASIC OFFSET
SCROLR B    @0           RETURN
PRINTX TEXT 'PRINTER NAME(BLANK FOR SCREEN)'
       EVEN
* OPEN PRINTER PAB, IF NEEDED
OPNPAB MOV  @PDATA1+8,R0
       JEQ  OPNPBR
       SZCB @HFF,@SCREEN SCREEN AND PRINTER
* SAVE GROM ADDRESS
       MOVB @>9802,@GROMSV    GRMRA
       NOP
       MOVB @>9802,@GROMSV+1  GRMRA
       DEC  @GROMSV      CORRECT
       LI   R0,PAB1      PRINTER PAB
       LI   R1,PDATA1    PRINTER PAB DATA
       LI   R2,42        42 BYTES
       BLWP @VMBW        WRITE TO VDP
       LI   R0,PAB1+9    LENGTH BYTE
       MOV  R0,@PNTR
       BLWP @DSRLNK      GOTO DSR ROUTINE
       DATA 8            CODE FOR NORMAL FILE I/O
       JNE  OPNRET       NO ERROR
       B    @IOERR       ERROR!
OPNRET LI   R0,PAB1      PRINTER PAB
       LI   R1,>0300     "PRINT" OPCODE
       BLWP @VSBW        WRITE TO VDP
OPNPBR RT                RETURN
* CLEAR SCREEN
CLS    MOV  R11,R10
       CLR  R0
       BL   @VDPWRT
       LI   R0,>8000
       LI   R1,>300
CLS1   MOVB R0,@VDPWD
       DEC  R1
       JNE  CLS1
       B    *R10
* CLOSE PRINTER PAB, IF NEEDED
CLSPRN MOVB @SCREEN,R0   SCREEN AND PRINTER?
       JNE  CLSPN1       NO, JUST SCREEN
       LI   R0,PAB1      PRINTER PAB
       LI   R1,>0100     "CLOSE" OPCODE
       BLWP @VSBW        WRITE TO VDP
       LI   R0,PAB1+9    LENGTH BYTE
       MOV  R0,@PNTR
       BLWP @DSRLNK      GOTO DSR ROUTINE
       DATA 8            CODE FOR NORMAL FILE I/O
       JNE  CLSPN1       NO ERROR
```

```
       B    @IOERR       ERROR!
CLSPN1 RT
* SET SCRBUF TO SPACES
CLRBUF LI   R0,SCRBUF    R0=START OF SCRBUF
       LI   R1,>2020     R1=TWO SPACE CHARS.
       LI   R2,>20       R2=LENGTH OF SCRBUF
CBUF1  MOV  R1,*R0+      MOVE SPACES TO SCRBUF
       DECT R2           AT END?
       JNE  CBUF1        IF NOT, THEN GOTO CBUF1
       RT                RETURN
* TYPES (D/F,D/V,I/F,I/V,PRG)
TYPEBY BYTE >00,>80,>02,>82,>01
FILE   MOV  R11,R10      SAVE RETURN ADDR
       BL   @CLRBUF
* GET FILE NAME
       LI   R0,CATBUF    SECTOR BUFFER
       LI   R1,SCRBUF    SCREEN BUFFER
       LI   R2,10        10 BYTES
       BL   @MOVE        MOVE
* GET SIZE
       MOV  @CATBUF+14,R0     GET SIZE
       INC  R0           INCREMENT FOR FDR SECTOR
       LI   R1,SCRBUF+11 SCREEN BUFFER
       BLWP @CNS         CONVERT NUMBER TO STRING
* GET TYPE
       MOVB @CATBUF+12,R2     GET TYPE BYTE
       ANDI R2,>8300     GET ONLY BITS 0,6,7
       LI   R1,4
TYPE1  CB   @TYPEBY(R1),R2    SEE IF RIGHT TYPE
       JEQ  TYPE2        YUP
       DEC  R1           DONE?
       JOC  TYPE1        NOPE
       JMP  TYPPRG       CALL IT A PROGRAM FILE
TYPE2  INC  R1           +1
* CHECK PROTECTION
       MOVB @CATBUF+12,R0     GET TYPE
       ANDI R0,>0800     GET ONLY BIT 4
       JEQ  TYPE2A       NOT PROTECTED
       LI   R0,>5000     "P"
       JMP  TYPE3
TYPE2A LI   R0,>5500
TYPE3  MOVB R0,@SCRBUF+26     SCREEN BUFFER
* STILL IN TYPE...
       CI   R1,5         COMPARE R0 TO 5 (CODE FOR PROGRAM FORMAT)
       JNE  TYPE4        IF NOT EQUAL TO 5, THEN GOTO TYPE1
TYPPRG LI   R0,PROG      R0=PROG (TEXT FOR PROGRAM TYPE)
       LI   R1,SCRBUF+16 R1=SPACE IN VDP BUFFER FOR TYPE
       LI   R2,7         R2=7 (LENGTH OF TYPE)
       BL   @MOVE        GOSUB MOVE
       JMP  FILER        PRINT LINE
```

```
TYPE4  DEC  R1           DECREMENT R1
       A    R1,R1        DOUBLE IT
       MOV  @FLETAB(R1),R0    R0=ADDR. OF THE TYPE
       LI   R1,SCRBUF+16 R1=SPACE IN SCRBUF FOR TYPE
       LI   R2,5         R2=LEN OF TYPE
       BL   @MOVE        GOSUB MOVE
* RECORD LENGTH (X/X RL, I.E. D/V 40)
       MOVB @CATBUF+17,R0
       SRL  R0,8
       LI   R1,SCRBUF+21 R1=SPACE IN SCRBUF FOR RECORD LENGTH
       BLWP @CNS         CONVERT NUMBER TO STRING
FILER  B    *R10         RETURN
FILES  MOV  R11,@FILESR+2     SAVE RETURN ADDRESS
       LI   R5,DBUF1
FILES1 MOV  *R5+,R0
       JEQ  FILESR
       BL   @READ        READ SECTOR
       LI   R0,VDPBUF
       LI   R1,CATBUF
       LI   R2,28
       BLWP @VMBR
       CLR  R0           R0=0
       BL   @FILE        GOSUB FILE
       LI   R0,SCRBUF
       LI   R1,>1E
       BLWP @WRTLIN      GOSUB WRITE
       CLR  @KEYNUM      CLEAR KEYBOARD NUMBER
       CLR  @STATUS
       BLWP @KSCAN       GOSUB KSCAN
       CB   @KEY,@HFF    COMPARE KEY TO >FF
       JEQ  FILES1       IF NO KEY HIT THEN FILES1
* MOD FOR FCTN 4 ADDED BY KEN WOODCOCK. THANKS KEN!!
       CB   @KEY,@H02
       JEQ  FILESR
FILES2 BLWP @KSCAN       GOSUB KSCAN
       MOVB @STATUS,R0   MSBY OF R0 TO STATUS BYTE
       JEQ  FILES2       IF EQUAL TO 0, THEN GOTO WAITKY
       CB   @KEY,@H02
       JNE  FILES1
FILESR B    @0           RETURN
HEAD   MOV  R11,@HEADR+2 SAVE RETURN ADDRESS
       BL   @CLRBUF
       CLR  R0
       BL   @READ        READ SECTOR 0
       LI   R0,VDPBUF
       LI   R1,DBUF1
       LI   R2,SECLEN
       BLWP @VMBR
       LI   R0,HEAD0     R0=START OF HEAD0
       LI   R1,SCRBUF    R1=START OF SCRBUF
       LI   R2,14        R2=14 (LENGTH OF HEAD0)
```

```
       BL   @MOVE        GOSUB MOVE
       LI   R0,DBUF1
       LI   R1,SCRBUF+14 R1=SCRBUF+14
       LI   R2,10
       BL   @MOVE
       LI   R0,SCRBUF
       LI   R1,>1E
       BLWP @WRTLIN      WRITE TO SCREEN
       LI   R0,HEAD1     R0=HEAD1
       LI   R1,SCRBUF    R1=SCRBUF
       LI   R2,>1E       R2=>1E (LENGTH OF HEAD1)
       BL   @MOVE        GOSUB MOVE
* THANKS TO MACK MCCORMICK FOR THE BIT COUNTING ROUTINE
       LI   R2,200
       CLR  R3
       LI   R4,DBUF1+56
BITC0  MOVB *R4+,R1
       AI   R1,>0100
       SRL  R1,8
       JEQ  BITC1
       DEC  R1
       JNE  BITC2
       AI   R3,8
       JMP  BITC1
BITC2  LI   R0,8
BITC4  SRL  R1,1
       JOC  BITC3
       INC  R3
BITC3  DEC  R0
       JNE  BITC4
BITC1  DEC  R2
       JNE  BITC0
* NOW R3=#FREE SECTORS
       MOV  R3,R0
       LI   R1,SCRBUF+5
       BLWP @CNS
       MOV  @DBUF1+10,R1
       S    R0,R1        NOW R1=#USED SECTORS
       MOV  R1,R0
       LI   R1,SCRBUF+15
       BLWP @CNS
       LI   R0,SCRBUF
       LI   R1,>1E
       BLWP @WRTLIN      WRITE TO SCREEN
       LI   R0,HEAD2     R1=HEAD2
       BLWP @WRTLIN      GOSUB WRITE1
       LI   R0,HEAD3     R1=HEAD3
       BLWP @WRTLIN      GOSUB WRITE1
HEADR  B    @0           RETURN
DM99TX TEXT '                              '
```

```
       TEXT '  D I S K   M A N A G E R   9 9 '
       TEXT '                                '
       TEXT '     FAIRWARE BY  MIKE DODD     '
       TEXT '                                '
       TEXT '  1. FILE COMMANDS              '
       TEXT '  2. CATALOG DISK               '
       TEXT '  3. RENAME DISK                '
       TEXT '  4. INITIALIZE DISK            '
       TEXT '  5. TEST DISK                  '
       TEXT '  6. TOGGLE PRINTER             '
       TEXT '  7. RETURN TO BASIC            '
       TEXT '                                '
       TEXT '  ENTER NUMBER:                 '
       EVEN
DM99VC DATA FILLNK,CATLNK,DRELNK
       DATA INTLNK,TSTLNK,PRNLNK
       DATA RETURN
DM99   LWPI MYWS
       BL   @SETUP
       LI   R0,>7E00
       MOVB R0,@>8301
       BL   @CLS
       CLR  R0
       LI   R1,DM99TX
       LI   R2,>1C0
       BLWP @PBASIC
       LI   R0,>1AF
       BL   @GETONE
       TEXT '1234567'
       BYTE 0
       JEQ  CHK2
       DEC  R1
       SLA  R1,1
       MOV  @DM99VC(R1),R9
       BL   @CLS
       B    *R9
CHKKEY MOV  @>2002,R0
       CI   R0,>24F4
       JNE  CHK1
       CLR  @>83C4
       RT
CHK1   CLR  R0
       LI   R12,>24
       LDCR R0,3
       CLR  R12
       TB   9
       JEQ  CHK2
       LI   R12,>24
       LI   R0,>0200
       LDCR R0,3
       CLR  R12
```

```
       TB   8
       JNE  DM99
CHK2   RT
* END OF DM99-2 FILE
```

Disk 48. Contents of file DM99/DOC

DISK MANAGER 99, Version 2.1

Release dates: Version 1.0 85.1115
Version 1.1 86.0328
Version 2.0 86.0913
Version 2.1 86.1202

Disk Manager 99 is a fairware program. Please give it to anyone who wants it, but do not alter the program in any way. Also, please copy the source code, as well as the other files. If you would like a copy of V1.1 or V2.1 and do not already have it, send me a disk, mailer, return postage, and $1 (or $6 to cover it all). The advantage to V1.1 is that it will interface with RUNning XBasic programs, which V2.x will not. If you like this program, please send whatever you think it is worth to:

Mike Dodd
116 Richards Drive
Oliver Springs, Tenn 37840 USA
(615)435-1667

Paid users will receive notices of any updates.

Disk Manager 99 is a resident disk manager program for use with Extended Basic. It can unprotect files, write-protect files, rename files, rename a disk, test a disk, catalog a disk to screen or printer, and initialize a disk. Disk Manager 99 can be used from command mode (or program mode with V1.1) without harming the program in memory. The only routine that takes an abnormally large space in memory is the initialize disk routine, and it will check to see if enough memory is available before beginning. The program requires at least one disk drive, memory expansion, and Extended Basic. A printer is optional. On the disk are eight files:

FILENAME	*CONTENTS*
XBDM99	Driver source code for Extended Basic version
XBDM99/O	Object code for Extended Basic version
DM99-1	Main source code file, part 1
DM99-2	Main source code file, part 2
DM99/DOC	Instructions (this file)
PRINTDOCS	XB program to print instructions
LOAD	XBasic program that instantly loads DM99

To load the Extended Basic version, you should type CALL INIT :: CALL LOAD("DSK1.XBDM99/O") **ENTER**, or RUN the LOAD file for the fast loader program. Note that using the fast loader will destroy any assembly language or Extended Basic programs already in memory. Loading XBDM99/O does not have this problem. If you already have an Extended Basic program in memory, you must either SAVE it, RUN the LOAD program, and reload your program; or, load the XBDM99/O file.

Version 1.1 and Version 2.x both can do the same things, with the exception that with V2.x you cannot specify a number of tracks other than 40 in the initialize disk routine. The difference is that you access DM99 with CALL LINK statements in V1.1, and by menu selection in V2.x. First, the instructions for accessing V1.1:

The format for each of the commands follows. The lower case letter "d" will indicate the drive number. Items in brackets are optional.

CALL LINK("UNPROT",d,"filename") — will unprotect "filename"

CALL LINK("PROT",d,"filename") — will protect "filename"

CALL LINK("FRENAM",d,"old filename","new filename") — renames "old filename" as "new filename"

CALL LINK("DRENAM",d,"new diskname") — will rename a disk

CALL LINK("TEST",d[,"printer name"]) — will test the disk. Will output to screen, or printer if specified

CALL LINK("CAT",d[,"printer name"]) — will catalog the disk. Will output to screen, or printer if specified. Press any key to pause and resume.

CALL LINK("INIT",d,"disk name",#sides,density,"verify?"[,tracks]). Will initialize disk. Can specify number of sides, and the density (1=single, 2=double). If "verify?" = "V" then it will verify the sectors. If number of tracks is not specified, then it will assume 40 tracks.

Now for instructions on Version 2.1:

There are several ways to access DM99 V2.1. The easiest is to press **CTRL D** in command mode. Another way is to type CALL LINK("DM99"). Both bring up a menu of:

1. FILE UTILITY
2. DISK CATALOG
3. DISK RENAME
4. DISK INITIALIZE
5. DISK TEST
6. TOGGLE PRINTER
7. RETURN TO BASIC

Press the number of your choice. File utility allows you to delete, rename, unprotect, and protect files. Toggle printer will allow you to enter a printer name. The output from the catalog disk and disk test options will now be directed to the printer as well as the screen. Press toggle printer again to go to screen only. See the text below for details of the other options.

The other way to access DM99 V2.1 is by CALL LINKs that directly access the specific area needed, bypassing the menu. When the program asks for "DRIVE?" press a number from 1 through 9. Explanations for the commands follow.

File utility — CALL LINK("FIL") — Asks for drive number and filename. It then searches the disk for the file and prints the directory information on the screen. In the C field, enter N for no operation or D for delete file. In the filename field, enter a new filename to rename the file. In the P field, enter P to protect or U to unprotect. **FCTN S** and **FCTN D** are active. On pressing **ENTER**, all commands will be carried out and the user will be returned to Basic.

Disk Catalog — CALL LINK("CAT") — Asks for drive number. Press any key to pause and resume the listing. Press **FCTN 4** to break out of the catalog. The "Numb" field on the second line displays the total number of files present on the disk.

Disk Rename — CALL LINK("DRENAM") — Asks for drive number and new diskname, then renames the disk.

Disk Init — CALL LINK("INIT") — Enter the drive number and disk name. Then enter the number of sides (1 or 2), the density (S or D), and "VERIFY?" (Y or N) without pressing **ENTER**. Press **FCTN E** to go up a field. If you enter Y for verify, DM99 will check every sector to make sure it is okay.

Disk Test — CALL LINK("TEST") — Asks for drive number. If you press **ENTER** without entering a device name the test report will go to screen only. The test is a read-only, non-destructive test.

Here are more complete instructions for accessing V1.1. Users of V2.1 may also find this section useful.

Unprotect a file — CALL LINK("UNPROT",d,"filename") — Searches DSKd for filename, then unprotects it.

Protect a file — CALL LINK("PROT",d,"filename") — Searches DSKd for filename, then protects it.

Rename a file — CALL LINK("FRENAM",d,"old filename","new filename") — Searches DSKd for old filename, and renames it as new filename. Before it renames it, it searches the disk for the new filename. If it is present, it will give an I/O error.

Rename a disk — CALL LINK("DRENAM",d,"new diskname") — Renames the disk in DSKd as new diskname.

Test a disk — CALL LINK("TEST",d[,"printer name"]) — Tests every sector on DSKd. If it finds a bad sector, it will print out the sector number to the screen. At the end, it will print "TOTAL ERRORS: XXXX". If there are no errors, it will not print a number, just "TOTAL ERRORS:". The printer name is optional. If specified, it will print all test output both to the screen and the printer.

Catalog a disk — CALL LINK("CAT",d[,"printer name"]) — Scans the disk in DSKd, and prints the catalog to the screen. Press any key to pause and resume. The printer name is optional. If specified, it will print out the catalog both to the screen and to the printer.

Initialize a disk — CALL LINK("INIT",d,"disk name",# sides,density,"verify?"[,tracks]) — Will initialize the disk in DSKd with the disk name, with # sides, and density (1=single, 2=double). If "verify?" = "" then will not verify sectors. If "verify?" = "V" then will verify sectors. The number of tracks is optional. If not specified, will assume 40 tracks.

If you have any questions about this program, please write or call me. If you have come up with updates to the program, please send the updated files to me and I will distribute it, with credit to you.

IMPORTANT NOTE REGARDING DOUBLE DENSITY INITIALIZATION - The XBasic version will not initialize double density on a CorComp disk controller card. If anyone can figure out how to get the XB version of DM99 to initialize in double density, PLEASE LET ME KNOW!

ACKNOWLEDGMENTS: I would like to thank the following people who helped me with Disk Manager 99 Version 2.1:

J. Peter Hoddie, who provided me with a copy of the TI disk controller manual, which enabled me to save quite a bit of memory in DM99.

D.R. Fudge, who was always there to test copies of DM99, and who always lent an ear during my trying times with this program. Craig Miller (MG), who was always willing to answer my every question about our little computer, and in doing so, provided immense technical help in the creation of DM99 V2.1

Ken Woodcock, who wrote the routine to exit from the catalog and test disk routines with the FCTN 4 key.

Dave Peden, who has sent me uncountable files, gave me suggestions for DM99, and who has always managed to cheer me up whenever I'm down.

Glen Pedersen, who, within a mere two days, beta tested all functions of DM99 V2.1. He is also the only person who has been able to give me reports on compatibility with the Myarc controller.

Last, but not least, Dean Mitchell, who probably gave me the fullest report on the test copy of DM99 V2.1 of all the people I sent it to. He gave me many suggestions, most of which are now present in DM99 V2.1.

IMPROVEMENTS

Version 1.1 — has a faster XB loader that loads DM99 is 20 seconds, instead of 55 seconds in V1.0 Version 1.1 fixes the bug with accessing the RS232 card and the thermal printer. Version 1.1 catalogs a disk 42% faster than V1.0.

Version 2.0 - lets the user access Disk Manager 99 by one keypress, allowing faster entering of commands. Also has all the improvements of Version 1.1.

Version 2.1 - fixes two major bugs resident in Version 2.0. Has a faster XB LOAD program, displays the total number of files on disk when cataloging a disk, checks for illegal file names (space or period), and is 100% compatible with the Myarc disk controller.

PERSONAL NOTE: I have had to delete the console basic version of DM99 in V2.1, due to problems interfacing with Basic's error routines. If anyone out there uses the console basic version, please let me know and I will continue working on it.

Disk 48. Contents of file XBDM99

```
*******************************************************
* DISK MANAGER 99. Extended Basic version             *
* For use with the Extended Basic cartridge           *
* Will allow you to access many disk-related functions*
* with a CALL LINK statement. (C) 1985,86 by Mike Dodd*
* Version 2.1 - Revision dates: V1.0 - 85.1115        *
*                               V1.1 - 86.0328        *
*                               V2.0 - 86.0810        *
*                               V2.1 - 86.1202        *
*******************************************************
* This program is fairware. Please copy it for your   *
* friends, but please do not alter the program. Also, *
* please copy the source code. Although the people you*
* give it to may not be interested, they many give it *
* to someone who is                                   *
*******************************************************
* If you like this program, and think it is worth     *
* paying for, please send whatever you think it is    *
* worth to:                                           *
* MIKE DODD                                           *
* 116 RICHARDS DRIVE                                  *
* OLIVER SPRINGS, TENN 37840 USA                      *
* (615)435-1667                                       *
*******************************************************
* IMPORTANT----SAVE THIS FILE AS XBDM99
*
* DEFINITION STATEMENTS
       DEF  FIL,DM99,DRENAM
       DEF  TEST,CAT,INIT
* REFERENCE EQUATES
VMBW   EQU  >2024
VMBR   EQU  >202C
STRREF EQU  >2014
NUMREF EQU  >200C
ERR    EQU  >2034
KSCAN  EQU  >201C
VSBR   EQU  >2028
VSBW   EQU  >2020
ERRMF  EQU  >C556        MEMORY FULL
ERRFE  EQU  >C5B5        FILE ERROR
* IMPORTANT: MUST CHANGE FOLLOWING ADDRESS TO >0800 IF
* USING WITH CONSOLE BASIC
PAB    EQU  >0968
DNTBUF EQU  >096A        BUFFER FOR INITIALIZE DISK.
       COPY "DSK2.DM99-1"
       COPY "DSK2.DM99-2"
* IMPORTANT: MUST CHANGE ERROR HANDLING ROUTINE IF
* USING WITH CONSOLE BASIC
```

```
IOERR2 MOV  R3,@>83EC    GPLWS R6
       LWPI >83E0
       B    @>60
* GPLLNK - by MILLLERS GRAPHICS
* MODIFIED BY MIKE DODD TO REDUCE # OF LABELS
* TOTAL: 70 BYTES
GPLLNK DATA GLNKWS       R7
       DATA GLINK1       R8
       DATA GLINK2       R9  WHERE GPL RETURNS TO US
       DATA >176C        R10 GROM ADDR FOR XML OPCODE
       DATA >50          R11  WHERE PUTSTK ADDR RESIDES
GLNKWS EQU  $->18
       BSS  8            R12-R15
GLINK1 MOV  *R11,@>83E8  PUTSTK ADDR IN GPLWS R4
       MOV  *R14+,@>83EC GPL ROUTINE ADDR IN GPLWS R6
       MOV  @>200E,R12   SAVE VALUE AT >200E
       MOV  R9,@>200E    PUT XMLRTN ADDR IN >200E
       LWPI >83E0
       BL   *R4          SAVE GROM ADDRESS
       MOV  @GLNKWS+20,@>8302(R4) PUSH XML ADD ON STACK
       INCT @>8373       ADJUST STACK POINTER
       B    @>60         GOTO GPL INTERPRETER
GLINK2 MOV  @>166C,R4    GET GETSTK POINTER
       BL   *R4          RESTORE GROM ADDRESS
       LWPI GLNKWS
       MOV  R12,@>200E   RESTORE >200E
       RTWP              ALL DONE - RETURN TO CALLER
* DSRLNK - by MILLERS GRAPHICS
* MODIFIED BY MIKE DODD TO REDUCE # OF LABELS
* REQUIRES GPLLNK
* WILL ACCESS CS1 AND CS2
* TOTAL - 186 BYTES, INCLUDING DSRLNK AND MG'S GPLLNK
*         116 BYTES, NOT INCLUDING MG'S GPLLNK
DSRLNK DATA DSRWS,DLINK1
DSRWS  EQU  $
DLINK1 MOV  R12,R12
       JNE  DLINK3
       LWPI GPLWS
       MOV  @>50,R4
       BL   *R4
       LI   R4,>11
       MOVB R4,@>402(R13)
       JMP  DLINK2
       DATA 0
       DATA 0,0,0
DLINK2 MOVB @>83E9,@>402(R13)
       MOV  @>166C,R5
       MOVB *R13,@DLINK5
       INCT @DLINK4
       BL   *R5
       LWPI DSRWS
```

```
       LI   R12,>2000
DLINK3 INC  R14
       MOVB *R14+,@>836D
       MOV  @>8356,R3
       AI   R3,-8
       BLWP @GPLLNK
DLINK4 BYTE >03
DLINK5 BYTE >00
       MOVB @DSRWS+7,@>8C02
       MOVB R3,@>8C02
       SZCB R12,R15
       MOVB @>8800,R3
       SRL  R3,5
       MOVB R3,*R13
       JNE  DLINK6
       COC  @>837C,R12
       JNE  DLINK7
DLINK6 SOCB R12,R15
DLINK7 RTWP
       AORG >2002        FIX FFA POINTER SO THAT CHKKEY DOESN'T CLR ISR LINK
       DATA >24F6
       AORG >83C4
       DATA PTITLE
       END
```

Disk 49. More Games

Version:
Requires: XB
Author: John Behnke, others
Language: XB, AL
Updated: 08/17/86

A collection of games including Boxing, Missile Command, and Rat Race. Also two different bowling programs and an assembly language variation of Pac-Man.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-49
Sectors total           = 360
Sectors used            = 337
Sectors available       = 21
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00b -README       4 DIS/VAR 80   >166 002 >00c 001
002 >002 BOWLING1     49 INT/VAR254   >022 048
003 >00a BOWLING2     18 PROGRAM      >155 017
004 >005 BOXING       34 PROGRAM      >0af 033
005 >007 BUZZARD      66 DIS/FIX 80   >0e9 065
006 >003 M            37 INT/VAR 80   >052 036
007 >006 MISSILE      26 PROGRAM      >0d0 025
008 >004 MONOPOLY     58 INT/VAR254   >076 057
009 >008 RAT/RACE     42 PROGRAM      >12a 041
010 >009 SCROLL        3 DIS/FIX 80   >153 002
```

Disk 49. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain Software Disk #49

More Games

BOWLING1 and BOWLING2: Two different bowling games in Extended BASIC. Varying levels of complexity.

BOXING: Interesting graphic game in Extended BASIC. By John Behnke

BUZZARD: Great assembly games. Loads with Editor/Assembler option 3.

M: Data file for Monopoly.

MONOPOLY: Extended BASIC version of Monopoly. Not very fast but very complete.

MISSILE: Missile Command in Extended BASIC. By John Behnke.

RAT/RACE: Betting game with amusing graphics by John Behnke. Extended BASIC.

SCROLL: Assembly support for RAT/RACE.

Notes compiled by J. Peter Hoddie.

Disk 50. RAG Assembler

Version: 6.0 *Author*: Art Greene
Requires: EA *Language*: AL *Updated*: 3/1/88

A complete MACRO assembler for the TI. Allows you to build and use complex macros for use in your assembly language programs. Comes with terminal emulator and disk cataloger programs as examples. This is the assembler that TI should have written!

```
dskdir. v2.0. 12-dec-96
Disk name               = RAGASM7-1
Sectors total           = 360
Sectors used            = 312
Sectors available       = 46
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 ASMMREF      66 DIS/VAR 80   >022 065
002 >003 ASMMREF1     44 DIS/VAR 80   >063 043
003 >004 ASMUSER      82 DIS/VAR 80   >08e 081
004 >005 ASMV7         8 DIS/VAR 80   >0df 007
005 >006 LOAD          3 PROGRAM      >0e6 002
006 >007 RAGASM       32 PROGRAM      >0e8 031
007 >008 RAGASN       33 PROGRAM      >107 032
008 >009 RAGINSASM    16 PROGRAM      >127 015
009 >00a RAGMAC       25 DIS/VAR 80   >136 024
010 >00b XRAGINSASM    3 PROGRAM      >14e 002
```

Disk 50. Contents of file ASMMREF

RAG SOFTWARE

TI-99/4A MACRO ASSEMBLER

Version 7

MACRO DESCRIPTIONS AND MACRO WRITING

CONTENTS

MACRO DESCRIPTIONS

DEVELOPING MACROS

September 1988

MACRO DESCRIPTIONS

The following sections describe the use of the macros in the RAGMAC macro library. There are three classes of macros in this library. First, general purpose macros:

[label] BE gad	Branch Equal
[label] BNE gad	Branch Not Equal
[label] CALL gad,p1,p2,p3,p4	Call Subroutine
[label] IF gas,relop,gad,target	IF word
[label] IFB gas,relop,gad,target	IF Byte
[label] IFSW gas,{ON|OFF},target	IF Switch
[label] LDB gas,wad	Load Byte
[label] MOVBL gas,gad,length	MOVe Bytes Long
[label] RCALL gad,R0=p0,R1=p1,R2=p2	Call Subroutine
[label] SETSW gad,{ON|OFF}	Set Switch
[label] SETV gas[,wad][,NOP]	SET Vdp address

Second, macros that invoke the subroutines provided as part of this package. *Note*: The two modules SRCRAGIO and SRCRAGDIS must be separately assembled and must be loaded with your program. The macros that call subroutines in these two modules are:

[label] BEEP [WAIT|NOWAIT]	Issue accept tone
[label] HONK [WAIT|NOWAIT]	Issue reject tone
[label] GET gad,gar	GET record
[label] INPUT gas,length,back-exit	INPUT from keyboard
[label] PUT gad,gar	PUT record
[label] PRINT gas[,gar]	PRINT to screen
[label] OPEN gad	OPEN DCB
[label] CLOSE gad	CLOSE DCB

Third, macros used to assemble control blocks.

[label] DCB [PAB=,END=,ERR=]	Data Control Block
[label] PAB [RL=,BU=,SO=,FD=, {VA|FI},{DI|IN},{SE,RE}, {INP|OU|AP|UP}]	Define PAB
[label] SCB [TOP=,BOT=,LEN=,SO=, INP=,TE,PR,SP,SI,SC, RM,ML]	Screen Control Block
prefix PABEQU base-PAB-name	Labels for PAB fields

BE — Branch Equal

```
[label]     BE    destination [comment]
```

Causes a branch to the destination if the EQ status is set. The destination operand is coded as a general address.

Examples:

```
            BE    @GOOD
TEST        BE    *R8
```

BEEP — Issue Accept Tone

```
[label]     BEEP  [WAIT|NOWAIT]
```

Calls the RAGBEP entry in RAGDIS to issue the accept tone via GPL, and optionally waits till the tone finishes.

Examples:

```
            BEEP  WAIT
GOOD        BEEP  NOWAIT
            BEEP
```

BNE — Branch Not Equal

```
[label]     BNE   destination [comment]
```

Causes a branch to the destination if the EQ status is not set. The destination operand is coded as a general address.

Examples:

```
            BNE   @GOOD
TEST        BNE   *R8
```

CALL — Call Subroutine

```
[label]     CALL  destination,p1,p2,p3,p4 [comment]
```

Causes a BLWP to the destination subroutine and generates a parameter list following the BLWP for the parameters "p1" to "p4". All parameters are optional and DATA statements are generated only for as many parameters as are specified.

Examples:

```
            CALL  @DSRLNK,8
CSUB1       CALL  @SUB1,PARM1,PARM2,PARM3
```

CLOSE — Close a Data Control Block

```
[label]     CLOSE gad
```

Calls RAGCLO in RAGIO to terminate processing of a file. The address of the DCB, "gad", is specified as a general address. Closing a DCB that is not open has no effect.

Examples:

```
DONE        CLOSE @OUTPUT
            CLOSE *R8
```

DCB — Define Data Control Block

```
[label]     DCB   PAB=,END=,ERR=
```

Assembles a Data Control Block for the use of OPEN, CLOSE, GET and PUT calls to the RAGIO subroutine package. PAB= specifies the VDP address to which the PAB is written. END= specifies the label of the routine which is transferred to when end of file is read. ERR= specifies the label of the routine which is transferred to if an error occurs during file processing. PAB=, END= and ERR= are all optional and may be specified in any order. The DCB must be followed in CPU RAM by the PAB for the file. A full description of the DCB fields is given in the source of RAGIO.

Examples:

```
INPUT       DCB   PAB=>0C00,END=EOFIN
            PAB   BU=>380,INP,FD=DSK1.INPUT
OUTPUT      DCB   PAB=>0C20,ERR=ERROUT
            PAB   BU=>400,RL=80,VA,DI,OU
```

NOTE that the DCB should be closed after an end of file or a file error; otherwise it cannot be reused. (An alternative to closing the DCB is to zero the first word of the DCB.)

GET — Get Next Record

```
[label]     GET   gad,gar
```

Calls the RAGGET entry in RAGIO to get the next sequential record from the file defined by the DCB "gad". The record is returned in the area designated by "gar". Both "gad" and "gar" are coded as general addresses.

Examples:

```
NXTREC      GET @INDCB,@INREC
            GET @INDCB,R5           R5->RECORD AREA
```

HONK — Issue Reject Tone

```
[label]     HONK  [WAIT|NOWAIT]
```

Calls the RAGHON entry in RAGDIS to issue the reject tone via GPL, and optionally waits till the tone finishes.

Examples:

```
            HONK WAIT
BAD         HONK NOWAIT
```

IF — IF Word Jump

```
[label]     IF    source,relop,destination,target     [comment]
```

Compares the word at "source" to the word at "destination" then jumps to label "target" depending upon the "relop". The relational operator "relop" is any of the jump conditions (i.e. EQ for JEQ, H for JH, etc.). The source and destination operands are coded as general addresses, OR the destination may be coded as a literal value and the source as a register value. A literal is written as an equals sign followed by a self defining value. When a literal destination is coded, a CI instruction is generated.

Examples:

```
            IF    @X,EQ,*R5+,EQUAL
TEST        IF    *R3,GT,@Y,GREAT
            IF    R4,EQ,=10,EQUAL         R4 = 10?
```

IFB - IF Byte Jump

```
[label]     IFB   source,relop,destination,target     [comment]
```

Compares the byte at "source" to the byte at "destination" then jumps to label "target" depending upon the "relop". The relational operator "relop" is any of the jump conditions (i.e. EQ for JEQ, H for JH, etc.). The source and destination operands are coded as general addresses.

Examples:

```
            IFB   @X,EQ,*R5+,EQUAL
TEST        IFB   *R3,GT,@Y,GREAT
```

IFSW - IF Switch Jump

```
[label]     IFSW  source,{ON|OFF},target        [comment]
```

Tests the switch "source" and jumps to label "target" depending upon whether the switch is ON or OFF. The source operand is coded as the general address of the switch word. A switch is ON if the value of the word is non-zero and is OFF if the word is zero. Switches can be set with the SETSW macro described later.

Examples:

```
            IFSW  @SW1,ON,SW1ON
TEST        IFSW  *R3,OFF,SKIP
```

INPUT — Read Input from Keyboard

```
[label]     INPUT gas,len,back
```

Calls the RAGKIN entry in RAGDIS to read data from the keyboard. The input characters are displayed on the screen as defined by the SCB designated by "gas", a general address. The maximum length of the input is specified by "len" which may be a general address of the word containing the maximum length, or may be a literal value. A literal value is an equals sign followed by a self-defined value. If **FCTN 9 (BACK)** is pressed a transfer is made to the label "back". If the SCB is specified as zero then the SCB specified in the previous call to RAGDIS is used.

Examples:

```
GETKEY      INPUT @SCB1,@LEN,BACK
            INPUT @SCB2,=10,REDO
            INPUT @0,=>10,BACK
```

LDB — Load Byte

```
[label]     LDB   source,destination
```

Loads the "source" byte as a word into the "destination" register. The "source" operand is specified as a general address. The "destination" operand must be a workspace register.

Examples:

```
LOAD        LDB   @A,R1
            LDB   *R1+,R4
```

MOVBL — Move Bytes Long

```
[label]     MOVBL source,destination,length     [comment]
```

Moves the bytes from "source" to "destination". The number of bytes moved is specified by "length" which may be the general address of the word containing the length or may be a literal length. A literal is written as an equals sign followed by a self defining value. This macro generates a loop to move the bytes. The "source" and "destination" operands are coded as general addresses. If "source" is specified as a symbolic memory reference then R0 is used. If "destination" is specified as a symbolic memory reference then R1 is used. If "length" is specified as a symbolic memory reference or a literal then R2 is used.

Examples:

```
            MOVBL @X,@Y,@LEN
MXY         MOVBL @X,@Y,=32
            MOVBL *R5+,*R7+,R3
            MOVBL *R10,@X,=20
```

OPEN - Open a Data Control Block

```
[label]     OPEN  gad
```

Calls the RAGOPE entry in RAGIO to prepare a file for processing. The address of the DCB, "gad", is specified as a general address.

Examples:

```
OPEN1       OPEN  @OUTPUT
            OPEN *R8
```

PAB - Peripheral Access Block

```
[label]     PAB   [RL=,BU=,SO=,FD=,
                  {VA|FI},{DI|IN},{SE,RE},
                  {INP|OU|AP|UP}]
```

Assembles a Peripheral Access Block in CPU RAM as defined by the operands. The operands may occur in any order. RL= specifies the record length. BU= specifies the buffer address in VDP RAM. SO= specifies the screen offset value. FD= specifies the file/device name. The record format is specified as VAriable or FIxed. The data format is specified as DIsplay or INternal. The access mode is specified as SEquential or RElative. The processing mode is specified as INPut, OUtput, APpend or UPdate.

Examples:

```
FILE1       PAB   BU=>0C00,INP,FD=DSK1.INPUT
FILE2       PAB   BU=>0D00,OU,VA,RL=80
```

Note: The file/device name field of the PAB is always assembled at least 28 bytes long.

PABEQU — Name PAB Fields

```
            prefix PABEQU base-PAB-name
```

Generates names for the PAB fields. The names are generated using the 1 to 4 character "prefix" followed by a two character field name. The two character field names are:

OP	PAB operation code (byte)
FL	PAB flags/error code field (byte)
BU	VDP buffer address (word)
RL	Record length filed (byte)
CC	Character count field (byte)
RN	Record Number (word)
SL	Save/Load length (word)
SO	Screen offset (byte)
ST	Status (byte)
NL	Name length field (byte)
FD	File Description (bytes)

PRINT — Screen Display

```
[label]     PRINT gas,gar
```

Calls the RAGDS1 entry in RAGDIS to display the data specified by "gar" on the screen as specified by the SCB "gas". Both "gar" and "gas" are coded as general addresses. The data is a variable length record (i.e. a one byte length followed by the data to be displayed). If no "gad" is specified, a call is made to RAGSCL to scroll the screen up one line. If the SCB is specified as zero the SCB from the previous call to RAGDIS is used.

Examples:

```
SHOW        PRINT @SCB1,DATA
            PRINT @0,*R7
SPACE       PRINT @SCB2
```

PUT — Put Data Record to a File

```
[label]     PUT   gad,gar
```

Calls the RAGPUT entry in RAGIO to write the next sequential record into the file defined by the DCB "gad". The record to be written is in the area designated by "gar". Both "gad" and "gar" are coded as general addresses.

Examples:

```
NXTREC      PUT   @OUTDCB,@REC
            PUT   @OUDCB,R5          R5->RECORD AREA
```

RCALL — Call Subroutine

```
[label]     CALL  destination,R0=p0,R1=p1,R2=p2       [comment]
```

Causes the specified registers to be loaded using LI instructions and a BLWP to the "destination" subroutine. All parameters are optional and LI instructions are generated only for those parameters specified.

Examples:

```
            RCALL @VMBW,R0=VADDR,R1=CADDR,R2=20
VREAD       RCALL @VSBR,R0=>0020
```

SCB — Screen Control Block

```
[label]     SCB   [TOP=,BOT=,LEN=,SO=,INP=
                  TE,PR,SP,SI,SC,RM,ML]
```

Assembles a Screen Control Block for use by RAGDIS. TOP= specifies the VDP address of the first position of the top line of screen. BOT= specifies the VDP address of the first position of the last line of the screen. LEN= specifies the length of a screen line. SO= specifies the screen offset to be added to each character displayed on the screen. INP= specifies the VDP address of the first byte of the input area. TE specifies the VDP is in text mode. PR specifies input is to be prompted for with a number sign. SP specifies that a scroll is to be done before a PRINT. SI specifies that a scroll is to be done before INPUT. SC specifies that a scroll is to be done before both PRINT and INPUT. RM specifies Return on Maximum input length without ENTER being pressed. ML specifies that PRINTS are to be Multi Line if the data exceeds the length of a single line.

Several SCBs can be used giving you a window capability.

Examples:

```
SC1         SCB   TOP=2,BOT=>62,INP=>300,LEN=28,SP
SC2         SCB   TOP=>82,BOT=>2C2,LEN=28,ML,SP
```

SETSW - Set Switch

```
[label]     SETSW gad,{ON|OFF}
```

Sets the switch word specified by the general address "gad" ON or OFF. The switch value for OFF is zero (set via the CLR instruction). The switch value for ON is non-zero (set by the SETO instruction).

Examples:

```
            SETSW @SW1,ON
SET         SETSW *R3,OFF
```

SETV — Set VDP Address

```
[label]     SETV  gas[,wad],[NOP]
```

Sets the VDP address specified by "gas". If gas is a symbolic memory address or a literal then R0 is used. If specified, "wad" is a register that has the VDP write address address, >8C02. If specified, NOP, indicates that a NOP instruction should be generated after the VDP address is set.

Examples:

```
            LI R15,>8C02
            SETV =>0C00,R15
SET         SETV R3
            SETV R3,R15,NOP
```

Disk 50. Contents of file ASSMREF1

DEVELOPING MACROS

The macro library is a DISPLAY/VARIABLE 80 file and can be edited the same as an assembler source file. Note that the order of the macros in the library is not important. When developing a new macro you should place the macro definition in your source file. Then when you are satisfied that the macro works you should copy it to the macro file.

All macro definitions are kept in memory during assembly so that the size and number of macro statements is limited. When you copy your macro definitions to the macro library you should "compress" them. That is you should remove all unnecessary blanks and all comments. You should periodically review your use of macros and remove from the library any that you find you are not using.

The following discussion of macros is in the nature of a tutorial on what a macro is, and how to develop one.

Let us try some analogies in order to get a feeling for what macros are and how to make use of them. Like most analogies, none of them are completely accurate.

1. Macros are like subroutines, (generally very small subroutines),
2. Macros are like COPY statements,
3. Macros are like user defined functions in BASIC (i.e. DEF SQ(X)=X*X),
4. Macros are like small programs that generate assembler source statements.

First, we will look at analogy number 1. Suppose that in a program you had many places where you want to move 10 bytes of data from one place in memory to another. You could code a "MOVE" subroutine as shown below.

```
Line/Pos  12345678901234567890123456789012345678901
001       *...
002               BL   @MOVE
003       *...
004               BL   @MOVE
005       *...
006       MOVE    LI   R0,X          move 10 from x to y
007               LI   R1,Y
008               LI   R2,10
009               MOVB *R0+,*R1+
010               DEC  R2
011               JGT  $-4
012               RT
013       X       BSS  10
014       Y       BSS  10
015               END
```

Now, if you wanted extreme speed you would repeat the code in lines 6 through 11 at line 2 then again at line 4 and so on for every reference to MOVE. If you used your move subroutine a lot of times in the program, this repetition would get tiresome. With the macro assembler, you could define a macro and use it wherever the move was needed. For example:

```
Line/Pos  12345678901234567890123456789012345678901234567890
001       $MACRO MOVE              BEGINNING OF DEFN
002              LI   R0,X         move 10 from x to y
003              LI   R1,Y
004              LI   R2,10
005              MOVB *R0+,*R1+
006              DEC  R2
007              JGT  $-4
008       $END                     END OF DEFN
009       *...
010              MOVE
011       *...
012              MOVE
013       *...
014       X      BSS  10
015       Y      BSS  10
016              END
```

When this code is assembled, the assembler stores away the macro definition in lines 1 to 8, then later when it sees the macro name, MOVE, used as an operation code in lines 10 and 12, it substitutes the statements inside the macro definition into the source.

Using analogy number 2, the above could be accomplished by putting lines 2 through 7 into a file, say DSK1.MOVE and then recoding the program as shown below.

```
Line/Pos  12345678901234567890123456789012345678901234567890
009       *...
010              COPY "DSK1.MOVE"
011       *...
012              COPY "DSK1.MOVE"
013       *...
014       X      BSS  10
015       Y      BSS  10
016              END
```

At this point, you should enter the above two source programs and assemble them. Use options R (for registers), L (for listing) and G (for show generated statements). Compare the two listings and see what statements were actually assembled. Note on the listing that plus signs precede statements generated by a macro.

The above example of the use of a macro is very trivial. So trivial that you probably would not use a macro for this purpose. Let's extend the example a bit more.

Suppose, that each time you wanted to use MOVE, you wanted to move different strings of different lengths. Say the first time you wanted to move 20 bytes of A to B and the second time you wanted to move 10 bytes of X to Y. In this case, the "COPY" solution would not work since the code copied from the file is always the same, but it can be done with a macro.

Macros are like subroutines (i.e. CALL COLOR(. . .) in BASIC) in that they can have parameters. One parameter is the symbol you code in the label field of the macro statement. There are a maximum of nine other parameters which are the values coded in the operand field of the macro statement. The macro parameters have fixed "special" names. The symbol "&P0" is the name of the label field parameter, "&P1" is the name of the first operand, "&P2" the name of the second operand, and so on up to &P9". When the assembler sees one of these special names within a macro definition it substitutes the value specified on the macro statement. Thus each time you code a macro statement (i.e. every time you use the macro name as an operation code) with different operands, different values are substituted and different statements are "generated" by the macro. Let's now recode our example.

```
Line/Pos  12345678901234567890123456789012345678901234567890
001       $MACRO MOVE               BEGINNING OF DEFN
002       &P0    LI   R0,&P1          &P1 IS FROM
003              LI   R1,&P2          &P2 IS TO
004              LI   R2,&P3          &P3 IS LENGTH
005              MOVB *R0+,*R1+
006              DEC  R2
007              JGT  $-4
008       $END                       END OF DEFN
009       *...
010              MOVE A,B,20
011       *...
012       NEXT   MOVE X,Y,10
013       *...
014       A      BSS  20
015       B      BSS  20
016       X      BSS  10
017       Y      BSS  10
018              END
```

At this point you should enter and assemble the above example using the RLG options. In the listing you can see that "A" was substituted for "&P1" in the first generation for MOVE and that "X" was substituted in the second generation. Note also that the label coded on line 12 is substituted for "&P0".

Now our macro has become less trivial and considerably more useful. It is still a simple use of macros. Let's take our macro one step further. Suppose that the length of the move in line 10 was calculated by some other part of the program and was stored at "LNGT". Then we would want the macro to generate the statement

```
MOV  @LNGT,R2
```

instead of

```
LI   R2,20
```

for line 10, but to generate the same code as before for line 12. The following example accomplishes this.

```
Line/Pos  12345678901234567890123456789012345678901234567890
001       $MACRO MOVE              BEGINNING OF DEFN
002       &P0    LI   R0,&P1         &P1 IS FROM
003              LI   R1,&P2         &P2 IS TO
004       $IF    '&P3(1.1)',EQ,'@',GENMOV
005              LI   R2,&P3         &P3 IS LENGTH
006       $GOTO  COMMON
007       $LABEL GENMOV
008              MOV  &P3,R2         &P3 IS @LENGTH
009       $LABEL COMMON
010              MOVB *R0+,*R1+
011              DEC  R2
012              JGT  $-4
013       $END                       END OF DEFN
014       *...
015              MOVE A,B,@LNGT
016       *...
017       NEXT   MOVE X,Y,10
018       *...
019       A      BSS  20
020       B      BSS  20
021       X      BSS  10
022       Y      BSS  10
023       LNGT   BSS  2
024              END
```

Again, you should enter and assemble this example.

This example shows how macros are like programs that generate assembler statements (analogy number 4). At line 4 in the macro definition, the $IF macro directive tests if the first character of the third operand is an at sign (indicating that the number of bytes to move is in storage). If it is an at sign, the MOV instruction at line 8 is generated, otherwise, the LI at line 5 is generated and the $GOTO directive causes generation to continue with the code common to both options.

Line 4 uses "substring notation". The bracketed numbers following &P3 tell the assembler to substitute only part of the third operand. Substring notation is similar to the BASIC SEG$(&P3,1,1). Note, however, that a period is used between the two numbers.

We now have a fairly complex macro that could be useful in many different programs. You could at this point put the definition (lines 1 to 13) in your macro library and then use the MOVE macro in all your programs just as though it were an ordinary operation code.

Let's not stop yet though. Let's extend the macro some more. First, read the description of the MOVBL macro given earlier in the manual. Now, modify your MOVE macro to function like MOVBL. Give it some thought before you peek at the definition given in the macro library on the assembler disk. Try assembling a few MOVBL macros (with the RGL options) and see what code is generated.

Disk 50. Contents of file ASMUSER

RAG SOFTWARE

TI-99/4A MACRO ASSEMBLER

Version7

USER'S GUIDE

CONTENTS

September 1988

INTRODUCTION

The assembler reads a source program, written in assembler language, and translates it into an object program, in machine language. The source program statements are read from the "source file" which may be augmented by one or more "copy files". The object program is written to the "object file". During assembly, an "object listing file" can be created which shows the object code generated along with the source statements.

Before the object program can be executed it must be loaded into the computer's memory. The Assembler does not execute programs, it simply translates them from assembler language to machine language.

Availability

This package is being made available via the Fairware concept. If you are using the package, send a donation to:

R. A. Green
1032 Chantenay Drive
Gloucester, Ont. Canada
K1C 2K9

And, at the same time, distribute complete copies of the package to your friends.

Getting Started

Before you do anything else, make a working copy of the Assembler distribution disks. Save the distribution disks as backup in case your working copy is damaged. The assembler is distributed on three single sided single density disks. The disks contain the following files.

RAGASM/A Part A

ASMMREF	Macro Reference
ASMMREF1	Continued
ASMUSER	This User's Guide
ASMV7	Version 7 Fixes/Features
LOAD	XB Loader for RAGASM
RAGASM	The Assembler (Segment 1)
RAGASN	The Assembler (Segment 2)
RAGINSASM	Assembler installation program
RAGMAC	Macro definitions
XRAGINSASM	XB Loader for RAGINSASM

RAGASM/B Part B

ASM9900	9900 Reference
ASM99001	Continued
SRCRAGIO	Source for RAGIO subroutine
SRCRAGDIS	Source for RAGDIS subroutine

RAGASM/C Part C

ASMAREF	Assembler Language Reference
ASMAREF1	Continued
ASMAREF2	Continued
ASMAREF3	Continued
ASMAREF4	Continued

Assembler Installation

The assembler is designed to work with various loaders and printers, and to produce a compact listing to reduce print time and paper usage. In order to do this you must "install" the Assembler with options suitable to your environment. The RAGINSASM (E/A Option 5) program or the XRAGINSASM (XBASIC) program on the distribution disk will prompt you for the values for your environment and then modify the Assembler on disk so that it will use your values.

In order to produce a compact listing and still have the listing contain full information, the printer should be set up to print in elite mode at 8 lines per inch. Most printers can do this although it is not required. To accomplish this printing, the assembler will send a "setup sequence" of control characters to the printer before printing the first line of the listing, and will send a "reset sequence" of control characters to the printer after printing the last line of the listing. The assembler will truncate any lines that are too long for the printer and will count the number of lines per page. The length of a line and the number of lines per page must be coordinated with the way your printer is set up. The Assembler as distributed is set up for a GEMINI 10X printer.

During entry of your default values, press **ENTER** to proceed to the next prompt or press **BACK (FCTN 8)** to begin again at the beginning. Typing an asterisk (*) as the first character of the line indicates that the corresponding default value is not to be changed. A response of all blanks or of just pressing **ENTER** indicates that there is to be &no default value. Each of the prompts is described below.

Installation Drive Number

Enter the number of the disk containing the original Assembler modules. During installation the Assembler module, RAGASN, is loaded from this drive and then saved back to the drive. The default installation drive is 1 (i.e. DSK1).

Printer Name

Enter the device/filename for your printer. If no printer name is entered, then the Assembler will prompt for the printer name each time it is run. The distributed printer name is "PIO".

Printer Setup Sequence

Enter the printer control codes required for making your printer print the assembler listing in the mode you want. For example: in elite mode at 8 lines/inch, or in compressed mode at 8 lines/inch. Enter the data in hexadecimal (2 digits per byte). You will have to refer to your printer manual to determine the proper codes for your printer. The distributed setup sequence is "1B42021B30", which makes the GEMINI 10X print in elite mode at 8 lines per inch.

Printer Reset Sequence

Enter the printer control codes required to make your printer revert to its normal printing mode. This sequence should contain a form feed to position the printer at the top of the page. Most printers have a single "reset" code. Enter the data in hexadecimal. You will have to refer to your printer manual to determine the proper codes for your printer.

The distributed reset sequence is "122764", which is form feed and reset for the GEMINI 10X.

Lines Per Page

Enter, in decimal, the number of lines to print before a form feed is issued. The number of lines per page depends upon your paper size and the printer setup. For example, enter 75 for 11" paper at 8 lines/inch, or 55 for 11" paper at 6 lines/inch. You must allow for the top and bottom margin on the page. The distributed value is 75 lines per page.

Maximum Line Length

Enter, in decimal, the maximum number of characters to be printed per line. The Assembler can print up to 116 characters on a line. All lines will be truncated to the length you specify. The value specified depends upon the capability of your printer and on the way your printer is set up. Most printers can print an 80 character line in standard mode and a 96 character line in elite mode. The distributed value is 96 characters per line.

Default Options

Enter up to 6 one letter option codes that are to be used as the defaults when no options are entered during an assembler run. The distributed default options are "RLX".

Type of Return (RT or BLWP)

The Assembler can exit in two ways. First, it can return to the loader (or cartridge) that loaded it. This is what is done if "RT" is specified. This is the normal return for the Editor/Assembler Cartridge Option 5 and the TI-Writer Cartridge Option 3. Second, it can execute a "BLWP @0" which causes the 4A to reset. This is done if "BLWP" is specified. Some loaders are not prepared for programs to return (i.e. a popular RAM DISK LOADER), the BLWP return will suit these loaders. The distributed value is RT.

RUNNING THE ASSEMBLER

The RAG SOFTWARE Macro Assembler is independent of the module used to run it. It has been run using Option 3 of TI-Writer, Option 5 of Editor/Assembler, Option 1 of GRAM Kracker and Extended BASIC. Once loaded, the Assembler can process a batch of source files.

Assembler Prompts

The Assembler will prompt you for the macro definition file name, the source program file name, the object file name, the printer file name, the assembly identification information, and the options for this assembly. When an assembly is completed, the Assembler will ask if you want to assemble another program. Note that all device/file names are specified in the usual form.

MACROS:

Enter the device and file name of your macro library. A null entry (i.e. just press **ENTER**) indicates that no macro library file is required for this assembly. The file name of the supplied macro library is RAGMAC. When you press **ENTER**, the Assembler will read the macro definitions into memory before continuing with the next prompt. Once read, the macro file is no longer required so that your macro library could be on a separate disk.

An asterisk can be entered as the macro file name to indicate that the macros are to be read from RAM in the module plugged into the console (loaded there in a previous assembly), or when doing other than the first of a batch of assemblies to indicate that the macros from the previous assembly are to be used.

SOURCE:

Enter the device and file name of your assembler source statements.

OBJECT:

Enter the device and file name into which your object program is to be written.

PRINTER:

Enter the device and file name for the assembler listing file. This prompt occurs only if there was no default printer specified when the assembler was installed.

ID/DATE:

Enter up to 9 characters of identification for this assembly. The information entered appears in three places during the assembly: on the listing heading line, on the last record of the object file, and as the value of the &S4 system macro symbol. As the prompt suggests, today's date is probably the most useful identification, then both the listing and the object file will be dated.

OPTIONS:

Enter the one letter option codes. Except for the "&" option, the codes may be entered in any order. The option codes are:

C Output the object file in compressed format.

F Show full assembled data in the listing. If this option is not specified then a maximum of 6 bytes of data is shown on the listing (a single line) for BYTE, DATA, TEXT and STRI statements. The full data is, of course, always assembled into the object file.

G Include macro generated statements in the listing. The macro generated statements are identified in the listing by a plus sign following the statement number.

L Produce a listing of source statements.

M Include macro directives in the listing. Macro directives are identified in the listing by a minus sign preceding the statement. R - define the register symbols R0 - R15.

S Produce a symbol table listing.

X Produce a cross-reference listing.

Y Produce a cross-reference listing of all symbols except the register symbols.

4 Use the 4K RAM at address >7000 for storage of macros (i.e. Mini Memory RAM).

8 Use the 8K RAM at address >6000 for storage of macros (i.e. Super Space or GRAM KRACKER RAM).

& Set the system macro symbol &S0 to the remainder of the options line.

MORE?

If you wish to continue with this batch of assemblies, enter "Y" and the Assembler will reinitialize itself and prompt as above for the next program to be assembled. Any other entry will cause the assembler to terminate.

If during prompting you wish to restart from the beginning, simply press **BACK**. Note, however, once the macro file has been read BACK will only return you to the "SOURCE" prompt.

The Object File

The object file output by the Assembler contains identification so that the source, object and listing can be tied together. First, the source file name is placed on the first object record, which also contains the IDT information. Second, the last record of the object file (the colon record) contains the Assembler name/version, and the ID/DATE information that was entered.

The machine language object file created by the Assembler is a FIXED 80 file containing "tagged object" in either compressed or uncompressed format. The tagged object contains fields beginning with a tag followed by data. The tag identifies the type of data in the field. The first tag in an object program is either "0", which indicates that the object code is in uncompressed format, or ">01", which indicates that the object code is in compressed format.

In uncompressed format, each two byte data field is represented by four hexadecimal digits. In compressed format, each two byte data field contains the actual two bytes of data.

The following table lists the tags used and gives the content of the data field that follows the tag.

TAG	*CONTENTS OF DATA FIELD*
>01	2 Bytes, size of relocatable portion of program
	8 Characters, text from the IDT Directive
0	2 Bytes, size of relocatable portion of program
	8 Characters, text from the IDT Directive
1	2 Bytes, Entry point in absolute code
2	2 Bytes, Entry point in relocatable code
3	2 Bytes, REF chain in relocatable code
	6 Characters, REF name
4	2 Bytes, REF chain in absolute code
	6 Characters, REF name
5	2 Bytes, DEF value in relocatable code
	6 Characters, DEF name
6	2 Bytes, DEF value in absolute code
	6 Characters, DEF name
9	2 Bytes, absolute location counter
A	2 Bytes, relocatable location counter
B	2 Bytes, absolute code
C	2 Bytes, relocatable code
F	No data, end of object record

The last record of an object file begins with a colon. It contains only information on the Assembler name, version and the ID/DATE information.

Table Sizes

The Assembler builds several tables during an assembly. Each of the tables is described below.

The Symbol Table is used to save all symbols, macro names and operation codes defined in the assembly. Each entry in the table is 12 bytes. Two entries are made for each REF symbol. The symbol table is built in high memory from the end of the assembler code up to >F600, the maximum size is approximately 1,330 entries.

The Cross Reference Table is used to accumulate the references to each symbol in the assembly. The Y option excludes the register symbols from cross referencing. Each entry in the table is 4 bytes. The cross reference table is built in VDP RAM, the maximum size is approximately >29C0 bytes giving a maximum of 2,672 entries.

The Macro Definition Table is used to save all macro definitions encountered in the macro file or the source file. All the lines of the macro definitions are stored as variable length strings with the length byte preceding the statement. The macro table is built in VDP RAM, the maximum size is approximately >29C0 bytes providing enough space to hold the equivalent to a 40 sector macro file. The macro file will be moved to module RAM if the 4 or 8 option is specified, or will be moved to High Memory after pass 1 if there is enough space after the symbol table is built. If the macro table cannot be moved to CPU RAM then it must share the VDP RAM with the cross reference table, limiting the size of the cross reference table.

The OBJREC Directive Table is used to save the BEFORE text during pass 1 and to save the AFTER text during pass 2. Each text entry is stored as a variable length string with the length byte preceding the text. The table size is >037F bytes, providing enough space to hold about 20 40-byte object records.

DIAGNOSING ASSEMBLER FAILURES

Every attempt has been made to insure that the Assembler has no bugs, however, in every complex program the possibility of bugs always exists. In addition, bugs may also exist in the operating system facilities used by the Assembler. There are three possible type of failures.

1. The Assembler completes normally, but some instruction or data was assembled incorrectly.

2. The Assembler completed, but the listing or disk files were incomplete or in error.

3. The Assembler did not complete and/or the system required rebooting.

In all three cases, the first thing to do is to correct all source errors that were found by the Assembler. The Assembler can really only correctly assemble correct programs, although it tries to diagnose incorrect statements. As the old saying goes: "Garbage in equals Garbage out".

In the first case above, you should be sure that you understand what should be assembled. The language supported is fully described in the "Assembler Language Reference" document supplied with the Assembler. The compatibility statement made in that document is for information only and not as a definition of the language supported. If the EQUV assembler directive and/or macro definitions which test which pass the Assembler is in are used, these should be checked carefully.

In the second and third cases above, either the Assembler or the operating system may be suspect. In these cases, you should reduce the dependency of the assembler on the operating system. This can be done by not using any of the listing options.

Finally, all bugs discovered as well as any usability problems should be reported to:

RAG SOFTWARE
R. A. Green
1032 Chantenay Dr.
Gloucester, Ont.
CANADA K1C 2K9

If possible, a disk with the source program that can not be assembled should be sent. This will make finding the bug easier. If a disk is sent, the material on the disk will *only* be used for finding the bug and will be returned with a corrected version of the assembler.

TECHNICAL MATERIAL

The documentation distributed with the Assembler consists of three manuals:

ASMAREF	Assembler Language Reference
ASMMREF	Macro Reference and Tutorial
ASM9900	9900 CPU Reference

which can be printed with the TI-Writer Formatter.

More information on the hardware and programming the computer can be obtained in other publications such as those listed below.

1. *TMS9900 16-Bit Microcomputer Preliminary Data Manual*, Texas Instruments, Inc., 1981

2. Texas Instruments Home Computer Editor/Assembler, Texas Instruments, Inc., 1981

3. *Introduction to Assembly Language for the TI Home Computer*. Ralph Molesworth, 1983 Steve Davis Publishing P.O. Box 190831 Dallas, Texas 75219 ISBN 0-911061-01-0.

4. *Learning TI-99/4A Home Computer Assembly Language Programming*. Ira Mc Comic, 1984 Wordware Publishing, Inc. Plano, Texas 75074 ISBN 0-13-527862-7

5. *Fundamentals of TI-99/4A Assembly Language*. M. S. Morley, 1984 Tab Books, Inc. Blue Ridge Summit, PA 17214 ISBN 0-8306-1722-1

Disk 50. Contents of file ASMV7

RAG SOFTWARE Macro Assembler
Version 7, September 1988

BUGS FIXED.

1. A zero length source line caused the Assembler to crash.
2. The same bug was in the RAGIO subroutine.

NEW FEATURES.

1. All new documentation
   a. User's Guide
   b. Assembler Language Reference
   c. Macro Descriptions and Macro Writing
   d. 9900 CPU Reference

2. An installation program to set your default values into the Assembler. No more disk sector patching. New installation option to make the Assembler exit via

```
BLWP  @0
```

3. Printer file may be on disk.

4. Assembler restarts for a batch of assemblies, optionally reusing macro definitions in memory.

5. Macro definitions moved to CPU RAM if possible to allow a larger cross reference table.

6. Two new characters, underscore and percent, can be used as the second or following characters in symbol names.

7. Two new assembler directives. STRI for defining strings. EQUV for redefining symbols.

8. New macro directive, $OPCODE, allows definition of new machine instructions (i.e. those for the 9995).

9. New notation for strings, they may be coded in hexadecimal.

NEW MACROS

1. CALL — Call subroutines with parameter list.
2. RCALL — Call subroutines with parameters in registers 0, 1 and 2.
3. LDB — Load byte value into a register.
4. SETV — Set VDP address.
5. IF — Expanded to allow a literal, which generates a Compare Immediate instruction.
6. SETSW — Set switch ON or OFF.
7. IFSW — Test switch value.

Disk 50. Contents of file RAGMAC

```
$MACRO BE
$SET &L1,6
$IF '&P1(1.1)',EQ,'@',G
$SET &L1,4
$LABEL G
&P0 JNE $+&L1
 B &P1
$END
$MACRO BEEP
&P0 BLWP @RAGBEP
$IF '&P1',EQ,'NOWAIT',N
 LIMI 2
 MOVB @>83CE,@>83CE
 JNE $-6
 LIMI 0
$LABEL N
$END
$MACRO BNE
$SET &L1,6
$IF '&P1(1.1)',EQ,'@',G
$SET &L1,4
$LABEL G
&P0 JEQ $+&L1
 B &P1
$END
$MACRO CLOSE
$IF '&P1(1.1)',NE,'@',R
&P0 BLWP @RAGCLO
 DATA &P1(2)
$EXIT
$LABEL R
&P0 MOV &P1,@$+8
 BLWP @RAGCLO
 DATA 0
$END
$MACRO DCB
$SET &L1,0
$SET &L2,0
$SET &L3,0
$SET &L9,0
$SET &L8,'&P1'
$GOTO P
$LABEL R1
$SET &L8,'&P2'
$GOTO P
$LABEL R2
$SET &L8,'&P3'
$LABEL P
```

```
$IF '&L8',NE,'',&L8(1.2)
$LABEL R
$SET &L9,&L9+1
$GOTO R&L9(5)
$LABEL PA
$SET &L1,'&L8(5)'
$GOTO R
$LABEL EN
$SET &L2,'&L8(5)'
$GOTO R
$LABEL ER
$SET &L3,'&L8(5)'
$GOTO R
$LABEL R3
&P0 DATA 0,&L1,0,0,0,&L2,&L3,0,0
$END
$MACRO GET
$IF '&P1(1.1)',NE,'@',D
$IF '&P2(1.1)',NE,'@',B
&P0 BLWP @RAGGET
 DATA &P1(2),&P2(2)
$EXIT
$LABEL B
&P0 MOV &P2,@$+10
 BLWP  @RAGGET
 DATA  &P1(2),0
$EXIT
$LABEL D
$IF '&P2(1.1)',NE,'@',C
&P0 MOV &P1,@$+8
 BLWP @RAGGET
 DATA 0,&P2(2)
$EXIT
$LABEL C
&P0 MOV &P1,@$+12
 MOV &P2,@$+10
 BLWP @RAGGET
 DATA 0,0
$END
$MACRO HONK
&P0 BLWP @RAGHON
$IF '&P1',EQ,'NOWAIT',N
 LIMI 2
 MOVB @>83CE,@>83CE
 JNE $-6
 LIMI 0
$LABEL N
$END
$MACRO IF
$IF '&P3(1.1)',EQ,'=',A
&P0 C &P1,&P3
```

```
 J&P2 &P4
$EXIT
$LABEL A
&P0 CI &P1,&P3(2)
 J&P2 &P4
$END
$MACRO IFB
&P0 CB &P1,&P3
 J&P2 &P4
$END
$MACRO INPUT
$IF '&P2(1.1)',EQ,'=',S
&P0 MOV &P2,R0
$GOTO C
$LABEL S
&P0 LI R0,&P2(2)
$LABEL C
$IF '&P1(1.1)',NE,'@',R
 BLWP @RAGKIN
 DATA &P1(2),&P3
$EXIT
$LABEL R
 MOV &P1,@$+8
 BLWP @RAGKIN
 DATA 0,&P3
$END
$MACRO MOVBL
$SET &L1,'&P1'
$SET &L2,'&P2'
$SET &L3,'&P3'
&P0 EVEN
$IF '&P1(1.1)',NE,'@',A
$SET &L1,'*R0+'
 LI R0,&P1(2)
$LABEL A
$IF '&P2(1.1)',NE,'@',B
$SET &L2,'*R1+'
 LI R1,&P2(2)
$LABEL B
$IF '&P3(1.1)',EQ,'@',C
$IF '&P3(1.1)',NE,'=',G
$SET &L3,'R2'
 LI R2,&P3(2)
$GOTO G
$LABEL C
$SET &L3,'R2'
 MOV &P3,R2
$LABEL G
 MOVB &L1,&L2
 DEC &L3
```

```
 JNE $-4
$END
$MACRO LDB
&P0 MOVB &P1,&P2
 SRL &P2,8
$END
$MACRO OPEN
$IF '&P1(1.1)',NE,'@',R
&P0 BLWP @RAGOPE
 DATA &P1(2)
$EXIT
$LABEL R
&P0 MOV &P1,@$+8
 BLWP @RAGOPE
 DATA 0
$END
$MACRO PAB
$SET &L1,0
$SET &L2,0
$SET &L3,0
$SET &L4,0
$SET &L5,0
$SET &L6,0
$SET &L7,0
$SET &L9,0
$SET &L8,'&P1'
$GOTO P
$LABEL R1
$SET &L8,'&P2'
$GOTO P
$LABEL R2
$SET &L8,'&P3'
$GOTO P
$LABEL R3
$SET &L8,'&P4'
$GOTO P
$LABEL R4
$SET &L8,'&P5'
$GOTO P
$LABEL R5
$SET &L8,'&P6'
$GOTO P
$LABEL R6
$SET &L8,'&P7'
$GOTO P
$LABEL R7
$SET &L8,'&P8'
$LABEL P
$IF '&L8',NE,'',&L8(1.2)
$LABEL FI
$LABEL SE
```

```
$LABEL DI
$LABEL UP
$SET &L9,&L9+1
$GOTO R&L9(5)
$LABEL VA
$SET &L1,16
$GOTO FI
$LABEL IN
$IF '&L8(1.3)',EQ,'INP',I
$SET &L2,8
$GOTO FI
$LABEL I
$SET &L3,4
$GOTO FI
$LABEL OU
$SET &L3,2
$GOTO FI
$LABEL AP
$SET &L3,6
$GOTO FI
$LABEL RE
$SET &L4,1
$GOTO FI
$LABEL RL
$SET &L5,'&L8(4)'
$GOTO FI
$LABEL BU
$SET &L6,'&L8(4)'
$GOTO FI
$LABEL SO
$SET &L7,'&L8(4)'
$GOTO FI
$LABEL FD
$SET &L0,'&L8(4)'
$GOTO FI
$LABEL R8
&P0 DATA &L1+&L2+&L3+&L4,&L6
 BYTE &L5,0,0,0,&L7
$IF '&L0',EQ,'',N
 BYTE P&S1-$-1
 TEXT '&L0'
P&S1 BSS 0
$EXIT
$LABEL N
 BSS 29
$END
$MACRO PABEQU
&P0OP EQU &P1+0
&P0FL EQU &P1+1
&P0BU EQU &P1+2
```

```
&P0RL EQU &P1+4
&P0CC EQU &P1+5
&P0RN EQU &P1+6
&P0SL EQU &P1+6
&P0SO EQU &P1+8
&P0ST EQU &P1+8
&P0NL EQU &P1+9
&P0FD EQU &P1+10
$END
$MACRO PRINT
$IF '&S2',EQ,1,S
$IF '&P1(1.1)',NE,'@',B
$IF '&P2(1.1)',NE,'@',A
&P0 BLWP @RAGDS1
 DATA &P2(2),&P1(2)
$EXIT
$LABEL A
&P0 MOV &P2,@$+8
 BLWP @RAGDS1
 DATA 0,&P1(2)
$EXIT
$LABEL B
$IF '&P2(1.1)',NE,'@',C
&P0 MOV &P1,@$+10
 BLWP @RAGDS1
 DATA &P2(2),0
$EXIT
$LABEL C
&P0 MOV &P1,@$+14
 MOV &P2,@$+8
 BLWP @RAGDS1
 DATA 0,0
$EXIT
$LABEL S
$IF '&P1(1.1)',NE,'@',D
&P0 BLWP @RAGSCL
 DATA &P1(2)
$EXIT
$LABEL D
&P0 MOV &P1,@$+8
 BLWP @RAGSCL
 DATA 0
$END
$MACRO PUT
$IF '&P1(1.1)',NE,'@',B
$IF '&P2(1.1)',NE,'@',A
&P0 BLWP @RAGPUT
 DATA &P1(2),&P2(2)
$EXIT
$LABEL A
&P0 MOV &P2,@$+10
```

```
 BLWP @RAGPUT
 DATA &P1(2),0
$EXIT
$LABEL B
$IF '&P2(1.1)',NE,'@',C
&P0 MOV &P1,@$+8
 BLWP @RAGPUT
 DATA 0,&P2(2)
$EXIT
$LABEL C
&P0 MOV &P1,@$+12
 MOV &P2,@$+10
 BLWP @RAGPUT
 DATA 0,0
$END
$MACRO SCB
$SET &L1,0
$SET &L2,736
$SET &L3,32
$SET &L4,736
$SET &L5,0
$SET &L6,0
$SET &L9,0
$SET &L8,'&P1'
$GOTO P
$LABEL R1
$SET &L8,'&P2'
$GOTO P
$LABEL R2
$SET &L8,'&P3'
$GOTO P
$LABEL R3
$SET &L8,'&P4'
$GOTO P
$LABEL R4
$SET &L8,'&P5'
$GOTO P
$LABEL R5
$SET &L8,'&P6'
$GOTO P
$LABEL R6
$SET &L8,'&P7'
$GOTO P
$LABEL R7
$SET &L8,'&P8'
$GOTO P
$LABEL R8
$SET &L8,'&P9'
$LABEL P
$IF '&L8',NE,'',&L8(1.2)
```

```
$LABEL R
$SET &L9,&L9+1
$GOTO R&L9(5)
$LABEL TO
$SET &L1,'&L8(5)'
$GOTO R
$LABEL BO
$SET &L2,'&L8(5)'
$GOTO R
$LABEL LE
$SET &L3,'&L8(5)'
$GOTO R
$LABEL IN
$SET &L4,'&L8(5)'
$GOTO R
$LABEL SO
$SET &L5,'&L8(4)'
$GOTO R
$LABEL PR
$SET &L6,&L6+128
$GOTO R
$LABEL SC
$SET &L6,&L6+72
$GOTO R
$LABEL TE
$SET &L6,&L6+32
$GOTO R
$LABEL RM
$SET &L6,&L6+16
$GOTO R
$LABEL SP
$SET &L6,&L6+8
$GOTO R
$LABEL SI
$SET &L6,&L6+64
$GOTO R
$LABEL ML
$SET &L6,&L6+4
$GOTO R
$LABEL R9
&P0 DATA &L1,&L2,&L3,&L4
 BYTE &L5,&L6
$END
$MACRO SETV
$SET &L1,'@>8C02'
$IF '&P2',EQ,'',NOREG
$SET &L1,'*&P2'
$LABEL NOREG
$IF '&P1(1.1)',EQ,'@',A
$IF '&P1(1.1)',EQ,'*',B
$IF '&P1(1.1)',EQ,'=',C
```

```
&P0 SWPB &P1
 MOVB &P1,&L1
 SWPB &P1
 MOVB &P1,&L1
 &P3
$EXIT
$LABEL A
&P0 MOVB &P1+1,&L1
 NOP
 MOVB &P1,&L1
 &P3
$EXIT
$LABEL B
&P0 MOVB @1(&P1(2)),&L1
 NOP
 MOVB &P1,&L1
 &P3
$EXIT
$LABEL C
&P0 LI R0,&P1(2)
 SWPB R0
 MOVB R0,&L1
 SWPB R0
 MOVB R0,&L1
 &P3
$END
$MACRO IFSW
&P0 ABS &P1
$GOTO &P2
$LABEL ON
 JNE &P3
$EXIT
$LABEL OFF
 JEQ &P3
$END
$MACRO SETSW
$GOTO &P2
$LABEL ON
&P0 SETO &P1
$EXIT
$LABEL OFF
&P0 CLR &P1
$END
$MACRO CALL
&P0 BLWP  &P1
$IF &S2,EQ,1,D
 DATA &P2
$IF &S2,EQ,2,D
 DATA &P3
$IF &S2,EQ,3,D
```

```
 DATA &P4
$IF &S2,EQ,4,D
 DATA &P5
$LABEL D
$END
$MACRO RCALL
&P0 EVEN
$IF &S2,EQ,1,D
 LI &P2(1.2),&P2(4)
$IF &S2,EQ,2,D
 LI &P3(1.2),&P3(4)
$IF &S2,EQ,3,D
 LI &P4(1.2),&P4(4)
$LABEL D
 BLWP &P1
$END
```

Disk 51. RAG Assembler

Version: 6.0 *Author*:
Requires: *Language*: *Updated*: 3/1/88

See disk 50

```
dskdir. v2.0. 12-dec-96
Disk name               = RAGASM7-2
Sectors total           = 360
Sectors used            = 298
Sectors available       = 60
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >005 -README       3 DIS/VAR 80   >146 002
002 >004 ARCHIVER     33 PROGRAM      >126 032
003 >002 RAG/2       112 INT/FIX128   >022 111
004 >003 RAG/3       150 INT/FIX128   >091 149
```

Disk 51. Contents of file -README

In order to fit the RAG Assembler Version 7.0 onto two disks we had to Archive the contents of this disk. Simply run the ARCHIVER file on this disk and use the extract option to unpack the RAG/2 and RAG/3 files onto blank floppies. We apologize for any inconvenience, however it sure beats having to buy a third disk!

The Boston Computer Society
TI-99/4A User Group

Disk 52. JPH Games Disk

Version:
Requires: XB

Author: J. Peter Hoddie
Language: AL, XB

Updated: 08/17/86

Asteroids and Snake in assembly. J. Freddie Frog, Space Battle, and several others in Extended BASIC. Programs from the deep dark past of J. Peter Hoddie.

```
dskdir. v2.0. 12-dec-96
Disk name               = JPH_GAMES
Sectors total           = 360
Sectors used            = 245
Sectors available       = 113
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00a -README      32 DIS/VAR 80 Y >0ef 031
002 >003 ASTER        29 PROGRAM    Y >03e 028
003 >005 FISH         25 PROGRAM    Y >061 024
004 >006 FROG         34 PROGRAM    Y >079 033
005 >007 KONG         40 PROGRAM    Y >09a 039
006 >009 LOAD          9 PROGRAM    Y >0e7 008
007 >004 LOADER        8 DIS/FIX 80 Y >05a 007
008 >002 SNAKE        29 PROGRAM    Y >022 028
009 >008 SPACEWAR     39 PROGRAM    Y >0c1 038
```

Disk 52. Contents of file -README

JPH GAMES DISK

This disk is FAIRWARE. Any individual or non-profit User Group may distribute this disk for a reasonable (no more than $5) copy fee. A considerable amount of work went into producing these games. They were originally intended for commercial release. When TI left the home computer market, these plans were shelved. Over two years later I dug them out, cleaned them up a bit and collected them on this disk. If you enjoy these programs please send a donation of up to $10 to the author:

J. Peter Hoddie
12 Paul Revere Road
Lexington, MA 02173

SNAKE

You control the snake with joystick number one or the arrow keys. You can select speeds and snake lengths from 1 to 9 with 1 being the easiest length and the most difficult speed. The game can be interrupted with **FCTN 9**. The object is to collect the blinking targets. While doing this you must avoid hitting the strange looking bouncing purple monsters, the trees, the edges, and yourself. If you try to move down while you are traveling up that is considered hitting yourself. There are also rising red diamonds. You can hit these while they are rising to score points but if you hit them when they are full grown you loose a life. There are about 10 screens and then they recycle at a more difficult level. This was my first major assembly program. The music is from Pictures at an Exhibition by Mussorgsky.

ASTEROIDS

This is a variant of asteroids written because I was sick and tired of Extended BASIC versions of asteroids on the TI. There is one minor difference. Instead of wrapping around the edges you and the asteroids bounce off them. Shots are absorbed by the edges. You can use joystick #2 to control the ship. Left and right to rotate, up to thrust, down to hyperspace, and fire to fire. The keyboard can also be used. **K** and **L** to rotate. **D** to fire, **F** to thrust, **W** to hyperspace. To start the game you must hit either the joystick button or a key on the keyboard. There is a maximum of five large asteroids on the screen at once. Hyperspace can be helpful in an emergency but there is no guarantee that you will be put in a safe location. The music that is played when you die is a VERY speeded up run from Beethoven's Tempest Sonata. (The graphics in this program could use some cleaning up. I just don't have the artistic talent. Anyone interested? — jph)

J. FREDDY FROG

This game is yet another version of Frogger for the TI. It is named after the President (Chairman of the Board?) of Texas Instruments, J. Fred Bucy. Anyhow, you use joystick #1 to move the frog. You must avoid cars on the highway and hop on logs and lilypads (which sometimes sink) to get across to the home base. Your score is only updated after a frog dies or reaches a home base. Scoring is as follows:

	Forward Hop	*Home Base*	*Finish Screen*	*Frogs to Finish Screen*
Beginner	5	75	500	2
Intermediate	10	150	1000	3
Master	15	225	1500	4

SPACE BATTLE

Player one uses joystick #1 and player two uses joystick #2 to control their ships. Player one starts in the upper left corner of the screen and player two starts in the lower right corner. Use the joystick button to fire. There are many options for this game and all should be self explanatory except for the "missile type" option. Directional missiles move in the direction that your ship is moving. Guided missiles move toward your enemy and are fired at a speed related to your distance from your enemy. You can have only one missile on the screen at a time, so until that one disappears you cannot fire again. You must avoid going near the sun as you will burn up. Occasionally you will hear a strange noise and your ship will be instantly teleported to another part of the screen. This means that your ship has encountered a hole in the fabric of space which has caused it to fall through. Once this hole has been used it closes up and moves to another point on the screen.

KLIMBING KONG

Use joystick #1 to control the man. To start the game press the fire button. As long as the man is blinking he cannot be killed. You must avoid the monsters and the portions of the screen that do not contain bricks on the ground. There are eight different screens. To complete a screen you must complete the objective specified at the top of the screen. Scoring is simple. You get 500 points for completing a screen, 5 points for jumping up and 15 points for jumping either left or right.

FISHY BUSINESS

Use joystick #1 to control your fish. Press the fire button to start the game. Your objective is to eat all the seaweed while avoiding the big fish. You only have 250 time units and they go fast, especially if you don't move. There are four different screens. Bonus points are awarded for completing a screen in less than the allotted time. If your fish is about to be eaten you can hold down the fire button to escape death. However you will lose points for doing this. Be careful though. If your score reaches zero because of this you lose a fish anyway. You start out with a certain amount of points equal to the level number minus one times 2000 at the start of the game. You get 10 points for each piece of seaweed you eat. Furthermore you get a bonus of 10 points for each remaining piece of seaweed at the end of a screen. On the first screen it costs 50 points to save your life, 100 on the second, 150 on the third, and so on. If you have more than 8000 points after any screen you will receive a free fish.

The first two programs on this disk (SNAKE and ASTEROIDS) are in assembly language. A fast loader has been created so that they load in 15 to 20 seconds instead of the usual 2 minutes that assembly language files of this length take to load.

I realize that FAIRWARE games have been less then a smash success in the past. I hope that with six games, including two in assembly language, that you will find this disk worth $10. I have added prescan to all the Extended BASIC programs to make them start up quickly so you don't have to wait around. The assembly loader was written for the same reason. If you have questions, comments or whatever, let me know. I have no current plans for an update but if the response is great enough and/or someone is interested in helping out I will consider an update.

By the way, for those of you who care, I am the co-director of the Boston Computer Society TI User Group and the author of that 22-page report on the Chicago '85 TI show as well as the reports on Myarc's new computer and Extended BASIC 2. For a software list of over 60 public domain programs, listing several publications, and membership information on the BCS send a self addressed stamped envelope to:

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Disk 53. Music Compiler

Version: 1.1 *Author*: Chris Morgan
Requires: XB *Language*: AL *Updated*: 08/17/86

Allows you to create musical compositions and then play them while you program or while your program is running! Comes with several excellent demos.

```
dskdir. v2.0. 12-dec-96
Disk name               = COMPILER
Sectors total           = 360
Sectors used            = 347
Sectors available       = 11
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BANNER       18 DIS/VAR254   >022 017
002 >003 BSCSUP       15 DIS/FIX 80   >033 014
003 >004 COMPILER     31 DIS/FIX 80   >041 030
004 >005 COMPLR-DOC    9 PROGRAM      >05f 008
005 >006 ENTERTAINR   56 DIS/VAR254   >067 055
006 >007 IN-MILL      19 DIS/VAR254   >09e 018
007 >008 INVENTIONF   15 DIS/VAR254   >0b0 014
008 >009 LOAD          2 PROGRAM      >0be 001
009 >00a LOADER        6 DIS/FIX 80   >0bf 005
010 >00b MAINSCREEN   10 DIS/VAR254   >0c4 009
011 >00c MAPLELEAF    26 DIS/VAR254   >0cd 025
012 >00d MUSICBOX     12 DIS/VAR254   >0e6 011
013 >00e NEVERONSUN   21 DIS/VAR254   >0f1 020
014 >00f PUPPYTOWN    23 DIS/VAR254   >105 022
015 >010 S             4 PROGRAM      >11b 003
016 >011 SS            2 DIS/VAR254   >11e 001
017 >012 VARIATIONS   13 DIS/VAR254   >11f 012
018 >013 VENUS-RACE   22 DIS/VAR254   >12b 021
019 >014 W-BOOGIE     28 DIS/VAR254   >140 027
020 >015 XBLOAD       15 DIS/FIX 80   >15b 013 >016 001
```

Disk 54. RLE Graphics

Version:
Requires: XB, EA

Author: Travis Watford
Language: AL

Updated: 08/17/86

Program which allows use of VIDTEX bitmap graphics on the TI. Files can be transferred via modem and then displayed, printed, or loaded in Graphx or TI-Artist. Many sample screens on disk. This program is a must for anyone into TI graphics.

```
dskdir. v2.0. 12-dec-96
Disk name               = RLE-BCS
Sectors total           = 360
Sectors used            = 218
Sectors available       = 140
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >008 -README       6 DIS/VAR 80   >0de 005
002 >003 COSBY        12 DIS/VAR 80   >044 011
003 >004 ELVIRA       15 DIS/VAR 80   >04f 014
004 >005 HERRON       43 DIS/VAR 80   >05d 042
005 >006 PAMELA       32 DIS/VAR 80   >087 031
006 >007 PICTURE      57 DIS/VAR 80   >0a6 056
007 >002 RLE          35 DIS/FIX 80 Y >022 034
008 >009 WEATHER      18 DIS/VAR 80   >0e9 017
```

Disk 54. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain Disk #54
RLE Graphics

The main program on this disk is the RLE file. It loads with Editor/Assembler option #3. The program name is START. Once you have the program up and running, just give it a filename and it will show you the picture. To SAVE a picture to be loaded into Graphx (and thusly TI-Artist) hit **S** while the picture is on the screen. Use **P** to print the picture to an Epson compatible printer. Hit **ENTER** to load another picture. Use the other keys to change colors.

All the files on this disk besides RLE and -README are all pictures. Just for reference Herron is currently #1 on the FBI's most wanted list.

For your information this program decodes a picture stored in a format called VIDTEX. This format is supported by CompuServe. In the future it is hoped that this RLE-MAX program will be updated to allow for uploads of GRAPHX/TI-Artist pictures as well as downloads.

All pictures and the program itself come from CompuServe's TI Forum. They were provided to the BCS by Ken Van Tassell and Walt Howe. No documentation is currently available that's why I threw this file together.

J. Peter Hoddie
Co-director
BCS TIUG

Disk 55. Disk+Aid

Version:
Author: Don Thomsom
Requires: XB, EA
Language: AL
Updated: 08/17/86

One of the most complete and user-friendly disk sector editor programs ever written. Previously available for $20 from Thomson Software.

```
dskdir. v2.0. 12-dec-96
Disk name               = JPHBCS2
Sectors total           = 360
Sectors used            = 254
Sectors available       = 104
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 DISK+AID-O    9 DIS/FIX 80 Y >054 008
002 >004 DISKOBJ      51 PROGRAM    Y >022 050
003 >003 LOAD          3 PROGRAM    Y >05c 002
004 >005 MANUAL        2 DIS/VAR 80 Y >05e 001
005 >006 MANUAL1      65 DIS/VAR 80 Y >05f 064
006 >007 MANUAL2      40 DIS/VAR 80 Y >09f 039
007 >008 MANUAL3      38 DIS/VAR 80 Y >0c6 037
008 >009 MANUAL4      46 DIS/VAR 80 Y >0eb 045
```

Disk 55. Contents of file MANUAL

```
.PL 55
.IF DSK1.MANUAL1
.IF DSK1.MANUAL2
.IF DSK1.MANUAL3
.IF DSK1.MANUAL4
```

Disk 55. Contents of file MANUAL1

DISK + AID FOR THE TI-99/4A COMPUTER

Version 3.0

EVALUATION COPY

The most complete sector editor and data recovery program

For the TI-99/4A Home Computer

By

THOMSON SOFTWARE
1461 Beach Street
Muskegon, MI 49441-1099

This is an evaluation copy of DISK + AID. In several places of this manual you will notice the following message:

EVALUATION COPY — SECTION OMITTED

I feel that there is enough of the manual included to allow you to make a good evaluation of DISK + AID. There are many reasons why I did not include the entire manual on disk:

1. I would like to encourage more of our users to become distributors. I have priced DISK + AID very fairly and I feel it is a great bargain. I hope you will support the lower priced software by readily paying. Other software houses will be forced to take notice.

2. I feel that the honesty of our "REGISTERED" users should be rewarded. They are entitled to extra value and support which should be rightfully denied to those who do not pay.

3. The complete documentation will not fit on the disk with the program.

I hope that you will not be closed minded about omissions, and hope to have you on my list of "REGISTERED" contributors soon.

Anyone may obtain an evaluation copy of DISK + AID from a friend or club. After you have had the chance to use and evaluate the program in your own environment, you are trusted to either send in a contribution of to discontinue use of the program. In any case, you are encouraged to copy this program for evaluation by others.

DISK + AID is one of the most powerful, versatile and user friendly data recovery programs on the market today for the TI-99/4A Home computer. It is constantly being updated and improved at the request of users and discretion of the author. DISK + AID is protected under copyright. The author intends to prosecute anyone to the fullest extent of the law for making any claims toward ownership of DISK + AID.

TABLE OF CONTENTS

PART 1 - INTRODUCTION

Thank you for coming to THOMSON SOFTWARE for your utility software needs. I feel that you have made a good investment in a program that will be of value to you for a long time to come.

SECTION 1. Loading DISK + AID with Extended Basic

To load DISK+AID with EXTENDED BASIC, select EXTENDED BASIC from the main title screen with the program disk in drive 1. With the new speed loader, DISK+AID should load and automatically run in about 8 seconds. If you wish to load DISK+AID from XB, type the following line at command mode:

```
CALL INIT :: CALL LOAD("DSK1.DISK+AID-O") :: CALL LINK("START")
```

This will load the program and start running it for you.

SECTION 2. Loading DISK + AID with Mini-Memory

If you are running the program from Mini-Memory the first thing to do is reset the computer back to the master title screen with the Mini-Memory command module plugged in. Press any key to clear the title screen.

Now select option 1-LOAD AND RUN. The computer will ask you for the file name. At the file name prompt, type in DSK1.DISK+AID-O. The program will boot in and start running automatically.

SECTION 3. Loading DISK + AID with Editor/Assembler

Reset the computer to the master title screen with the Editor/Assembler command module plugged in. Press any key to clear the master title screen and then select ED/ASSEMBLER. Press option 3. LOAD AND RUN.

At the FILENAME prompt type in DSK1.DISK+AID-O and press **ENTER**. The program will boot in and start running automatically.

SECTION 4. I/O Error message decoding

There are error messages that may be encountered while running DISK + AID. These messages and their meanings are as follows:

EVALUATION COPY — SECTION OMITTED

SECTION 5. SINGLE SECTORING error message decoding

There are errors that may be encountered when performing single sector operations. These messages are described in the following pages.

EVALUATION COPY — SECTION OMITTED

PART 2 - OPERATING DISK + AID

SECTION 6. STARTING OUT

The first screen you will have is the credit/title screen. Press any key to continue. The next screen you will have will be the WARNING screen that informs you of the dangers of using a single sectoring program.

The next screen is the MAIN MENU. Displayed on this screen are all of the commands available to you. To make the program more user friendly, I have incorporated a single key stroke command entry feature. As soon as you press a key that represents a valid command you will be sent to that routine.

At the top of the screen is the status line. The status line displays the following items:

— The extreme left side of the line will have either a "C", "G", "V" or nothing if you are just entering the program. These letters represent the area of memory that was last accessed.

— The next thing will be the word DISK and a number after it. This number represents the disk drive that the program will go to for sector reading and writing.

— The next thing in the line will be either the current sector number or the memory address in CPU, VDP or GROM memory as indicated by the letter at the extreme left side of the screen. When you do a read or write operation, the sector number is pulled from this part of the status line.

*** WARNING ***

BE SURE YOU HAVE THE SECTOR AND DISK NUMBER YOU WANT TO READ OR WRITE ENTERED BEFORE EXECUTING THE READ OR WRITE SECTOR COMMAND.

If you have a "C" on the left side of the line and a >A000 in this part of the line and you press "C", the program will display 256 bytes of CPU memory starting at >A000.

EVALUATION COPY - SECTION OMITTED

— The last thing on the status line is the last mode you accessed. The more important commands will be displayed here when selected. Not all the commands will be displayed however.

I would first like to point out a couple of important considerations when using DISK + AID.

This program uses VDP RAM extensively. There is an 880 byte buffer for the ASCII buffer, 880 bytes for the HEX buffer, 960 bytes for the screen image table, and 256 bytes for the sector I/O buffer plus a PAB table. I will be mentioning mainly the ASCII and HEX buffers throughout the manual because of their importance.

The ASCII and HEX buffers are used to store the ASCII and HEX equivalents of a converted sector or memory block. The HEX buffer is especially important for SECTOR WRITE operations because the information that is in this buffer is what is written to the disk. The same goes for altering memory. If you accidentally go to alter memory when you don't want to and press **ENTER** from the working field, the current HEX buffer will be written out to memory. It is a good idea to stay away from all of these buffers. When explaining the different commands I will also tell you whether the operation affects these buffers or not.

Another important consideration is the middle of the status line. When you are performing a memory view operation and then want to perform a SECTOR operation the SECTOR number will automatically be updated and displayed on the status line. The same thing happens when going from a SECTOR operation to a MEMORY operation; the status line will automatically be updated and displayed.

When at the command mode make sure the ALPHA LOCK is depressed or the commands will not register.

SECTION 7. O-SELECTING AN OUTPUT DEVICE

When entering an output device name you may use the LEFT or RIGHT arrows to move the cursor where you want it. You can also use FCTN 3 to clear the device name from the screen and memory storage area. After you are finished typing in the output device name, press ENTER and you will be back to the command mode.

This command does not affect the VDP buffers.

SECTION 8. A-ALTER SECTOR OR MEMORY

The alter mode will enable you to change a sector of information or alter the contents of any CPU or VDP RAM in the computer system. The program features complete on screen editing in either ASCII or HEX. If you have the screen toggled to ASCII you will be editing in ASCII. If you have the screen toggled to HEX you will be editing in HEX.

You can enter this mode only if there is ASCII or HEX information being displayed on the screen.

After pressing "A", the cursor will automatically be moved to the upper left part of the field. At this point, you can use any of the arrow keys to position the cursor where you want it. The program will not let you exceed the boundaries of the field whether in ASCII or HEX.

If you entered this mode by accident and don't care to edit the current information being displayed on the screen, simply press **FCTN 4** to clear you out of the alter mode and put you back to the command mode.

There are a few of things to remember when editing information on the screen.

1. If you are editing a sector of information, the new data will not be written back to the disk unless you select **FCTN W**. This is the only way new data can be put back to the disk.

2. If you are editing memory, the new data will not be written out UNTIL you press **ENTER**. If you press **FCTN 4**, the contents you have altered on screen will not be written back to memory.

3. This editing feature is simply a screen editing feature and does not take effect until you press the correct keys. After you press either **FCTN 4** or **ENTER** you will be returned to the command select mode.

Editing the screen will affect both the ASCII and HEX buffers.

SECTION 9. B-BACK SECTOR

This command will allow you to decrement the sector number by one (1). When you press "B" you can see the sector number on the screen decrement by one unless it is zero, then it will stay at zero. This does not affect the read or write function until you select the correct commands to read or write.

The VDP buffers are not affected using this command.

SECTION 10. C-VIEWING CPU MEMORY

EVALUATION COPY — SECTION OMITTED

Disk 55. Contents of file MANUAL2

SECTION 11. D-SCREEN DUMP

Screen dumps can be performed on any screen at any time the cursor is flashing next to the word MODE on the status line. If there is no valid device name typed in you will be sent to the select output device prompt to enter a device name. The information that was on the screen at the time will be saved to and then returned after you press ENTER after inputting the device name.

The screen dump will work on any valid device except cassette.

After the screen dump is completed you will be returned to the command mode.

The VDP buffers are not affected.

SECTION 12. E-COMPARING SECTORS

EVALUATION COPY — SECTION OMITTED

SECTION 13. F-FORWARD SECTOR NUMBER

When you press the "F" key you will notice that the sector being displayed on the status line will be incremented by one. There is no upper limit other than >FFFF. At that point the counter will roll back around to 0000. If you increment to a sector number that is not valid for your system, then try to read or write that sector an error will occur and the screen will turn red.

You may press the "F" key any time the cursor is flashing next to the word MODE on the status line.

One thing to remember: when you use the "F" key to forward the sector count and you do a read operation or a write operation, the number is taken from this spot on the screen.

The VDP buffers are not affected during this operation.

SECTION 14. G-VIEWING GROM MEMORY

EVALUATION COPY — SECTION OMITTED

SECTION 15. H-CHANGING MEMORY ADDRESS

EVALUATION COPY — SECTION OMITTED

SECTION 16. I-DISPLAYING SCREEN BUFFER

When you press "I" you will display the last ASCII or HEX screen that was read in from memory or the disk.

If you have an ASCII screen of a disk sectors contents and decide you need to view the menu to refresh your memory, as soon as you are done with the menu press "I" and the ASCII screen you were viewing will be displayed once again on the screen just as you left it. The ASCII and HEX buffers in VDP RAM will remain unchanged until you alter them using the program or read in a new memory block or read a new disk sector. At that time the two buffers will be updated with the new information.

This command will not toggle the screen for you. You still have to toggle the screen using the "T" command. The only effect on the VDP buffers is that they will be displayed.

SECTION 17. M-MOVING A SECTOR

EVALUATION COPY — SECTION OMITTED

SECTION 18. N-SECTOR NUMBER

This command will enable you to manually enter the sector number you want right on the status line.

When you press "N" the most current sector number will be displayed and the cursor will appear in this field. After you enter the number, press **ENTER** and it will be your new number.

The VDP buffers are not affected during this operation.

SECTION 19. P-PRINTING SECTORS

If you enter this command and there is no device name entered, you will be returned to the enter output device mode to enter a device name.

When you press "P", you will be prompted for the starting sector number you want printed and the ending sector number you want printed.

When you press enter after selecting your last sector number, the program will start printing the sectors out. If you wish to PAUSE the printing process press any key. To restart the printing process press any key. If you wish to break out of the print sectors mode, simply press **FCTN 4**. You will be returned to the command mode.

If you enter this command by mistake, press **FCTN 4** to clear yourself back to the command mode.

To see a sample of a sector printout, turn to the back of the manual.

The VDP buffers are affected during this operation.

SECTION 20. Q-QUIT ING THE PROGRAM

To exit the program and get back to the TI title screen, simply press "Q"

SECTION 21. R-READ SECTOR

When you press "R", the program will take the current DISK and SECTOR number from the status line and read the sector in.

At this point the program will check to see if the screen update flag is set. If the "$" is on the status line, the sector information will be displayed in either HEX or ASCII depending on how you have the toggle flag set. If the "$" is not on the screen, nothing will be displayed.

After the command is completed, you will be returned to the command mode.

The VDP buffers are affected during this operation.

SECTION 22. S-SEARCH STRING

This mode of operation will enable you to search for a string up to 40 characters in either ASCII or HEX. The program will also allow you to turn screen update on or off for faster searching of the disk.

When you enter this mode you will be prompted for the sectors you want to search. The program will start the search at the first sector number you enter and conclude the search at the last sector number you enter. If you entered this mode by mistake, you can press **FCTN 4** to get back to the command mode.

When you have entered the sectors, you will be prompted for an ASCII or HEX string. Press "A" for an ASCII string and press "H" for a HEX string.

After you enter "A" or "H", you will be prompted for the actual string you wish to search for. If you have entered a string before, it will be displayed on the line. If you wish to clear the previous string press **FCTN 3**. You can still exit this mode by pressing **FCTN 4** and you will be returned to the command mode. You can also use the arrow keys to position the cursor where you want it.

Upon pressing enter the program will start the search. If the screen update mode is "ON", each sector will be displayed in ASCII or HEX depending on whether you are searching for an ASCII string or a HEX string. If you are searching for an ASCII string, each sector will be displayed in ASCII. If you are searching for a HEX string the sectors will be displayed in HEX.

EVALUATION COPY — SECTION OMITTED

During the search routine you have the option of stopping the search at any time. If you press **FCTN 4**, the search will be terminated and you will be returned to the command mode. If you press any other keys the program will pause and wait for you to press another key to continue or **FCTN 4** to terminate.

When the program finds the string you have asked for, the sector that it was found in will be displayed in ASCII or HEX depending on whether you were searching for an ASCII or HEX string. If you were searching for an ASCII string, the sector will be displayed in ASCII. If you where searching in HEX, the sector will be displayed in HEX. When the string is found, it will be displayed regardless of whether the screen update flag is ON or OFF.

If the program can not find the string a message will appear at the bottom of the screen indicating that the string was not found. When the search is concluded you will be returned to the command mode.

The VDP buffers are affected during this operation.

Disk 55. Contents of file MANUAL3

SECTION 23. T-TOGGLE BETWEEN ASCII AND HEX

Pressing the "T" will toggle the screen between ASCII and HEX. If a block of information is not being displayed nothing will happen. Toggling works on either memory or a sector of information.

SECTION 24. U-UPDATING SCREEN DISPLAY

EVALUATION COPY — SECTION OMITTED

SECTION 25. V-VIEW VDP MEMORY

EVALUATION COPY — SECTION OMITTED

SECTION 26. FCTN W (~)-WRITING A SECTOR TO A DISK

*** WARNING ***

IF YOU ARE WRITING A SECTOR TO A DISK BE VERY CAREFUL THAT YOU ARE WRITING WHAT YOU WANT WHERE YOU WANT IT. THIS PROGRAM HAS THE POTENTIAL OF DESTROYING DATA AND INFORMATION ON A DISK IF THE USER IS NOT FULLY AWARE OF WHAT THEY ARE DOING !!!!

When writing a sector out to a disk the program uses the SECTOR and DISK number being displayed on the status line. If you don't want to write to this sector or disk make sure you change the SECTOR or DISK number to the one that you want.

The data that is currently in the VDP RAM buffer is what is going to be written out to the disk so make sure it is the way you want it written out. Use the ALTER mode to change it to what you want.

To write a sector out simply press FCTN W and the operation will be completed.

The only effect to the VDP buffers is that the HEX buffer will be written to the disk. They can still be displayed as usual.

SECTION 27. FCTN C (`)-CURRENT CPU MEMORY ADDRESS

EVALUATION COPY — SECTION OMITTED

SECTION 28. FCTN G (})-CURRENT GROM MEMORY ADDRESS

EVALUATION COPY — SECTION OMITTED

SECTION 29. FCTN Z (\)-CURRENT VDP MEMORY ADDRESS

EVALUATION COPY — SECTION OMITTED

SECTION 30. FCTN T (])-TOGGLE STATUS LINE

Pressing **FCTN T** will allow you to toggle the SECTOR number and memory ADDRESS number in the center of the status line. If you wish to view the current memory address being pointed to in the memory area as indicated on the left side of the status line press FCTN T until you get the word ADDRESS in the middle and the address after it.

If you wish to view the current sector number press **FCTN T** until the word SECTOR is displayed on the status line and the sector number is after it.

The VDP buffers are not affected during this operation.

SECTION 31. FCTN I (?)-MAIN MENU

This command is your help command. If you need to refresh your memory as to the commands simply press **FCTN I(?)**.

The VDP buffers are not affected during this operation.

SECTION 32. FCTN P (")-Mapping Sectors

This command will enable you to take any directory sector or sector zero (0) and get the sector bytes translated into an English format instead of trying to figure it out yourself. The only sectors the program will not map is a sector that is initialized with >E5's.

To map sector zero simply put the sector counter on the status line on 0000. Set the DISK number to what you want and select read "R" to read the sector in. When the sector information is displayed on the screen and the cursor is flashing next to MODE press **FCTN P**. You will now get the translated version displayed on the screen. The information in this map breaks down as follows:

DISKNAME. Self explanatory

NUMBER OF SECTORS. This DECIMAL number is the number of sectors on disk.

SECTORS/TRACK. Number of sectors per track. Normally 09 for TI-DOS. DECIMAL number.

INITIALIZATION CODE. For TI-DOS and DISK MANAGER this is "DSK".

PROPRIETARY This tells you whether the disk is proprietary or not.

TRACKS/SIDE. For TI-DOS this number will normally be DECIMAL 40.

SIDES. Number of sides.

DENSITY. Tells you whether the disk is single or double density.

OF SECTORS USED. Self explanatory. DECIMAL number.

OF SECTORS FREE. Self explanatory. DECIMAL number.

You will now be prompted yes or no to view the BIT MAP in sector zero. If you select NO you will be returned to the command mode. If you wish to see it you will be asked if you want to list it to a output device. The program will select any valid device except cassette.

Next step is to read in sector "1" to see which sectors are being used as directory sectors. Refer to SECTION 34 for more information on SECTOR 1. After deciding which directory sector you wish to view, set up the SECTOR number and DISK number to read the sector in. When it is displayed and the cursor is flashing next to the word MODE press **FCTN P**. You will now get the translation of what all those bits and bytes mean. The information is described below:

FILENAME. Self explanatory

DIRECTORY SECTOR. This is the number read from the status line. After you have read in the sector don't change the status line sector number.

FILE TYPE. This part will tell you whether the file is DISPLAY/VARIABLE, DISPLAY/FIXED, INTERNAL/VARIABLE, INTERNAL/FIXED or PROGRAM.

PROTECTED. This will tell you whether the file is protected or not. Be advised this is not the same protection as Extended Basic's SAVE PROTECTED. This is the write protect DISK MANAGER puts in when requested.

RECORD LENGTH. DECIMAL number representing the maximum size limit of each record in FIXED and VARIABLE length files. i.e. DISPLAY/VARIABLE 80.

RECORDS/SECTOR. This DECIMAL number represents the number of FIXED records that will fit into one sector.

TOTAL RECORDS. This DECIMAL number represents the actual number of records that the disk controller writes in the file.

SECTOR OFFSET. This HEX number is the number of bytes the file goes into the last sector for that file.

NUMBER OF DATA SECTORS. This DECIMAL number is the actual number of sectors used to store the file.

SECTORS USED. This is what the sector fragment table will translate into for a file. This is a table of the actual sectors used to store the file.

Any time you map a sector you can also use the screen dump feature to make a permanent record.

If you try to map anything other than SECTOR 0 or a directory sector there is no telling what you can end up with for a read out.

Using this command will not affect the VDP buffers.

Disk 56. Memory Manipulator

Version:
Requires: XB, EA

Author: Don Thomson
Language: AL

Updated: 8/17/86

Program to let you explore all the memory in your TI and take apart what you find. Very user friendly. Very complete. Also developed by Thomson Software and generously made available to user groups.

```
dskdir. v2.0. 12-dec-96
Disk name               = JPHBCS1
Sectors total           = 360
Sectors used            = 214
Sectors available       = 144
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >003 MEMORY-1     52 DIS/VAR 80 Y >064 051
002 >004 MEMORY-2     92 DIS/VAR 80 Y >097 091
003 >005 MEMORY-HDR    2 DIS/VAR 80 Y >0f2 001
004 >006 MEMORY-O     68 DIS/FIX 80 Y >022 066 >0f3 001
```

Disk 56. Contents of file MEMORY-1

MEMORY MANIPULATOR FOR THE TI-99/4A COMPUTER

Version 1.0
EVALUATION COPY

Complete memory exploration program

For the TI-99/4A Home Computer

By

THOMSON SOFTWARE
1461 Beach Street
Muskegon, MI 49441-1099

This is an evaluation copy of MEMORY MANIPULATOR. In several places of this manual you will notice the following message:

EVALUATION COPY — SECTION OMITTED

I feel that there is enough of the manual included to allow you to make a good evaluation of MEMORY MANIPULATOR. There are many reasons why I did not include the entire manual on disk:

1. I would like to encourage more of our users to become distributors. I have priced MEMORY MANIPULATOR very fairly and I feel it is a great bargain. I hope you will support the lower priced software by readily paying. Other software houses will be forced to take notice.

2. I feel that the honesty of our "REGISTERED" users should be rewarded. They are entitled to extra value and support which should be rightfully denied to those who do not pay.

3. The complete documentation will not fit on the disk with the program.

I hope that you will not be closed minded about omissions, and hope to have you on my list of "REGISTERED" contributors soon.

Anyone may obtain an evaluation copy of MEMORY MANIPULATOR from a friend or club. After you have had the chance to use and evaluate the program in your own environment, you are trusted to either send in a contribution or to discontinue use of the program. In any case, you are encouraged to copy this program for evaluation by others.

TABLE OF CONTENTS

SECTION CONTENTS

Thank you for coming to THOMSON SOFTWARE for your utility software needs. I feel that you have made a good investment in a program that will be of value to you for a long time.

First thing that we are going to cover is how to load MEMORY MANIPULATOR in MINI-MEMORY and Editor/Assembler. The object code on the diskette that contains MEMORY MANIPULATOR is in COMPRESSED format. The object code can be loaded from any valid disk drive that is properly connected to the computer.

SECTION 1. Loading MEMORY MANIPULATOR with Mini-Memory

If you are running the program from Mini-Memory the first thing to do is reset the computer back to the master title screen with the Mini-Memory command module plugged in. Press any key to clear the title screen. Select option 3 which is to RE-INITIALIZE MEMORY. If you already have a program in memory that you want to keep, make sure you save it as instructed by the Mini-Memory manual. Press **PROC'D** to get the memory cleared so you can load MEMORY MANIPULATOR.

Now select option 1-LOAD AND RUN. The computer will ask you for the file name. At the file name prompt, type in DSK1.MEMORY-O. The program will boot in and start running automatically.

SECTION 3. Loading MEMORY MANIPULATOR with Editor/Assembler

Reset the computer to the master title screen with the Editor/Assembler command module plugged in. Press any key to clear the master title screen and then select ED/ASSEMBLER. Press option 3-LOAD AND RUN.

At the FILENAME prompt type in DSK1.MEMORY-O and press **ENTER**. The program will boot in and start running automatically.

SECTION 4. Error message decoding

EVALUATION COPY — SECTION OMITTED

SECTION 5. STARTING OUT

Welcome to MEMORY MANIPULATOR. Now that you have read through all of the preliminary stuff to get you acquainted we are going to tell you how to get full use from the program.

The first screen you will have is the credit/title screen. Press any key to begin. The next screen you will have will be the prompt for an output device name. I recommend that you enter a device name at this time. The reason is, if you come across a screen that you want to dump to printer and the device name is not typed in, you will be returned to this screen and the screen you wanted to dump will be lost.

The next screen is the MAIN MENU. Displayed on this screen are all of the commands available to you. To make the program more user friendly we have incorporated a single key stroke command entry feature. As soon as you press a key that represents a valid command you will be sent to that routine.

At the top of the screen is the status line. The status line displays the following items:

The left side will have the most recent output device name you have entered. The file name and any opening attributes however are not displayed.

The middle of the line will have the last memory area you accessed/displayed. This will be empty if you last selected the following command modes: D, H, R, S.

The right side of the line will have the command mode you currently have selected. The only time this will not change is when you select the T or M command.

Each time you select a command mode the status line will be updated to reflect your selection.

SECTION 6. SELECTING AN OUTPUT DEVICE

EVALUATION COPY — SECTION OMITTED

SECTION 7. CHANGING SCREEN/CHARACTER COLORS

EVALUATION COPY — SECTION OMITTED

SECTION 8. A-ALTERING MEMORY

Altering Memory mode will enable you to alter the contents of any RAM (CPU or VDP) in the computer.

Selecting this command mode can only be done after just completing the following command modes: C, G, F, V.

When you select the A command you will be prompted with the option to alter in the HEX field or the ASCII field. Press (H) for HEX and (A) for ASCII. After you select the field you will see a cursor in the top line of the field you selected. You may now position the cursor anywhere in that field you wish. To alter the memory in the ASCII field simply press any of the valid ASCII keys. You will see the ASCII field as well as the HEX field update at this time. If you are altering in CPU or VDP memory the new information will be instantly written to memory. GROM cannot be altered. To exit the alter routine simply press **ENTER**.

Altering the HEX field is a little more involved. First position the cursor where you want it with the arrow keys. When you alter a byte you must type in both the characters in the byte before it is written to memory or updated in the ASCII field. If you only change one nibble in the byte it will not be updated and changed in memory. When you are done altering memory simply press **ENTER**.

After you have exited the alter command mode you can reenter it by pressing the **A** key. You will again be prompted for the field you wish to alter. Reentering the alter routine with the current screen can only be done if you haven't selected another command mode.

SECTION 9. C-SCANNING CPU MEMORY

This command will enable you to scan any of the CPU memory in the computer. If you turn the peripheral ROM cards on you will be able to scan the ROM contents.

After you select this command you will be prompted for the start and stop addresses for the memory boundaries you wish to scan. The program is set up to require a minimum number of keystrokes from the operator. When you enter an address you do not have to enter all four digits. How ever many digits you enter, if the total is less then four the program will pad the address with zeros where needed.

After you have typed in the desired start address press **ENTER**. The cursor will now be in the stop address field. The same thing applies in this field as the start address. When you have typed in the desired stop address press enter and the desired memory will start scrolling up the screen.

If you wish to stop the scrolling but not exit the scan routine press any key, BUT THE "S" KEY, until it registers. To start the scrolling press any key, EXCEPT THE "S" KEY, until it registers. If you wish to exit the scanning routine press "S" until it registers. You will now be back in the command mode.

If you entered this routine by mistake simply enter >0000 for the start and stop address and you will be returned to the command level.

Disk 56. Contents of file MEMORY-2

SECTION 10. D-SCREEN DUMPS

EVALUATION COPY — SECTION OMITTED

SECTION 11. E-SETTING AND RESETTING CRU BITS

Selecting this command allows you to set or reset any CRU or combination of CRU bits that you wish.

The first thing you will be asked for is the CRU bit address you want. Enter any of the valid CRU addresses at this time. Displayed along with the address selection are some of the addresses for selecting the peripheral cards that plug into the expansion box.

After you have entered the address you want press **ENTER**. The next prompt will ask you if you want to S)ET OR R)ESET the bit. Press "S" to set the bit or "R" to reset the bit. After you have selected SET or RESET you will have the ENTER CHOICE prompt to enter another command option.

SECTION 12. F-FINDING A STRING IN MEMORY

EVALUATION COPY — SECTION OMITTED

SECTION 13. G-SCANNING GROM MEMORY

This command will enable you to scan any GROM memory in the computer whether it is in the console or in a command module.

Scanning GROM memory is identical to scanning CPU memory. Follow the steps outlined in SECTION 10 for complete details.

SECTION 14. H-PERFORMING HEXADECIMAL MATH

EVALUATION COPY — SECTION OMITTED

SECTION 15. K-COMPARING MEMORY

This section will allow you to compare any memory contents with any other memory contents within the computer system.

The first prompt is for you to select the two areas of memory you wish to compare. After your selection a non flashing cursor will be displayed in front of your choice.

As an example we are going to select option 4 which will compare CPU memory to VDP memory. The next prompt will be asking you for the starting addresses in CPU and VDP memory along with the number of BYTES you want compared. The first START address in our example will be for CPU. The second START address will be for VDP. Now enter the number of BYTES you want compared. Remember that all numbers you enter are in HEX.

When you press **ENTER** you will see two columns start scrolling up the screen. For our example the left column is the CPU memory contents that don't exactly match the VDP contents, which is the column on the right. If the contents of the corresponding addresses are the same the contents will not be displayed.

If you wish to stop the scrolling of memory press any key except the "S" key. To restart scrolling press any key except the "S" key. If you wish to exit the compare memory routine press the "S" key. You will now be at the command entry mode.

SECTION 16. M-MENU (HELP)

Pressing the "M" key will take you to the main menu.

SECTION 17. N-MOVING MEMORY

This routine will enable you to move memory from any memory location in the computer to any RAM in the computer.

For our example we are going to select option 6 to move memory from GROM the CPU RAM.

When you press the number 6 a non flashing cursor will appear in front of option 6 to remind you of your selection. Next you will be prompted for the two starting addresses and the number of bytes you wish to move. For our use we are going to move memory starting at GROM address >0000 to CPU RAM address >E000 and we want to move >0100 bytes.

With the cursor in the first address field just press enter. The program will automatically set the address to >0000. For the second address just press E and press enter. The rest of the address will be padded with zeros. In the third field press 0 and then 1. The rest of the field is padded with zeros. When you press enter the memory will be moved and you will be returned the command level. To double check that the memory was moved enter the CPU scan routine and check memory from >E000 to >E100. You will find the it will have the same contents as GROM memory from >0000 to >0100.

SECTION 18. P-OUTPUTTING MEMORY TO A VALID OUTPUT DEVICE

This routine will allow you to output to any valid device the HEX and ASCII equivalent of any memory area between the address you select.

The first thing this routine does is go to check to see if a valid device name is connected. If you have selected disk the file will be opened under the name you selected. Another nice feature about this routine is that it will automatically split your output file, to DISK ONLY, into 20K byte pieces. This will allow you to load the file into TI-Writer or Editor/Assembler editor for viewing. When the first file has 20K bytes in it the program will close down that file add 1 to the ASCII code of the first digit in your file name and open a new file under that name.

For example, if we have selected the filename DSK1.A-SPOOL, the filename would be incremented to DSK1.B-SPOOL. If you completely fill a disk the last file will be closed down. An error will occur because of MEDIA FULL. You will be taken to the ENTER OUTPUT DEVICE prompt to enter a new device name. You will then go to the MAIN MENU. If the end boundary you had chosen wasn't reached, enter the address that the last file finished at and your original ending address and start outputting to a new disk.

As the program is outputting to the device it is also scrolling the memory up on the screen. To stop the scrolling and the outputting to device press any key except "S". To restart scrolling and outputting to the device press any key except "S". If you wish to exit the routine press "S". With "S" you will simply exit the routine.

After you exit the routine you will be at the command entry level again.

SECTION 19. Q-QUITTING THE PROGRAM

To exit MEMORY MANIPULATOR press "Q". You will now be returned to the master TI screen.

SECTION 20. V-SCANNING VDP MEMORY

This command will enable you to scan all of VDP memory.

Scanning VDP memory is the same as scanning GROM and CPU memory. Follow the instructions in SECTIONS 10 for complete details.

SECTION 21. S-SETTING THE SCREEN SCROLL SPEED

EVALUATION COPY — SECTION OMITTED

SECTION 22. T-SELECTING AN OUTPUT DEVICE

This routine will allow you to select any device you want. It doesn't have to be valid. The validity is checked at each routine that uses an output device. Follow the procedures as spelled out in SECTION 6.

CPU MEMORY MAP
Editor/Assembler

EVALUATION COPY — SECTION OMITTED

VDP MEMORY MAP
Editor/Assembler

EVALUATION COPY — SECTION OMITTED

CPU MEMORY MAP
EXTENDED BASIC

EVALUATION COPY — SECTION OMITTED

VDP MEMORY MAP
EXTENDED BASIC

EVALUATION COPY — SECTION OMITTED

GROM MEMORY
GENERAL SYSTEM USE

EVALUATION COPY — SECTION OMITTED

```
MEMORY MANIPULATOR
START:A000 STOP:A1FF CPU

000=46 49 4E 44 20 28 48 29 45 58 20 20 28 41 29 53 FIND (H)EX (A)S
010=43 49 49 20 53 54 52 49 4E 47 45 4E 54 45 52 20 CII STRINGENTER
020=53 54 52 49 4E 47 20 28 33 32 20 43 48 41 52 20 STRING (32 CHAR
030=4D 41 58 29 46 49 4E 44 20 53 54 52 49 4E 47 20 MAX)FIND STRING
040=49 4E 3A 41 4E 53 57 45 52 53 20 41 52 45 20 41 IN:ANSWERS ARE A
050=53 20 46 4F 4C 4C 4F 57 53 3A 23 31 2B 23 32 3D S FOLLOWS:#1+#2=
060=23 31 2D 23 32 3D 23 31 2A 23 32 3D 23 31 2F 23 #1-#2=#1*#2=#1/#
070=32 3D 52 4D 4E 44 52 3D 45 4E 54 45 52 20 4E 55 2=RMNDR=ENTER NU
080=4D 42 45 52 53 3A 46 49 52 53 54 20 20 23 3A 30 MBERS:FIRST #:0
090=30 30 30 53 45 43 4F 4E 44 20 23 3A 30 30 30 30 000SECOND #:0000
0A0=41 4C 54 45 52 20 20 28 48 29 45 58 20 28 41 29 ALTER (H)EX (A)
0B0=53 43 49 49 20 20 46 49 45 4C 44 43 50 55 56 44 SCII FIELDCPUVD
0C0=50 47 52 4F 4D 4D 4F 56 45 20 4D 45 4D 4F 52 59 PGROMMOVE MEMORY
0D0=31 2E 20 20 43 50 55 2D 43 50 55 32 2E 20 20 43 1. CPU-CPU2. C
0E0=50 55 2D 56 44 50 33 2E 20 20 56 44 50 2D 56 44 PU-VDP3. VDP-VD
0F0=50 34 2E 20 20 56 44 50 2D 43 50 55 35 2E 20 47 P4. VDP-CPU5. G
100=52 4F 4D 2D 56 44 50 36 2E 20 47 52 4F 4D 2D 43 ROM-VDP6. GROM-C
110=50 55 53 29 45 54 20 42 49 54 2C 20 20 52 29 45 PUS)ET BIT, R)E
120=53 45 54 20 42 49 54 44 49 53 4B 20 43 4F 4E 54 SET BITDISK CONT
130=52 4F 4C 4C 45 52 3D 3E 31 31 30 30 52 53 32 33 ROLLER=>1100RS23
140=32 2C 20 50 4F 52 54 20 31 26 32 3D 3E 31 33 30 2, PORT 1&2=>130
150=30 52 53 32 33 32 2C 20 50 4F 52 54 20 33 26 34 0RS232, PORT 3&4
160=3D 3E 31 35 30 30 54 48 45 52 4D 41 4C 20 50 52 =>1500THERMAL PR
170=49 4E 54 45 52 3D 3E 31 38 30 30 50 2D 43 4F 44 INTER=>1800P-COD
180=45 20 43 41 52 44 20 20 20 20 3D 3E 31 46 30 30 E CARD =>1F00
190=43 52 55 20 41 44 44 52 45 53 53 3A 3E 30 30 30 CRU ADDRESS:>000
1A0=30 4D 45 4D 4F 52 59 20 20 4D 41 4E 49 50 55 4C 0MEMORY MANIPUL
1B0=41 54 4F 52 2A 2A 2A 2A 57 52 49 54 54 45 4E 20 ATOR****WRITTEN
1C0=20 42 59 2A 2A 2A 2A 44 4F 4E 41 4C 44 20 4D 2E BY****DONALD M.
1D0=20 54 48 4F 4D 53 4F 4E 4D 41 49 4E 20 49 4E 44 THOMSONMAIN IND
1E0=45 58 00 15 00 1A 00 1D 00 1C 00 0D 00 0B 00 0A EX**************
1F0=00 0D 00 06 00 0D 00 19 41 2D 41 4C 54 45 52 20 ********A-ALTER
```

THE USER SUPPORTED CONCEPT

User supported Software If you are using this software and find it to be of value to you, your contribution will be appreciated. ($10 is requested)

THOMSON SOFTWARE
1461 Beach Street
Muskegon, MI 49441-1099

Regardless of whether you make a contribution, you are encouraged to copy and share this program.

User supported software is an experiment in distributing computer programs, based on these beliefs:

1. That the value and utility of the software is best assessed by the user on his/her own system.

2. That the creation of home computer software can and should be supported by the computing community.

3. That copying of programs should be encouraged, rather than restricted.

Anyone may request a copy of a user supported program by sending $2.00 and a blank, formatted disk to the author of this program. An addressed, postage paid return mailer must accompany the disk. A copy of the program, with limited documentation on the disk, will be sent by return mail. The program will carry a notice suggesting a contribution to the program's author. The $2.00 will be applied to the purchase price if you elect to send the suggested donation. Making a contribution is completely voluntary on the part of the user.

Free distribution of software and voluntary payment for its use eliminates costs for advertising and copy protection schemes. Users obtain quality software at reduced cost. They can try it out before buying, and do so at their own pace and in the comfort of their own home or office. The best programs will survive, based purely on their quality and usefulness.

Please join the experiment.

If you believe in these ideals, your contribution is solicited to help make them work.

WHY SHOULD I PAY?

You are trusted to use this copy of MEMORY MANIPULATOR for evaluation purposes only, until the requested donation is made. There are some solid reasons for sending in your payment:

1. It is the right thing to do. You will feel a lot better about using MEMORY MANIPULATOR.

2. When we receive your donation we will send you the latest version. This is the version with the newest features. New features are being added to this program.

3. You will receive a complete, printed copy of the documentation.

4. I will add you to my list of supporters. Every time a new version comes out, I will automatically send you a note giving you the option of buying the new version for a nominal update charge.

5. I will give you "PRIORITY" treatment if you need to write with problems or questions.

AVAILABILITY OF MEMORY MANIPULATOR SOURCE CODE

Source code for MEMORY MANIPULATOR is available for $30.00. This fee covers the distributable source code for MEMORY MANIPULATOR.

SOURCE CODE AGREEMENT.

I understand that the source code provided by the company or derived through any disassembler, whether manual, automated, or computerized, is for my exclusive use and will under no circumstances be released or modified and reassembled for marketing to any other party, whether I have made SUBSTANTIAL MODIFICATIONS to the software or not. The sole interpretation of any of the provisions of this license rests with the company. I also understand that the source code for MEMORY MANIPULATOR is not distributed under the user supported concept and is not to be passed to any other person in any way. IT IS FOR MY SOLE USE ONLY!! I agree to these terms by signing my name and filling out the information below.

_____
Signature

_____
FIRST NAME LAST NAME PHONE NO.

_____
Address

_____
City St. Zip

Return this form with your check for $30.00 to:

THOMSON SOFTWARE
1461 BEACH STREET
MUSKEGON, MI 49441-1099

ORDER FORM

PURCHASED FROM: THOMSON SOFTWARE
1461 Beach Street
Muskegon, MI 49441-1099

DATE:	MY ORDER NO.	YOUR ORDER NO.	VERSION NO.
/ /	DA04510		

SOLD TO: _____

_____

_____

SALESMAN:	TERMS:	DATE SHIPPED	F.O.B.
DT400	NET 30	/ /	

QUANTITY	DESCRIPTION	UNIT PRICE	AMOUNT
	Circle items below you want to purchase		
1	MEMORY MANIPULATOR ver 1.0	$10.00	$10.00
1	Complete Printed Manual (free with registration)	NC	NC
1	Source Code	$30.00	$30.00
	Please make checks payable to DONALD M. THOMSON III	TOTAL	

Disk 56. Contents of file MEMORY-HDR

```
.PL 55
.IF DSK1.MEMORY-1
.IF DSK1.MEMORY-2
```

Disk 57. Fas-trans

Version:
Requires: XB
Author: Bill Harms
Language: XB
Updated: 08/17/86

A complete system for managing your checkbook or other financial records. Has loads of features, lots of documentation. Very complete and great for learning from since it's written in Extended BASIC (but it is not slow).

```
dskdir. v2.0. 12-dec-96
Disk name               = FASTRAN
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >014 *FA----RAW   12 PROGRAM      >015 011
002 >005 *FA--INPUT   61 INT/VAR254   >042 060
003 >00b *FA-1-YEAR   42 PROGRAM      >147 033 >00c 008
004 >00a *FA-1YRTXT   10 DIS/VAR 80   >13e 009
005 >007 *FA-ASSORT    9 DIS/FIX 80   >0a5 008
006 >004 *FA-CATEGY    6 INT/FIX 16   >03e 004 >020 001
007 >009 *FA-CHANGE   72 INT/VAR254   >0f7 071
008 >008 *FA-REPORT   75 INT/VAR254   >0ad 074
009 >006 *FA-SORTER   40 PROGRAM      >07e 039
010 >002 *FA-VARIAB    2 DIS/VAR 80   >022 001
011 >003 LOAD         28 PROGRAM      >023 027
```

Disk 57. Contents of file *FA-1YRTXT

First: load category labels. Enter Cat. Label File Name: DSK1.*FA-CATEGY

Press S if same as before
1 =Proceed. 2=Load diff. label
  FAST-Tran One Year
by mo. + on all categories

*** MAIN MENU ***

1-Load full YEAR file for
 display/print/change

2-Scan YEAR File by Category

3-Run REPORTS prog to
 BATCH UPDATE YEAR file

4-Create a new Spreadsheet

5-Return to LOAD for other
 programs

6-Exit to Basic

Created by Bill Harms

Insert disk with LOAD prog. Press Y if you're Sure!
Usually you would Batch Update a YEAR file once it's Created with a month of category totals, then use the Option 1 for a few chgs.
1=Proceed. 2=Menu

Insert FASTRAN disk with *FA-REPORTS prog. and press 1.
Enter file: DSKn.YRmoyryour to scan:
Enter >0< to Return to Menu
Enter -1 to Print Screen
 FAS-Tran Scan
Enter Category # to View
2.5 min. to load YEAR file. ENTER: MENU for menu -OR- ENTER: DSKn.YRmoyryour
Scnprt Goto Value Print MenuReset CM:
1=Save/menu. 2=NOSAVE/menu 3=Return to spreadsheet

SAVE FULL YEAR DATA TO FILE
You can replace existing file which puts new data into the current file or create a new file.
If a new file, allow 51 sec. of disk space.
Enter Disk Drive # to use: 2
Enter YEAR (last 2 digits)86
Enter 4(max)more of your choice
month per file name is :
Use it or use different one, if you have changed >ALL< CM-BUDGETS & CM-ACTUALS
1=Proceed. 2=ReEnter Filename
1 to change more. 2 for Menu
Enter number of category:
Cat.# to chg?
Mo. to chg?
Do>ALL<future mo. per CM?Y/N
Enter % Growth Rate: 0 %
Enter new value:
Do Totals Recalc Now? (Y/N)
TURN ON PRINTER AND PRESS**SPACE BAR** WHEN READY.
Skip cat. if Total=0 ?(Y/N)Y
Enter width of paper. 1 = 8 1/2". 2 = 11". 1
Type pitch: (11" Comp.=1pg) 2=Elite. 3=Compressed. 2
Enter Date/any message.
Press ANY KEY -to Pause
1=Continue 2=Menu
NO GO !!-PRESS 1 FOR MENU
couldn't access file. PRESS ANY key FOR MENU !!

Disk 57. Contents of file *FA-VARIAB

```
 16
 5
Y
PIO
```

Disk 58. Fas-trans

Version: *Author*:
Requires: *Language*: *Updated*: 08/17/86

See disk 57.

```
dskdir. v2.0. 12-dec-96
Disk name               = 11111
Sectors total           = 360
Sectors used            = 242
Sectors available       = 116
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 *FA-INSTER    7 PROGRAM      >022 006
002 >003 *FA-INSTR1   89 DIS/VAR 80   >028 088
003 >004 *FA-INSTR2   59 DIS/VAR 80   >080 058
004 >005 09SUMS/FYI   14 INT/FIX 29   >0ba 013
005 >006 09SYLK/FYI   15 INT/FIX128   >0c7 014
006 >007 PAY-4-BILL    2 DIS/VAR 80   >0d5 001
007 >008 PAY-4-MARY    2 DIS/VAR 80   >0d6 001
008 >00a SAMPLE/CKS    3 INT/FIX 23   >0d7 002
009 >00b YR0482/FYI   51 INT/FIX126   >0d9 050
```

Disk 58. Contents of file *FA-INSTR1

FAS-Tran version 2.5 by Bill Harms April 26, 1986

CHECKBOOK RECAPPER/ PLANNER
Requires Ext. Basic, a Disk Drive, and Memory Expansion. A printer is highly recommended. The following should be kept on the system disk esp. if you use a single sided drive:
*FA----RAW, *FA--INPUT,
*FA-ASSORT, *FA-CHANGE, *FA-REPORT, *FA-SORTER,
*FA-VARIAB, *FA-1-YEAR, *FA-CATEGY, *FA-1YRTXT, LOAD

For Speedy totals of Trans- actions/Checks by category. Reading: *FA-INSTR1,2

This program is simply to get totals of your checks and deposits by category. The totals are useful for your income taxes, and for budgeting.

It can be used each week or month or other. The total $ of all checks on groceries or car gas or income of one type versus another will be added. Generally one would build monthly transactions files to be used for planning. I did this program because I was sick of spending hours just recapping the checkbook to get the totals by Category. With this prog it takes me about 1 hour at the end of the month to enter the checkbook trans. and get all the reports.

For checks or deposits that represent more than 1 type or category, just enter both with a slightly diff. check/transaction number. i.e. a Visa card payment for computer items and meals out would be entered as two separate transactions.

First make a copy of the disk of programs, using the same name(FASTRAN). Do not write protect the disk, since you will be setting up USE defaults. Get out an initialized disk to be used for various files you'll create.

To use the program, LOAD it. Then run the Option to customize the existing categories. Get a printout, change them for your needs. 70 CATEGORIES ARE SET UP program is set for a max. of 99. — Or use a word processor such at TI-Writer to setup the Category Labels. Each one is a max. of 15 digits long. Then use the Prog. Option-Change/Copy/Convert/Split to Convert the file to the type normally used by the various programs (INTERNAL/FIXED 16). You can input trans. with no category labels set up and then add them later, but it's confusing.

Once past the 1/2 hour of pain to set up the category labels select the Option - INPUT transactions. OR just try ALL Reports Options. Try the SAMPLE/CKS trans. file to see the output reports and files. Try the Sorted Trans. Option for more.

You will be entering data with the following max.'s of digits for each.

```
TRANS#  DATE   CAT. $ AMOUNT
<-6->  <-6->   <2>  <--8--->

65     08      12   12345.78
2001.1 851131  12   -99999
JOB 14 NOV.85  01   -9999.12
```

AMOUNT has a limit of 6 digit whole #', incl. sign or 8 incl. a decimal. The TRANS# and DATE can be 1 digit. If you will want to print a sorted list of Transactions (with Subtotals by sort criteria) optionally, on DATE: use a convention such as YYMMDD or MMDD or DD.

Here is a sample of one of the reports.

```
MONTH JAN. 1985 - U. S. BONK
Mr. Hodge P. Podge
 CATEGORY            TOTAL
----------           -----
01 Income-Co.A..P. $-1000.00
02 Income-Co.B.me  $ -500.99
08 Groceries - - - $  666.00
38 Computer Stuff  $ 2006.00
                    --------
TOTAL of 94 checks=$ 1171.01
                    ========
```

The 4 output reports and the Category Labels list can be Printed to disk as well as the normally used Printer so you can have a DISPLAY VARIABLE 80 output file, perhaps to use for other purposes. Just enter DSK1.FILENAME instead of say PIO or RS232/2.BA=4800.PA=N in the USE defaults setup of the LOAD prog.

PRINTER CODES: The only special codes used in the program are set to run on the Gemini 10X. They are in prog. *FA-1-YEAR. line 4037 has CHR$(27);CHR$(66);(CHR$(TI) — where TI is either 2 for Elite or 3 for Compressed. Line 4040 reinitializes the printer with CHR$(27);CHR$(64). All other print routines should work on any printer.

The input data is ADDED to a disk file so you can add more tomorrow or whenever. Then you can run the option for a listing of transactions or the REPORTS prog to get totals by category as shown above.

Also you can run the Option to SORT the trans. by date, trans. #, or category and then get a transaction listing in sorted order with subtotals. The RAM sort is a very fast assembly routine from D. R. Romer and J. Clulow. You should send him $10 if you like it. I like this printout for a detail listing of all trans. within the category groups and a total for each. The disk sort is slow, but you won't need it unless there are more than 350 transactions/checks.

Use the Change Files Option to correct mistakes in the trans. file or to browse the file of trans. You can also use the Change Files Utilities option to Combine two transaction files. It can also be used to Split a Trans. file (save a part of a file to a new filename. This is creating a trans. file from part of an existing one.).

Just as with the Category Labels creation you can use another program's output such as TI-Writer or BASIC to create check files or "Import" them from a different software created file. You would create the D/V-80 file and then convert it to the type normally used by the programs with the Option to Copy/Combine/Convert files in the CHANGE program. Transaction Files have as the first record the count of transactions in the file. It is a left justified positive integer. The 2nd through whatever(count)of records in the file are laid out thusly:

```
     TRANS DATE  CAAMOUNT
   ------------------------
      124
ie.  123456DEC 0499-55
or   Y     5     041234.12
```

There are no spaces between the (Fields)! TRANS 6 digits, DATE 6 digits, CA(Category) 2 digits, AMOUNT 8 digits max. Spaces (Char 32) are used to fill out each field to it's full size except the AMOUNT.

The program has an option to EXPLODE the net paycheck of 2 persons. When entering checkbook deposits of a paycheck you can enter the net amount and let the program record the details such as your gross pay, inc. tax, ins. deduct.(10max) To use the option set up the categories as noted above including

Gross pay, and the various deductions. Next select 1 or 2 normal category codes which will be used to TRIGGER the EXPLOSION option. Then run the checks entry prog. Pick the option to use the pay explosion. Enter the 1 or 2 key category codes you selected. Then when entering checks, enter the TRIGGER(1/2). Now you can load the sample file of pay detail(explosion) I set up as filename PAY-4-BILL and proceed to edit it to your taste. It saves time if all or most of your pay and deductions are the same each pay period and you want to use the info. for monthly, YTD, etc. totals.

Be careful when you change category labels during the year. The dollars coded to say 08 Meals-Out, would stay on 08 even if you change the label of 08 to AUTO

Feel free to enter transactions that are not checkbook(checking account) activity. Such things as Savings Account interest that don't hit your checking account, but are needed for the Income Tax calcs. at year-end or for budgeting. The 99 categories are anything you want them to be. They could be asset/liability accounts so that trans. amount recorded are charges/credits to the balance sheet ie. a House Payment can be split between interest expense and loan principle reduction. You could even do double entry bookkeeping! Units can be entered ie. of goods sold by part no. The date field could be a description. You can embed a category code in the trans. #. See Sorter Option.

You can have different category label files to be used with unique transaction files. In a month there could be two different trans. files each with different category codes and labels. ie. one for one or two checking accounts and another for Savings Account transactions or super detail of Medical or Auto or gallons of gas/miles driven or Amount of units bought/sold of a product.

Category totals can be saved to disk along with the label & # in 3 ways via the REPORTS program, DISPLAY VARIABLE 80, INTERNAL FIXED 29(this one can be used by other options and includes labels so you won't need to load the labels file as well as the whole transactions file), and DISPLAY FIXED 128(which should be changed to INTERNAL via the RAW prog. This 3rd way is in Symbolic Link Format. All round the amounts off to whole dollars. The files include the filename and total of all categories, which would usually be one month's worth. The file in Symbolic Link Format is loadable into Multiplan so one can copy the figures into another spreadsheet of months of data for planning, etc! You would set MULTIPLAN to Transfer/Options/Symbolic, then Transfer/Load the file name you used when creating the SyLk format file. R.M. Mitchell, the publisher of *Super 99 Monthly* developed the techniques used in the file creation routines. `Great' The RAW prog. is by B. Traver, an excellent Ext. Basic Sector Editor. If you use/like it then send them some $$.

ONE YEAR PROCESSING BY MONTH AND CATEGORY TOTAL Here is a fast, slick spreadsheet like facility that displays a window of data and auto. recalcs totals when you change a `cell' (an amount in a month for a category - ie. Meals Out in Feb.). The printout is 4 pages(if you use all 99 categories) and don't have an 11 in. printer, but you can scan it in both the *FA-1-YEAR and the *FA-REPORT progs as well as do screen dumps.

The REPORT (*FA-REPORT) prog. has an option to put a month worth of totals into the file(spreadsheet) or just scan the YR file one category at-a-time. These are the cat. totals of a month of transactions by category - on all categories. The YEAR prog.(*FA-1-YEAR) allows scanning, updating and printing the year file of totals. It has YEAR totals of the months by category and has Year-To-Date totals of actuals and if you use the budget methods has Year-to-Date totals of the budgets as well as Current Month budgets. You can use it for planning and tax estimating and recapping of actuals for both.

First you can run the Option to load the Year scan/update prog. Try the Option to `Create a new Spreadsheet'. Note the totals change after each amount is entered. The screen has 3 categories on display with a row for their amounts by month along with totals. You can Goto, Printscreen and Change all non totals. Here are the words on the screen of the spreadsheet that indicate your
choices.

```
>Scnprt Goto Value Print Menu<
>Reset CM:11 Calc  Alter:AUTO<
```

You press the key of the first letter of each word. Scnprt=Screen print, Goto= Go to a category, Value=enter a new value in a cell, Print=Print the whole sheet, Menu=Menu/Save sheet to disk as Year file, Reset CM:11 is to Reset the Current Month which is now 11, Calc=Window Calculator, Alter:AUTO=Alter the calculation of totals which is now set at AUTO to be MANUAL.

The Window Calculator is like a 10 key that allows +,-,*,/,% calcs as on an adding machine. You'll be returned to the spreadsheet when done with it.

Budgeting can be done. Since it's done differently by different people there are a few different ways to do it in these programs. One way is to type in all the category totals of all the months (in the *FA-1-YEAR prog.) This can be speeded somewhat by the ability to use a % growth rate for ALL future months based on the amount in the current month. So if you entered Jan. amounts for all the categories you foresaw, you could then do Feb. thru. Dec. in one key press for each of the categories. The % growth rate can be 0, which yields the same $ in each future month. The % can be negative. Since it's a growth rate the amount in each future month will be based on the month before.

Another way to get the basic budget #'s into the file quickly is to create a transaction file for each of the 12 months. Each trans. file would have 99 max. numbers (1 for each category you need). Then just sequentially update the YR file with the 12 trans. files in the REPORT prog. option.

The actuals for each month will overlay the budget $ as you use the REPORT prog option again each month of the year to load in actuals from the transaction files of each month's check book entries. You could use the YR file update Option of the REPORT prog right after you have used the REPORTS prog option to printout the Category totals of a month of trans. Also it could be done later from the trans. files(or from the category SUMS files created optionally in the REPORT prog.) One would again do the YR file update from the REPORT prog in a sequential manner -01 Jan. thru. 12 Dec. in ascending order. The budget amounts of a month are moved into the Current Month bucket before they are overlaid by the actuals so you can see the Variance. Also the Year-to-date budget amounts are updated with the cur. mo. budget amounts each time the REPORT prog. YR file update option is used to update NEW Current Month amounts.

The printout of the YR file has 16 columns (one for each month and the totals) and up to 100 rows (one row for each category plus one for the total). You get the printout from the prog. *FA-1-YEAR. Screen dumps of one category only are avail. in the REPORTS prog. and of one or three cats. in this YEAR prog. If you have a printer that can use 11 inch width paper, and use the Compressed print option, the printout will neatly fit on 2 pages with all 16 columns across one page. If you use the Elite option or use 8 1/2 inch width paper the printout will print 9 or 12 columns and then continue later. You would just tape the pages for a neat 16 column sheet. The printout option gives a choice of not printing rows of categories that have 0 in the year total field and a 0 in the Year to date budget field. Don't use it, if there are offsetting amounts in various months.

The 1-YEAR prog. display of the matrix can only display 6 digit amounts. Since the amounts are rounded by the REPORT prog., the biggest # is 999999 or -99999.

The YR file name includes the 2 digit month code. When saving the YR file each time after its been changed or updated remember to use the month code of the new current month (01 thru. 12).

Since both the REPORT program and the 1-YEAR program change the grand totals in the YR file (seen on screen in the 1-YEAR prog.) the following is a detail description of the calculations. As you'll see the CM (Current Month) code is important.

BATCH UPDATE - done in the REPORT prog.-

Any of the 12 months plus the Current Month Budget can be updated. All categories are updated with the category totals derived from the transaction files.

Prior Month- Any month before the NEW Current Month. A prior month would be Jan.(except for the first one of the year with the file YR0086/BEG) thru the month coded as the Current Month on the file So if you already have a file updated/changed with September amounts of perhaps actuals and say Oct. thru. Dec. of budgets, then the NEW CM will be 10 Oct. ie. an infile: YR0386/JIM=CM 03 and a NEW CM of 04.

Here the amount in the Prior Month will be changed. First the Year-To-Date (YTD) Actual will be changed. The Total of all Categories for all 12 months will be changed. The Total Year amount will also be recalculated, due to the revised Prior Month input amount.

Current Month- This is the -NEW- CM or 1+the CM given in the input YR file to be updated. If the YR file says 04, the new CM is 05 May. The new file to be saved is thus updated to the Current Month (with May amounts). Here the amount in the new CM (i.e. Budget) is added to the YTD-Budget amount that already exists. The $$ being loaded from the trans. file totals are added to the YTD-Actual amount already there. The input amount replaces the existing amount ie. budget in the new CM bucket. This changes the Year total for that category. And the grand total of all Categories for all months is also changed.

Future Month- Any month beyond the NEW CM. If the YR file being updated has a month of 03 the next FUTURE MONTH is 05, since the NEW CM is 04. Here the Year total is changed to reflect a different amount in the month. The YTD Budget and YTD Actual are not affected. The total of all categories for all months is changed.

Current Month Budget- This is the 13th field. When you update CM Budget, the YTD Budget is changed to reflect a different amount in the CM Budget field. The CM Budget is compared the to CM Actual to display/print a CM Variance in the 1-YEAR prog. as is the YTD-VARiance to Budget. If you were loading YR file on say Feb. it's file name shall be YR02------. To put in #'s for March, you would name the new file YR03---

CHANGES/RECALCS -done in the prog. *FA-1-YEAR

Here the CM (Current Month) has a default setting of 1+the CM as in the file name of the YR file loaded. ie YR0686/BIL would preset a CM of 07 July. You can reset the CM, but it affects the calculation of totals, etc.

The Variance of Actual to Budget is Shown for CM and YTD.

When you change a prior month ie. 01 -> 05, the YTD Actuals are changed. Same if you change the current month 06. The YEAR total of the category and total of all cat.s is changed.

When you change a future month the YTD's are not affected. This option updates the YEAR total of the category and of the total of all cats. As an option on future amts entry you can use the % growth method to base future month amounts on the C/urrent M/onth Amount.

A change of the cur. mo. $$$$ changes the year total of the category selected as well as the total of all year total of all categories.

You can change the CM Budget also, which will change the YTD-Budget only. This is the only way to change the YTD Budget am't in this prog.

CAUTION: If you reset the CM code and then began changing amounts in a month, you will need to change all the category amounts in the month selected because the formulas work off the CM code!!!!

As a part of the Change option in this prog. you can Switch the Recalcs and redisplay of amounts to be done only after you've entered up to 12 different amounts. This speeds data entry. It's called the MANUAL recalc method. The default method is called AUTO which recalc's all totals after each new amount is entered.

I've used ON BREAK NEXT at the start of each prog. so you don't need to worry about pressing **FCTN CLEAR** at a critical moment. The LOAD prog. disables the **FCTN QUIT** key. All programs and files are unprotected so you can quickly change them to suit your needs or desires — and perhaps, fix a bug or two.

BEWARE !! Programs have limited error checking for disk/file access. Make sure you have correct disk in correct drive and — with enough space for the file you wish to create.

The program options have lots of descriptions and on screen instructions. Still, if a bit confused, try them. There is usually a note about critical operations. You'll rarely need to refer to this background.

Please copy this program for your friends. Hopefully, there are no BUGS in it. It is Public Domain & it's illegal to sell it! However, if you like it >> please send me $5 &/or your comment at 6527 Hayes Ct. Chino, Calif. 91710. I'll send you any Bug Fixes and Enhanced Versions. HAVE FUN — EXPLORE in Harms' way.

Disk 58. Contents of file *FA-INSTR1

Here are the required programs and files that should be on your disk(s) copy.

LOAD	Starts it All. Updates file *FA-VARIAB with session variables i.e. printer device name. All programs can really be run separately.
*FA-INSTER	This prog prints or displays the instructions: *FA-INSTR1 and 2
*FA-INSTR1	D/V80. The instructions/background. read by prog *FA-INSTER
*FA-INSTR2	D/V80. The inst/ground cont. a detail desc of each program
*FA-VARIAB	D/V80. The defaults file created by LOAD prog. and used by others
*FA--INPUT	Enter transactions here and create files of them ie. SAMPLE/CKS I/F23. Also can use a D/V80 explosion file ie. PAY-4-BILL Get transaction listings to screen or printer
SAMPLE/CKS	I/F23 A sample file of transactions created by *FA--INPUT. It's used by *FA-REPORT and *FA-SORTER
PAY-4-BILL	D/V80. A start-up file for you to change and use in *FA--INPUT for the net pay explosion trigger!
PAY-4-MARY	D/V80. The second net pay file for a second trigger!
*FA-CHANGE	To customize up to 99 Category Labels To correct transaction file errors To copy, combine, or convert transaction files To convert category files. To get a disk listing
*FA-CATEGY	I/F-16 The set-up file of 70 out of 99 max. category labels
*FA-SORTER	RAM sort or disk sort transactions files and get sorted printouts with or without subtotals by one of 3 sort criteria chosen along with the grand total
*FA-ASSORT	D/F80. Assembly lang. prog for RAM Sort
*FA-REPORT	Get reports of category totals and save totals to disk as D/V80, SYLK, I/F29 or Update a month of amounts (all categories + total) in the YR file to use in the prog *FA-1-YEAR. Can get a disk listing. Samples of files created are 09SUMS/FYI and 09SYLK/FYI
*FA----RAW	Sector editor to go from Display to Internal Type. Used by the *FA-REPORT prog on the Option to create SYLK files. ie.09SYLK/FYI If it locks-up CPU just try again. It's touchy!

09SUMS/FYI	I/F-29. A Sample file of the 99 category #'s, labels, and amounts
09SYLK/FYI	I/F-128. A Sample file of the 99 category #'s, labels, and amounts
*FA-1-YEAR	Create YR------ files, has Scan and Update Options for 1 year by mo by each of the 99 cats + totals. Get big printout of full matrix incl a little Window Calculator needs file *FA-1YRTXT which has text for screens
*FA-1YRTXT	D/V80. File of the screen lines of text used by the prog *FA-1-YEAR — done to save prog space in TI RAM. It works for me.
YR0482/FYI	I/F126. 51-sector file of 100 records(example. not required) Created by the *FA-1-YEAR prog. Used by it and *FA-REPORT prog

PROGRAM OUTLINE for the interested users.

LOAD

100 - 230	Set-up, prescan, call loads for various
240	Load in speech, printer, colors variables from file *FA-VARIAB
250 - 430	User Colors change routine
440 - 570	Install true lowercase
580 - 630	User Speech change routine
650 - 710	User Printer change routine
720 - 790	Save above choice variables to file *FA-VARIAB
800 -1130	Menu for User program option & load that program
1140 -1290	Instructions load, Print/Display from file: *FA-INSTRU
1300	Error trap

*FA-INSTER

100 - 340	Reads file *FA-INSTRU and prints it or displays it runs the LOAD prog when done. This prog is to accommodate 1 disk drive users

*FA-INPUT

100 - 190	Set-up, prescan, load variables (color, speech, printer)
200 - 350	Menu for user choices of running other program options (and loading that program) or ENTERING CHECK DATA
360 - 460	User select max.# categories acceptable and User select of use of NET PAY EXPLOSION technique and User entry of Current Checkbook Balance and enter Trans. file to be Appended
480 - 550	Zero input arrays, Display Transactions/Checks entry screen
580 - 820	User input of up to 10 trans./checks on one screen
830 - 920	User Option to REDO of above checks data
930 - 960	User Option to get printout of Checks entry screen of data
1010 -1060	User option to Save Data(enter filename), or return to menu

1100 -1190	Fill out check #, date, cat. code to fixed length
1210 -1370	Combine all (check #, date, cat. code, amount into 1 record and SAVE
1380 -1590	User option to count checks in a file, and check for Explode triggers and go to Explode routine if any Triggers
1600 -1650	error traps
1670 -end	NET PAY EXPLOSION routines
1670 -1870	Load cat. #, cat. label, $ amount from user requested file
890 -2050	Display data of all 10 categories and other info.
2060 -2130	User select option to Accept or Escape or Redo any of data
2140 -2230	Accept routine — save $amount, category label code of above 10 plus the originally entered check # and date to the file of user choice
2260 -2620	Redo routine to load array with user input of new Cat.#,label, $ amount. Incl. save of that data to user selected Display/Variable 80 pay file and the save of data to the user selected checks file as above
2630 -end	Return to normal checks entry screen for perhaps more user entry of checks/transactions
3100 -3680	Subprog to get listing of transactions(print or screen)
*FA-REPORT	
110 - 210	Setup, prescan, load variables (speech, colors, printer)
220 - 280	On user requ. display totals of checks File that was read
290 - 410	Display main menu 1
420 - 520	Display last menu 3 if user option directs him there
530 - 640	Display mid. menu 2 if user option directs him there
650 - 870	Read file of checks to get totals of checks and $ amount by category, and total # checks, and Dollars first zeroing the 3 arrays
890 - 970	User select type of cat. totals file save
990 -1200	Save category totals to disk I/F-29
1210 -1320	Load cat labels for cat. totals disk saves
1330 -1590	On user request display or print totals by category
1850 -1940	Confirm user request of prog. end and error trap
2050 -2270	Call keys, error traps and user option running a diff. program
2280 -2540	On user request read and display or print a disk directory/list
2550 -2680	Call keys for pause control and error traps
2690 -2764	On user request to print an image of screen to the printer
2768 -2828	Routines to scan or update the YRfile with 100 records with a file of trans. or cat. sums loaded into RAM. (screen dump optn)
2837 -2920	Start of SYLK file creator
2930 -3000	Load into arrays cat. labels & amts per cat. sums I/F-29 file
3030 -3560	Finalize print of I/F128 SYLK format file for Multiplan

*FA-CHANGE

110 - 170	Setup, prescan, load vari. (speech, printer, color) from *FA-VARIAB
180 - 370	Display MAIN MENU and get user choice, and on user request run other programs, or end program
380 - 970	SUBPROGRAM TO CHANGE, ADD, DELETE CATEGORY LABELS
380 - 520	Setup, prescan, load file of cat. labels per *FA-CATEGY into array and display MENU for change, add, delete, end and get user choice
530 - 590	On user request ADD via Insert method new categories
600 - 640	On user request CHANGE categories
650 - 720	On user request DELETE categories
730 - 800	On user request SAVE categories to disk file *FA-CATEGY
810 - 960	On user request PRINT/DISPLAY categories now in RAM user can temp. reset printer device i.e. to disk
961 - 970	On user request END categories routine with note to save first
980 -1430	SUBPROGRAM TO CHANGE CHECKS IN A DISK FILE
980 -1010	Setup, prescan
1011 -1035	Scan file via REC for display of trans. information
1040 -1130	Clean out array of checks data and load file of checks that user requests
1140 -1150	Get user choice of changing check data or return to MAIN MENU
1160 -	Start of changes module
1160 -1200	User enter highest # cat. code to be used
1205 -1230	User entry of check #, date, cat. #, $amount
1240 -1280	Scan array for user requested check
1290 -1300	Display data of requested check
1310 -1320	User entry of changes
1330 -1350	User choice of doing another or not
1360 -1430	Save new checks info to user requested file name
1502 -2150	SUBPROGRAM TO DISPLAY/PRINT A DISK CATALOG
1502 -1860	Setup, prescan
1870 -1930	Get user choice of disk drive and either printer or display output
1940 -2150	On user option get and either display or print a disk listing
3000 -3710	SUBPROGRAM TO COMBINE OR COPY CHECK FILES
3000 -3102	setup, prescan
3130 -3250	Get user choices of input file 1, input file 2 (optional) and output file names
3260 -3710	Input data from first file and print to output file, then input (on user request) 2nd input file and add to output file display counts of checks processed

*FA-SORTER

100 - 160	Prescan, load user defaults
170 - 190	Main menu for ram or disk sort or copy a transaction file
200 - 240	Load category labels if user picks category sort
250 - 280	Main menu continued
290 - 340	On user option Copy a file
350 - 460	On user optn for Ram sort - load the trans. file
470 - 600	Menu - Sort by: Category, Date, Trans.# or goto Prior menu, Do other FAS-Tran jobs, Exit to Basic, Get reports from files already sorted by this prog.
610 - 670	On user option - run the LOAD prog.
680 - 760	On user option Ram sort data
770 - 820	User choice: Printouts/Disk Save, direct to Disksave or Menu
830 - 850	Printout transactions. get choice of listing with subtotals by sort criteria and total or just a listing
852 - 859	User set of print device temp. ie to disk
860 -1160	Printout continued
1170 -1230	Disk save of sorted trans
1240 -1310	Error traps
1320 -1350	Printout routine continued (print subtotals)
1360 -1470	Disk sort of disk file
1480 -1550	Error traps

*FA----RAW

100 - 130	Setup
140 - 190	User choice to proceed or return to Reports prog
100 - 220	User enter disk drive number
230 - 370	Call routines to Read and Write to appropriate disk sectors of filename. Write is to change file to INTERNAL
unseen	Barry Traver's RAW program image file that's loaded into low when prog is loaded to Read a sector and Write to that sector in byte 12 (File Descriptor Record)

*FA-1-YEAR

100 - 180	Setup incl. DIM array 16,100 for whole YR file
200 - 220	Data stmts for scan option
240 - 320	Load variables per *FA-VARIAB
340 - 560	Load category labels
580 - 900	Menu for scan, load file for review, update, run load prog or run *FA-REPORT prog or create a YR file or enter #'s into a blank sheet (to be saved as a YR file.
940 -1040	Per user goto routine to display neat screen and enter #'s into blank sheet
1100 -1120	To zero the whole 100 by 16 element array
1400 -1635	Scan a YR file and get screen dumps. This option only inputs and displays one category (record) at a time
1700 -1880	Load YR file into array
2000 -2440	Display 3 categories on fancy screen with menu options: Screen goto, change, print whole matrix(file), reset cur mo alter calcs
2450 -2451	Reset cur mo per user option
2452 -2453	Per user option alter calcs (auto/manual)
2480 -2520	Per user menu return, but choice of Save data or not first
2540 -2865	Per user save array to YR file
2920 -2980	Goto option routine
3020 -3440	Change routines — Can do % growth rate of a Cur Mo amt — Can do up to 12 changes before redisplay (manual recalc) for speed
4000 -6020	Print big array to paper. Has options for Elite/Compressed and for 8 1/2" or 11" width paper and skip of categories with 0 $
20090-20910	subprog to use a calculator in a window

Change the programs to suit your needs or use modified parts of them for a variety of related needs (or what is more likely — fix my bugs

Disk 59. J.P. Graphics

Version: *Author*: J.P. Morin (?)
Requires: EA, XB *Language*: Forth *Updated*: 08/17/86

A complete bitmap graphics system written in TI-Forth. Includes sound capabilities and an implementation of Logo graphics. One of the most powerful graphics programs available. Comes with extensive documentation and a fantastic demo.

```
dskdir. v2.0. 12-dec-96
Disk name               = JPGRAPHICS
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 FORTHSAVE    39 PROGRAM    Y >022 038
002 >004 JPGRAPHICS  310 DIS/FIX128 Y >050 280 >005 029
003 >003 LOAD          9 PROGRAM    Y >048 008
```

Disk 60. J.P. Graphics

Version: *Author*:
Requires: *Language*: *Updated*: 08/17/86

See disk 59.

```
dskdir. v2.0. 12-dec-96
Disk name               = JPGRAPHICB
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 JPDOCS       93 DIS/VAR 80 Y >022 092
002 >021 JPSOURCE    265 DIS/FIX128 Y >07e 234 >003 030
```

Disk 60. Contents of file JPDOCS

JPDRAWING by Jean-Pierre Morin (Version 3.0)

INTRODUCTION

JPDRAWING program is designed to draw color graphics in the TI-FORTH environment without having prior knowledge of TI-FORTH on the part of the user. Therefore, it is accessible and easy to assimilate. A few minutes per command should amply suffice to become familiar with the single-key and LOGO commands.

JPDRAWING requires the following minimal TI-99/4A system configuration:

- TI-99/4A console with monitor or TV set (preferably color)
- One disk drive storage system
- 32K expansion memory system
- Editor/Assembler module

Optionally, a printer will provide black and white screen dumps in two different formats.

FEATURES

- Full bit map or split screen (SPLIT2) screen formats
- Line, circle, rectangle, arc and many more drawing commands
- 7 different paint brushes
- Full drawing LOGO facilities
- Paint command
- Screen save and load to/from disk
- 2 print screen commands
- Some forth utilities like SAVE, ST-STTS, ASC
- 2 easy ways of adding new drawing commands
- Big letter typing
- Command saving and re-execution. (User-defined commands.)

LOADING THE PROGRAM

Using the Editor/Assembler's Option 3, type "DSK1.FORTH". The program will load automatically and execute a small demo upon your request. When the demo is through, you are free to take a crack at it! This program capabilities are best learned through experimentation; don't be afraid to try your fancy. Naturally, you will have put a write-protect tab on your program diskette first!

SCREEN DESCRIPTION

The screen consists of a column of 6 two-digit numbers at the top right corner; these 6 numbers are referred to as P1 to P6 (Parameter 1 to parameter 6). Immediately below them is the 15-color tablet and at the bottom right corner is the angle value used by LOGO commands (0=right, 90=down, 180=left, 270=up). A sprite (the cursor) defined with one of eight different shapes is used to show the type of paint brush in effect. This sprite is called the cursor in the following text. You can use the joystick #1 or arrow keys to move the cursor around. A second sprite, called the beginning sprite, can be anchored anywhere on the screen to draw lines, circles and others between it and the cursor.

FULL SCREEN BIT-MAP DRAWING COMMANDS

For each of the commands below (words in parentheses), a single key is assigned (far left) which takes effect immediately upon being pressed. The words themselves are used in COMMAND LISTS or in DESSIN2 mode to achieve the same results.

0 (KEY0 or PU): No trailing line appears on the screen when the cursor is moved.

1 (KEY1 or PD): The cursor is assigned a 1-pixel paint brush; when moved, a thin line is trailing.

2 (KEY2): The cursor is assigned an 8-pixel horizontal paint brush; when moved, a "ribbon" effect can be achieved.

3 (KEY3): The cursor is assigned a solid square of 3 by 3 pixels; when moved, it leaves a thick line behind.

4 (KEY4): The cursor is assigned a 3-pixel paint brush with this shape "/".

5 (KEY5): The cursor is assigned a 3-pixel paint brush with this shape "\".

6 (KEY6): The cursor is assigned an 8-pixel vertical paint brush.

7 (KEY7): The cursor is assigned a 3-pixel vertical paint brush with two pixels in between.

= (+BCKGRND): Advances the color tablet pointer to the next color in effect. See also the G command.

b (>BEG): Anchors the beginning sprite to be used by the a, c, d, g, l, P, R, or s commands.

l (S-LINE): Draws a smart line between the beginning sprite and the cursor position (See L (MLINE) command). The l command is slower than the L command, but works better in multicolor drawings. (See F (FST-MD) and S (SLW-MD) commands).

c (CIRCLE): Draws a circle between the beginning sprite and the cursor.

s (SQUARE): Draws a rectangle between the beginning sprite and the cursor.

h (HOME): Places the cursor in the middle of the screen.

/ (NXT-NMBR): Advances the parameter pointer and sets it in input mode (See i (INPT-NMBR) command).

!, @, #, $, % and ~ (SHIFT 1, 2, 3, 4, 5 and 6): Places the parameter pointer at P1 to P6 respectively, and sets it in input mode (See i (INPT-NMBR) command).

i (INPT-NMBR): Allows the input of a value beside the parameter pointer (P1 to P6). Theses parameter numbers are used as input by many functions (FXX), mainly LOGO functions. A key other than 0 to 9 will terminate the input; use space bar to exit and activate the use of the joystick.

d (ARC): Draws a right arc between the beginning sprite and the cursor:
P1 is the radius of the circle
P2 is the number of degrees of curvature of the arc.
(EX.: 6 will give you a 60 degrees arc, 18 will give you a 180 degrees arc (half a circle)).

e (EXCT): Executes a COMMAND LIST at block P1 line P2. This COMMAND LIST can be entered via the editor or automatically while you draw, after having pressed the O command, and terminated with the E command. See the COMMAND LIST FORMAT section.

g (ARG): Draws a left arc. (See d (ARD) command).

n (ST-ANGLE): Allows input of the drawing angle. A key other than 0 to 9 will terminate the input.

o (ST-CLR): Sets old color. The selected color will appear just below the white color. Used by CONTROL C and P commands.

p (PNT): Paint command. First set color using = command, put cursor inside what you want to paint and press p.

r (READ): Puts the current cursor column position in pixels into P1 and P2; if column=123, P1=1 and P2=23, and row position in pixels into P3 and P4.

q (ADJST): Adjusts cursor at the beginning of an 8-pixel structure, to fix more than 2 adjacent colors.

t (CLR-TBL): Restores the color tablet to its original setting. Very useful after a CONTROL C or P command.

a (ATTCH): Attaches the beginning and the cursor sprites together. Use the same command to de-attach.

w (WRITE): Puts cursor at position defined by P1 and P2 for column and P3 and P4 for row. If P3=01, P4=12 then row=112. Opposite of the r command.

x, y, and z: Saves the cursor position into internal variables (3 available). Commands X, Y and Z put the cursor back to these positions. Useful to "reference" a cursor location on the screen.

A (APPND-SV): Similar to O command, saves commands in COMMAND LIST at the end of the existing COMMAND LIST.

B (BCKGRND): Changes color of the top and bottom of the screen. First select the color using = command.

C (CHGN-SPRT-CLR): Changes sprites color (Sprites are the beginning sprite, the cursor, the parameter values, pointer arrows, and the angle). First select color with = command.

D (DBC-ON-OFF): Toggles debouncing ON and OFF. With debouncing ON, a key pressed down will not automatically repeat, allowing greater precision drawings.

E (END-SV): Closes (ends saving) the previously opened COMMAND LIST. See O and A commands.

F (FST-MD): Restores fast drawing mode for LOGO commands.

G (GT-CLR): Sets the color tablet pointer to P1.

H (F04): Draws a house of height P1 pixels.

L (MLINE): Draws a line between the beginning sprite and the cursor (faster than l, but not as smart).

O (OPN-SV): Initiates the save mode for a COMMAND LIST. "EX" is automatically written at block P1, line P2. Each and every command following the O command will be saved until an E command. The saved commands can be executed with the e command.

P (PRTL-MD-CLR): Modifies color on a portion of a screen delimited by the beginning and cursor sprites. First select color to be changed with the o command, then select the new color with = command (See CONTROL C command).

R (RPLCT): Copies part of the screen delimited by the beginning sprite and the cursor to column (P1,P2) and row (P3,P4). The r command can be used to setup P1, P2, P3 and P4.

S (SLW-MD): Selects slow drawing mode for LOGO commands.

T: Selects typing mode. All capital letters will be drawn with the specified angle. Use] to exit typing mode.

> Similar to T command. Advances an extra P1 pixels and rotates (P2, P3) between each letter. If P2=03 and P3=30, the rotation angle is 330 degrees to the right (equivalent to 30 to the left).

X, Y, and Z: Restores the cursor position. See x, y and z commands.

"(" and ")": All commands between (and) are repeated P5 times. P5 must be setup first. This command only works in save mode (command O or A) or when executing a COMMAND LIST (e command).

FUNCTION CLEAR (CONTROL B): Exit. Use DESSIN or JS in DESSIN2 mode to go back in drawing mode. Use MON to exit to the master console title screen.

CONTROL C (MD-CLR): Modifies color. First select the color you want to change using the = and o commands, then select the new color and press CONTROL C. To change a portion of the screen use the P command.

CONTROL E (CLS): Clears the screen.

CONTROL L (LD-SCRN): Loads screen P1, where P1 is an arbitrary screen number between 01 and 07. Be sure to have the appropriate diskette before executing this command. The screen must have been previously saved with the CONTROL S command.

CONTROL S (SV-SCRN): Saves screen P1 where P1 must be an arbitrary screen number between 01 and 07. Be sure to have the appropriate diskette before executing this command.

CONTROL T (CLT): Clears the first 4 lines in DESSIN2 mode.

CONTROL P (PRNT-SCRN): Prints screen in condensed format with a 90-degree rotation to the right. (. command is nicer but slower). Press Q for a while to cancel printing.

~ (PRNT-SCRN2): Prints screen. Press Q for a while to cancel printing.

FUNCTION BEGIN or CONTROL N (EXCT-FXX): Executes function XX (See FXX list below). You can create your own FXX functions; this command will execute them. You must first specify the function number XX in P6.

CONTROL R (RPT-FXX): Executes FXX P5 times rotating 360/P5 degrees each time.

FUNCTION REDO or CONTROL R (PTR-FXX): Similar to the CONTROL R command, but P4 pixels from the center for each rotation.

CONTROL W (RPTWR-FXX): Executes function FXX P4 times, advancing P5 pixels each time.

_: Rotates left P4 degrees.

?: Rotates right P4 degrees.

&: Saves angle value in an internal variable.

": Restores angle value from the internal variable.

{: Advances P3 pixels in the direction of angle.

}: Advances P3 pixels in the opposite direction from angle.

FXX FUNCTIONS

F01 Draws a star. P1=Hexagon side size in pixels.

F02 Advances P3 pixels, rotates right 360 * P1/P2 P2 times.

F03 Advances 0, 2, ..., 198 pixels rotating (P1 * 100 + P2) each time 100 times.

F04 Draws a house. P1=Height of the house in pixels.

F05 Draws a polygon (PG).
P1=Side length in pixels
P2=Number of sides

F06 Draws P1 polygons with a rotation of 360/P1 (RPG)
P1=Number of rotations
P2=Side length in pixels
P3=Number of sides

F07 Draws P1 polygons with a rotation of 360/P1 P4 pixels away from the cursor.
P1=Number of rotations
P2=Side length in pixels
P3=Number of sides
P4=Radius of rotation in pixels

F08 Draws a flower.
P1=Radius length in pixels
P2=Number of degrees for arc (60 is a good choice)
P3=Number of petals

F12 Similar to F08, puts a circle in the middle.

F09 Draws and rotates until you type "/".
P1=Side length in pixels
P2 and P3=Degrees for rotation. (EX.: 144 -> P2=1, P3=44).

F10 Repeats 720 times [advance P1, turn right P2, increment P2 by P3].

F11 Draws a tree.
P1=Longest branch in pixels
P2=Smallest branch in pixels
P3=Number of divisions for each branch
P4=Number of degrees between two branches.

F13 Similar to F11, using random numbers.

LOGO COMMANDS AND DESSIN2 MODE COMMANDS

You can create your own drawing functions using LOGO type commands. The best way to learn these commands is to go in DESSIN2 mode and try them. You can also use the full screen commands by using their names in full.

xx FD: Advances the cursor xx pixels in the direction of angle.

xx BK: Advances the cursor xx pixels in the opposite direction from angle.

xx RT: Right turn of xx degrees.

xx LT: Left turn of xx degrees.

ANGLE@: Returns the angle value.

xx ANGLE!: Sets the angle value.

PD: Puts cursor in drawing mode.

PU: Puts cursor in non-drawing mode.

XX [...]: Repeats instructions between [] XX times. Can only be used in definition mode.

xx COLPOS !: Sets the column position to xx.

xx ROWPOS !: Sets the row position to xx.

xx COLPOS+!: Adds xx to COLPOS.

xx ROWPOS+!: Adds xx to ROWPOS.

xx GET: Puts on the stack the first xx parameters.

xx TAB-NMBR @: Puts on the stack Pxx+1 (For internal offset).

JS or JOYSTICK: Goes in joystick drawing mode.

ASC x PKEY: Will execute the x command, where x is any one of single-key commands.

EXAMPLE

You want to create a rectangle function where P1=Height and P2=Length (both in pixels). Exit drawing mode and type:

: RECTANGLE PD 2 GET 2DUP 4 [FD 90 LT] ;

You can assign RECTANGLE to the R key so that every time the R key is pressed RECTANGLE is executed, using the ASC and DFN commands, as follows:

ASC R DFN RECTANGLE

Next type DESSIN, setup P1 and P2, press R and a rectangle will be drawn. Try changing the angle value with the n command. Press R again. You got it! You can also assign a function number to the RECTANGLE function as follows:

: F33 RECTANGLE ; (33 is an example, suit yourself!)

To execute, setup P1 and P2, set P6 to 33 (or what have you) and press FUNCTION BEGIN. You can also use FUNCTION REDO, CONTROL R or CONTROL W to draw multiple rectangles.

COMMAND LIST FORMAT

Using the O command, you can save all the commands up to the E command (END-SV). You can re-execute this COMMAND LIST with the e command. In both cases, you must first setup the TI-FORTH block number in P1 (any of the 90 blocks (0 to 89) available on a blank initialized diskette), and the line number within that block where your function starts in P2. You can use the editor to create or modify a COMMAND LIST stored on a block, or input a block in "real-time", as you draw, with the O command. The first 2 characters of a COMMAND LIST must be "EX", otherwise the e command will not execute it. The last character must be ";".

A few changes were made to the TI-FORTH editor:

- Control characters are displayed in reverse video.
- You can insert a control character by typing CONTROL I followed by the control character.
- CONTROL INSERT and CONTROL DELETE scroll the whole screen instead of only the line containing the cursor.

This TI-FORTH application program was designed and written by Jean-Pierre Morin. Have a good time!

JP GRAPHIC V3.1 ADDENDUM

V3.1 adds music to V3.0. To make room in memory for the music routines, the following functions were removed:

- F04 (HOUSE), F08, F12 (FLOWER), F09, F11, F13 (TREE).

These functions are available in source code and can be loaded into memory if desired. To load them, press "FCTN CLEAR" to exit drawing mode and type:

- 37 LOAD to load F08, F12 (FLOWER) AND F09
- 38 LOAD to load F04 (HOUSE)
- 39 LOAD to load F11, F13 (TREE)

Type DESSIN or DESSIN2 to go back to drawing mode.

Here is the DISK ORGANIZATION by BLOCK:

0	RESERVED FOR FILENAMES AND BIT MAP
1	FREE BLOCKS
2	DEMO PROGRAM
3	BOOTING SCREEN
4-5	ERROR MESSAGES
6-7	26 CAPITAL LETTERS ROUTINES
8-19	FORTH SYSTEM
20-36	JP GRAPHIC BINARY IMAGE PROGRAM
37	F08, F12 (FLOWER), F09
38	F04 (HOUSE)
39	F11, F13 (TREE)
40-48	FREE BLOCKS
49-60	FOREST SAVED IMAGE (P1=05)
61-72	HOUSE SAVED IMAGE (P1=06)
73-79	FREE BLOCKS
80-81	MUSIC SOUND LIST
82-89	FREE BLOCKS

To stop the music, exit drawing mode with "FCTN CLEAR" and type:

```
0 >83C4 ! 7 EMIT
```

Disk 61. Funnelweb disk #2

Version: 4.12 *Author*: Will and Tony McGovern
Requires: *Language*: *Updated*: 10/28/88

This is the second disk of the Funnelweb system. Requires disk #46.

```
dskdir. v2.0. 12-dec-96
Disk name               = FUNL4*1B
Sectors total           = 360
Sectors used            = 344
Sectors available       = 14
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -READ-ME     21 DIS/VAR 80   >022 020
002 >003 C99PFI;O      2 DIS/FIX 80   >036 001
003 >004 CF           31 PROGRAM      >037 030
004 >005 CG           25 PROGRAM      >055 024
005 >006 CP            4 PROGRAM      >06d 003
006 >007 CT8RAM        9 DIS/FIX 80   >070 008
007 >008 FMSAVE        6 DIS/FIX 80   >078 005
008 >009 FWDOC/EASM   33 DIS/VAR 80   >07d 032
009 >00a FWDOC/LOAD   36 DIS/VAR 80   >09d 035
010 >00b FWDOC/REPT   50 DIS/VAR 80   >0c0 049
011 >00c FWDOC/TIWR   24 DIS/VAR 80   >0f1 023
012 >00d FWDOC/UTIL   60 DIS/VAR 80   >108 059
013 >00e FWSAVE        6 DIS/FIX 80   >143 005
014 >00f LDFW          9 DIS/FIX 80   >148 008
015 >010 LDSR/S        4 DIS/VAR 80 Y >150 003
016 >011 LGEN/S        8 DIS/VAR 80 Y >153 007
017 >012 LH           12 PROGRAM      >15a 011
018 >013 SAVIT         2 DIS/VAR 80   >165 001
019 >014 XB4THLD       2 PROGRAM      >166 001
```

Disk 61. Contents of file -READ-ME

Funnelweb Vn 4.10 May 30, 1988

Universal Utility Environment

Memorial Day (#25) 1988

(1) General Notes

Funnelweb Vn 4.10 has been prepared to allow the TI Extended Basic module or Myarc XBII to provide a utility environment for TI-Writer, Editor/Assembler and Disk Manager utilities with some welcome improvements. It will also load one way or another from just about any assembly file loader. It gives seamless operation of a wide range of utilities, and the need for module swapping has been greatly reduced. It should also be Geneve 9640 compatible. Funnelweb can be used even in a system with only XB, 32K, and a single SSSD drive, by having several system disks for various purposes. It is even better in a system with DSDD drives and multiple RAMdisks.

The program has been written entirely at Funnelweb Farm and is distributed as "fairware". It is not to be sold nor distributed with excessive copy fees, nor made part of any commercial sale, nor placed on a copy-protected disk. Placing of these files on any electronic network or BBS without explicit permission (to be renewed for each new version) of the authors is expressly forbidden.

When you pass it on, please do so as a complete unmodified set of files. The Vn 3.0 and later programs are NOT in the public domain, but "fairware", with all rights reserved by the authors. No responsibility is accepted for consequences of its use. Please refer to the Fairware Notes at the end of this file. The Funnelweb package is issued with no copyrighted utility programs on it, and we request that the package be transmitted in its original form only.

The equipment that you will need to run Funnelweb is the same as needed for TI-Writer or E/A except of course that the specific module is not necessary.

Absolutely necessary

(1) TI-99/4A console
(2) 32K memory expansion
(3) Disk drive + controller

Highly desirable

(4) Two or more disk drives
(5) RS232/PIO and printer
(6) A RAMdisk or several

Other files that you will need to make full use of Funnelweb Vn 4.10 are

(1) E/A disk files
(2) DM-1000 docs
(3) c99-REL4 package
(4) Your utility files

TI-Writer and E/A manuals remain copyright items and may still be obtainable from TI. The program is designed to allow easy loading of various utilities.

The DM1000 files are part of the "fairware" DM-1000 package released by the Ottawa TI-99/4A User Group. Contact their Treasurer for further details. The MG/MH program files provided have been modified here from the last source code (Vn 3.5) sent to us by the Ottawa Group.

(2) Documentation

Detailed documentation of the various aspects of the program is to be found in the FWDOC files.

(1) FWDOC/LOAD :- General information on system and disk organization, the XB user's list, and program-wide facilities.

(2) FWDOC/TIWR :- Use of the program as a substitute for the TI-Writer module.

(3) FWDOC/EASM :- Source code edit and assembler operation, and program loading functions.

(4) FWDOC/UTIL :- Discussion of other programs used with and/or modified for Funnelweb.

(5) FWDOC/REPT :- Chronicle of bugs, fixes, updates, problems, and background information. Make sure you read this file.

(3) Fairware Notes

Your letters and contributions in appreciation of this program will be welcome. Many suggestions from correspondents have found their way into the program already. If you wish to interface to Funnelweb at the assembly language level, contact us for a file of EQUates and information.

We can suggest only that you judge the program on its intrinsic merits, perhaps measuring its worth by how much you use it as compared to other "fairware" or commercial programs that you use. Even individual components of the package are substantial programs. The source code of CF/CG for instance runs over 540 disk sectors. If contributions are made by check or IMO (the best way) they should be made payable to Tony or Will, or Will alone (rather than "and").

All letters needing an answer will be responded to sooner or later, but we just can't afford the time for or cost of mailing updates and don't volunteer to do so. If and when updates are issued they will be placed in distribution through our regular, and/or recent contacts, and we do not offer to provide distribution from Funnelweb Farm at near nominal cost or less. If you write new utilities for Funnelweb please to let us know the details.

May 30th, 1988
Tony and Will McGOVERN
215 Grinsell St.,
Kotara, NSW 2289
AUSTRALIA

Disk 61. Contents of file FWDOC/EASM

Funnelweb Vn 4.10 E/A EMULATION

(1) Source Editor

If the Central Menu Screen shows the Assembler side, selection of Editor loads the TI-Writer Editor modified for use as a source code Editor. It is generally more convenient with these modifications than the E/A Editor.

(i) The Editor now functions with word-wrap disabled, E/A tab defaults set, and files are saved to disk with no final tab record appended. Incoming tab records are still recognized. The disabling of word-wrap may spare you the distressing sight of 99 sectors of source code reformatting into one giant paragraph.

(ii) If a DIS/FIX 80 file must be written to disk, say in editing of uncompressed object code, use the PF option as F DSK etc. instead of SF. This is described in the TI-Writer manual.

(iii) The Source Editor loads CHARA2 as its character set from the E/A system drive. As supplied this contains a larger set than CHARA1, quite suitable for Assembly source which has a high proportion of whitespace. If this set is not wanted, copy CHARA1 or character file of your choice over CHARA2. The 5-sector form saves disk space, but the 9-sector files from TI-Writer may be used.

(iv) Pressing <ctrl-B> splits the current line much as <fctn-2> does in word-wrap mode in the TI-Writer editor. The effect is not recoverable except by retyping the blank part of either line.

(2) Assembler

This loads the E/A Assembler, in a version modified for Funnelweb.

(i) The filenames remain visible on the screen while the Assembler is executing. Some default entries are provided.

(ii) The AID <fctn-7> key calls up the Quick Directory routine at any time during the filename entry process. For this to be available file QD must be already loaded or present on the disk in the boot drive when the Assembler is loaded. File marking is not available here.

(iii) If a filename is found in the mailbox by the freshly loaded Assembler, it is written up as the source filename, and with the last two characters altered or truncated as the object filename. On second or later runs a default is supplied as it is unlikely that the same file would be assembled twice without editing in between.

(iv) The source filename is passed back to the Editor via the mailbox, and is immediately available for LF when the Editor is invoked after an Assembly.

(v) The object filename is passed back to the Object file loaders, and appears as default.

(3) Program File Loader

Selection of Option 3 from this central menu screen sets up a selection screen with 8 choices of Load environment shown. You will need to return to the Central Menu Screen to reactivate the AID <fctn-7> key for QD after loading any files as Funnelweb cannot readily know if QD will leave other programs unscathed or the reverse. Filenames entered may use disk volume names and be up to 25 characters long. Volume name access will not work for Horizon RAMdisk cards at CRU base >1200 and above. This also applies for object filenames.

(i) Option 1 emulates the TI-Writer module which hands over control in the GPL workspace, with Text mode set, and with a full set of characters loaded from GROM even if they are immediately overwritten by CHARA1. The utility must be in E/A program file format. An attempt is made to load CHARA1 from the boot drive, but no error is issued if it is not found.

(ii) Option 2 sets up a GPL type of environment adequate for most programs normally loaded by the E/A RUN PROGRAM FILE loader from GPL. Handover is in the GPL workspace and the presence of the E/A utilities cannot be assumed.

(iii) Option 3 supplies the E/A RUN PROGRAM FILE function for Program files that have been prepared from E/A object files which do use the E/A utilities. Handover is in the USRWSP at >20BA, but this is not written to after file loading starts so that it does not corrupt files which ignore the E/A utilities and load over them anyway.

The utilities are loaded if file EA is present on the disk in the boot drive when the option is selected (not necessary when running from the E/A module). If it is not found a warning honk is given and a discreet little message pops up. GPLLNK has been modified to work with Funnelweb (see FWDOC/REPT for details) and the first free address in low memory is unaltered.

(iv) The program file loaders will handle cassette files with CS1. as filename as described in the E/A manual.

(v) Pressing REDO (fctn-8) gives immediate re-entry to a program file without reloading it from disk, assuming the programmer has deemed it safe to set the appropriate flag, as in DP.

The last program file in a sequence may overwrite Funnelweb at the top of high memory without trouble while loading, but if a utility overwrites Funnelweb either in loading or while running, it should return to the Title Screen on exit or else reload Funnelweb.

(4) Object file loaders

LOAD/RUN handles E/A object files, compressed or not. Files which load over Funnelweb in the top of high memory lock up the computer, and may be loaded with Script-Load instead. Otherwise programs written strictly to E/A manual specifications should run correctly. Common sources of difficulty are discussed in FWDOC/REPT. The EA file must be on the disk in the boot drive when this option is chosen or the warning is given (unless running from the E/A module). DEBUG and SAVE from your E/A package both work normally.

(i) Options 5-8 give variations on Load/Run. The normal Load/Run option sets the last free address in high memory (LFHM) pointer in UTLTAB to protect Funnelweb as far down as the User List code and/or FMSAVE (currently >E98F).

(ii) SCRIPTLOAD (Option 5) is a batch file loader for object files. See FWDOC/UTIL for details of SL.

(iii) LOW-LOADER (Option 6) allows object files to be loaded starting in low memory at >2000, and then continuing in hi-mem. All normal REFed utilities are available as REFs. The REF/DEF table starts at >E200 and builds down from >E138. Code must not be AORGed above this. FMSAVE recognizes Low-Loaded files. See FWDOC/UTIL for details of LL.

(iv) If Option 7 is selected the LFHM is reset to the E/A default of >FFD7. This allows relocatable autostarting object code to load over Funnelweb if necessary, there being no memory contention because the autostart hands over to the program without returning to Funnelweb.

(v) Option 8 intercepts the Autostart of object files and the DEF table is displayed as for a normal file load. The LFHM is as for Option 4.

The Run part of the Load/Run procedure generally follows E/A conventions except that more information and help is provided along the way. Pressing <enter> without entering any filename transfers to the RUN function.

RUN is activated by cursor driven selection from a screen display of the DEF table. The DEF table may be inspected at any time during a multiple file load by pressing <enter>. <fn-S> and <fn-D> drive the cursor through the DEF entries, while PROC'D (or <ctrl-A>) transfers control to the program entry marked by the cursor, and REDO returns to load another file. Pressing BACK, <fn-9> or <ctrl-C>, aborts the load sequence.

Disk 61. Contents of file FWDOC/LOAD

Funnelweb Vn 4.10 LOAD INFORMATION

(1) First Things First

Before doing anything else, make a working copy from the distribution package. Keep the original safe as a master copy for backup and for passing on. All files may be copied in the normal way. Copy the frequently used short system files on to a fresh disk first to speed access in use. The usual distribution format is as DSSD.

If you have only SSSD disk drives, and do not wish to use the E/A functions, then QD, CHARA1, UL, DS, LOAD, ED/EE, FO/FP, MG/MH should be included, with loader utilities EA, SL, LL etc as required by your User Lists. SSSD users will find it convenient to have different partial system disks for various purposes.

(2) Loading Funnelweb

With XB and XB-II just auto-load DSK1.LOAD. Auto-loading as DSK1.UTIL1 is available from E/A if RUN PROGRAM FILE is selected and <Enter> pressed with no file-name. UTIL1 under name FW gives a easy CALL with the Miami UG ROS/MENU for Horizon style RAMdisks, and either LOAD or UTIL1/FW or both can be installed as MENU entries. The Utility option of TI-Writer also provides auto-load of DSK1.UTIL1 and the name is easily edited. Object file LDFW auto-runs from any compressed object file loader.

LOAD is a hybrid of Extended Basic and machine code and must only be edited by means of the CF/CG utility program, which is largely self-prompting in use. Direct editing must NOT be attempted. Use a disk manager to transfer LOAD from disk to disk (it can be reSAVEd unedited with TI XB but not with Myarc XBII).

(3) Boot Disk Tracking

Funnelweb either as XB LOAD or as UTIL1/FW searches the disk controller for the drive number used to load it, and writes this into the system file-names. This means that the Funnelweb disk may be used from any drive if booted from that drive. This works with most disk controllers. If the search fails the default is used. From loaders which first ask the user for drive numbers, the ones supplied are used. If a number key (1-8) is held down as the program starts to execute when loaded, that number is used instead for the system drive(s).

(4) Source Disk Specification

LOAD/UTIL1 has an escape hatch which disables boot tracking and allows the system drive to be specified directly. This is mostly intended for RAMdisks which do not follow the unspecified details of TI disk controller operation, and also allows separate source disks to be specified for TI-Writer and E/A system utility files, when a single SSSD or DSSD capacity is insufficient. CF/CG allows setting of these drive numbers as a single character "x" in the form DSKx. for each source disk. Forms such as RD. are not supported.

The secondary system drive number is for the E/A side of the Central Menu. If separate disks are specified, Funnelweb looks for the system files in the appropriate drive first before checking the other one. CHARA1/2 are the only system files which must reside in the particular drive. With this exception the system files can reside on either disk but fastest loading occurs when they are correctly partitioned.

(5) XB Selection screens

After auto-loading from XB a title screen appears while QD is loaded. The title screen will time out if no key is pressed first, to a screen with 18 choices, 9 by number and 9 by letter. The first 3 are preset internal paths and the remaining 15 are configurable. The first two choices, TI-Writer and Edit/Assm each lead to the Central Menu screen with the corresponding mode set. When these paths are taken the XB environment is destroyed and cannot be restored except by rebooting XB, which takes just 3 easy keystrokes.

(6) User's List Selection

The remaining 15 entries have names which are entered in the LOAD program by CF/CG. On selection of one of these 15 user list items Funnelweb loads the program configured into that option. This may be an assembly program, XB RUN of another XB program, or just return to the XB command mode. LOAD may be reRUN from the command line to allow recovery from XB RUN errors without complete reload from disk. If filenames are not of form DSKx., then boot disk tracking must be suspended. Some entries have been predefined for convenience and example.

At this stage the set of XB INIT utilities is still available for either XB programs or for assembly program files (options 1 or 2). These have been augmented by a DSRLNK (BLWP vector at >24F4 with function as defined in the TI-Forth source code), and a GPLLNK (BLWP vector at >24F8).

(7) Universal functions

(i) Screen colors

On almost any screen which invites selection by number, pressing 0 (zero) will cause the screen colors to cycle through the 10 configured color choices.

(ii) Quick Directory

The AID key, <fctn-7>, will call up a disk directory routine with some of the functions of that used for SD in the Editor. This is also available within the Formatter and Assembler. File QD must be on the Funnelweb boot disk, but do not load QD by name. The QD file will be reloaded if necessary. Pressing keys <1-9,0> as indicated on the current directory page will cause the filename indicated to be marked and written to the mailbox as workfile name if a Display file, as the program default if a program file, and to the object default if a D/F file. Any file may be marked for deletion.

(iii) QUIT

The system QUIT key is enabled at all times when keyboard entry is called for by Funnelweb itself. Each utility determines its own response to QUIT.

(iv) BACK/Ctrl-C

In Funnelweb itself <Ctrl-C> may be used instead of <Fctn-9> (BACK) just as in the Editor, and <Ctrl-A> usually substitutes for <Fctn-6> (PROC'D). Under some error conditions these alternatives may not be available.

(8) Central Menu Screen

The prime focus of the program are the Central Menu screens which offer 8 choices on the TI-Writer entry (hit the space bar if necessary).

1 EDITOR
2 FORMATTER
3 DISK UTILS
4 DATABASE
5 MODEM
6 ,,
7 ,,
8 USER LIST

The normal TI-Writer Editor is loaded from this screen. Entries 4-7 are completely configurable, and may be any Assembler program (with a two letter filename) that Funnelweb will handle, including Script-load files. User list files are of Opt 2 type, and allow an tree of choices as big as your disk system will allow. To avoid overwriting the formal User List (#8) save as Other type when configuring. Choice #3 is set up with a List file DS as example, grouping a disk-related utility set. USER LIST presents a third screen of user entered options. See the CF/CG instructions in FWDOC/UTIL.

A second menu screen comes up from other loaders or may be toggled with the first screen by hitting the space bar or any letter key. This screen contains the programmer's workbench.

1 EDITOR
2 ASSEMBLER
3 LOADERS
4 c-COMPILER
5 DISKPATCH
6 LINEHUNTER
7 ..
8 RESET/QUIT

As with the other screen, Options 4-7 are completely configurable. When option 8 is first taken it cancels any pending conditions, places the current filename in the mailbox, and resets memory pointers and system flags. It then shows Quit and a second press returns to the title screen. This is the recommended way to leave Funnelweb if the machine is not to be switched off immediately. As issued Option 5 loads a modified DISKO, and Option 4 the c99 REL4 compiler if the appropriate files are on your working disk. Option 6 LINEHUNTER is a new utility to help programmers use the Funnelweb system. Option 3 enables the program loading screen (see FWDOC/EASM). The Editor is TI-Writer modified for source code editing. If dual system drives are specified it is not necessary to repeat ED/EE on the secondary disk if the extra loader delay can be tolerated. See FWDOC/EASM for E/A details.

Disk 61. Contents of file FWDOC/REPT

Funnelweb Vn 4.10 BUGS & PROBLEMS

(1) Bugs and Mods

The bug report is being zeroed out at the time of issue of Vn 4.10 of Funnelweb, but new bugs are being created all the while. That's the price of progress! Fixes and changes since that release are listed in chronological order below. Minor changes in associated document files may well be made without explicit mention. This (Memorial Day) issue is the first to be sent outside Australia after a pre-release at the Brisbane TI-Expo.

Known Bug ..... The lockup with use of RE on an empty buffer in Editor is present in the *fix 1 files as issued by TI. Silly thing to do anyway !

If you come across a later issue of Funnelweb, use it, but it is a good idea to copy the whole disk. Significant changes are indicated in the update notices. Use CF/CG and your SYSCON file to re-customize the LOAD and UTIL1 files if these have been updated.

(2) Programmers' reference

The Funnelweb program has to interact with a number of external programs which (to TI's eternal shame) were never intended to work together, so its internal structure is an ad hoc response to many independent pressures. As the scope of Funnelweb grew it became necessary to define an interface so that other parts of the program could call on routines or data in the main body of the program. Some of the routines may also prove useful in other programs as well. Write for details.

The Funnelweb LOAD program is composed of 4 parts, the Extended Basic code, the XB user list data, and 2 pieces of machine code tucked between the XB code and the top of memory. DSRLNK (TI-Forth form — you handle the errors) and GPLLNK are in a >FA byte block immediately above the XB program code. These are shifted to follow the normal XB Utilities and the memory pointer updated, and are available for use by XB programs loaded from the XB User List (BLWP at >24F4 and >24F8 respectively) provided CALL INIT or CALL ILR are not invoked. This is followed by code and data used only while XB is still preserved. This same area is used later by UL and FMSAVE. When UTIL1 is prepared the QD routine and UTIL1's entry code are tucked between the entry point (>E006) and the actual Funnelweb code to speed up loading.

The end of the program is kept fixed at >FFD7 so the interface block is at known addresses. The program is pushed as high in memory as possible for obvious reasons, the >FFD7 limit being compatible with E/A and XOP 1 which is not used by Funnelweb (which does make transient use of the area above >FFD8, but does not alter the Load Interrupt vector at >FFFC).

(3) GPLLNKs with Funnelweb

Whenever the central menu screen is displayed the following conditions have been established.

(a) The GROM address is left pointing to an XML instruction in cartridge GROM, else in console GROM 0, and this address is also saved in the program.

(b) The GPL stack is dropped to just one entry which is this value copied to >8380.

(c) The table address pointed to by the XML instruction is loaded with the Funnelweb re-entry address. For maximal compatibility with existing software the XB return is at >2000 and the E/A return is at >2002. All others use PAD (>8300).

With these preparations it was possible to write a GPLLNK to fit invisibly in the E/A utilities without altering the FFALM pointer. In any event the E/A GPLLNK cannot be used because it branches to E/A module GROM code. Programs run other than from E/A or XB under Funnelweb may give trouble if not written to preserve the contents of location >8300.

(4) Returns

The easiest returns to Funnelweb are when Funnelweb is still all there unmolested, from >EBC8 to >FFD7. Funnelweb always hands over with R11 loaded with the return address. Any of the standard returns to GPL will return to Funnelweb provided that the XML table entry is preserved for B at >70 returns.

If Funnelweb is overwritten the simplest return is to the Title Screen. If your program doesn't make extensive use of VDP memory, it may be possible to stash Funnelweb (from >E9B0 to cover UL or else from >EBC8 to >FFD7) there below the disk DSR buffers and to restore it to CPU RAM after an irrevocable decision to exit has been made, and then a normal exit as if to GPL may be used.

If memory usage is too complete to allow this, Funnelweb may be reloaded from disk as the program file FW or UTIL1. Only a very simple loader is needed as there is only a single file and the memory location and start address, (>E006) and the length (through >FFD7) are prior knowledge. Only Atrax Robustus can get away with not deriving the start address from the file header in case of future changes but it is not anticipated that UTIL1 will ever extend to more than one file. It is recommended that the loader first try to load the file under name FW and if this is not found then to try it as UTIL1 before taking error action. If the application program is such that disks may have been swapped around, prompts should be issued for the primary and secondary disk numbers or the original drive assumed, as appropriate. The flag to turn on boot disk tracking in UTIL1 is at >FF1A. A null word here disables the tracking, in which case primary and secondary system drive numbers in ASCII form must be provided in the bytes at >FF18 and >FF19. Rewrite the drive numbers after loading but before handing over if these have been changed. Boot disk tracking should be suppressed for loading from Myarc RAMdisk. The version of DM1000 supplied follows this prescription.

Programs intended to work only with Funnelweb may re-enter it at the Central Menu Screen by LWPI of the FWB main workspace at >FF7C, SETO of R13, and CLR or SETO of R4, followed by a branch to the address contained in location >FF5C. A program (Opt 1-3) can tell that it was loaded from Funnelweb by comparing R11 at entry with the word at >FF9C.

(5) Filename transfers

Funnelweb has a standard way of transferring the D/V80 workfile name from one utility to another. The file name is stored on final exit from a component program at >A000 without any length byte, and padded with spaces out to >A050. This is the "mailbox". A file name so stored will survive OLD or RUN of an XB program or passage through the Title Screen. The utility entry routines inspect this location for the 3 ASCII characters "DSK" or "RD." and if this is found the block as far as the first space is written to the appropriate buffer in the utility program.

If utility programs are not to destroy the resident file-name then they should not overwrite this area, or else should restore it before exit, or load the file name to be passed back. Test with Formatter and Editor to see if this has been done properly. Modification of existing programs not available as source code may require a little ingenuity, and is not always possible.

(6) Problems

Various problems have been observed by users of Funnelweb. Many of these are intrinsic to the programs being loaded, the Formatter being prime catalyst for suggestions for improvements. These we can't do anything about. If you have commercial programs which you would like to use with Funnelweb but can't for one reason or another, particularly because of protection schemes, you should contact the sellers or originators of these programs to make your problem known. Good luck Charlie Brown ! Other problems are associated directly with use of Funnelweb.

(a) Loading difficulties

Lockup during file loads occurs because some part of Funnelweb has been overwritten. The code for program file loads is located close to the top of high memory and will be destroyed if overwritten by a file that is not the last one to be loaded in normal E/A sequence with the last byte incremented for each successor file. Altering the file sequence to put the overwriting file last in a multiple load will usually solve the problem. Use Disk-Patch to correct the load flag in the first word of the files being interchanged (>0000 for the last file, >FFFF for all others) and a disk manager to interchange file names. File lengths of >2000, such as produced incorrectly by E/A SAVE, may need to be corrected to >1FFA if the changed order would result in overwriting of previously loaded code.

Load/Run of object files is handled by the Loader in the E/A utilities. Autostarting of files is by a direct branch from the Loader to the program code. Files that are too long may refuse to load. Unsuitable absolute file addresses may destroy the code for DEF table entry.

(b) Running difficulties

The program file loader gives a fair approximation to GPL or E/A module environment. If problems are observed they are usually associated with key unit choice and key response in the program loaded, particularly with programs that set a key-unit of 0 instead of choosing the one they really want. The GPL program file and cartridge loaders now reset the GPL stack pointer at >8373 to >7E which seems to help.

Programs that sense which module they are running with and adjust their action accordingly can cause problems. Since the point of Funnelweb is to make all sorts of programs work with modules they weren't intended for or even no module at all, it has already taken the necessary actions and must not be second-guessed by programs it has loaded. The only solution is to modify the programs or else not use Funnelweb. Running from E/A gives the easiest way to switch back and forth between Funnelweb and a standard environment.

Some programs in E/A program format contain code to unload the standard utilities from the E/A GROM because the E/A module does not do this for Run Program File. The c99 memory image files prepared with the original unmodified C99PFI use this method to force the utility load. Running or return from such programs may not be graceful if Funnelweb was loaded from E/A. c99 is smart enough not to try getting code from non-E/A GROMs, but other programs may need revision.

Funnelweb has also adjusted the E/A XML object file loader path for its own purposes. If an assembly program needs to load an object file it should use a direct BLWP at LOADER to the E/A routine. Error returns may need special attention.

(c) Exit difficulties

Read (4) above on returns. Opt. 2 loads will be necessary for program files that overwrite the E/A utility area in low-mem but still use a GPL return via the contents of R11 as would be acceptable for a Opt. 2 (or E/A 5) program file load.

Some programs which do not appear from their load size to overwrite Funnelweb may in fact corrupt it while running. Some particular cases of pre-existing programs may be helped with minor changes, as a return to the title screen is usually preferable to a lockup.

An example of this kind is the Dragonslayer Spellchecker. An early version inspected of this commercial program runs normally under Funnelweb with load parameter 2, but crashes the machine on exit. This is improved by changing the words 0460 0070 to 0420 0000 to return to the title screen. Use a sector utility with string search capability to find this exit.

Disk 61. Contents of file FWDOC/TIWR

Funnelweb Vn 4.10 TI-Writer Loader

If you are not already familiar with the operation of TI-Writer some help is available from various User Group sources. The original manual may still be available from TI in the USA. Some tutorials are also available as 'fairware', such as "TI-Rewrite" by Dick Altman, 1053 Shrader St., San Francisco, CA 94117. The full multilingual capabilities of the Vn 2.0 European release of TI-Writer are not yet supported by Funnelweb.

(1) Editor

The Editor works in all respects as it does with the TI-Writer module, when selected with Formatter listed on the Central Menu Screen, with these improvements. There has been NO change from the original Editor in the maximum size of files that may be loaded or written.

(i) The color selections using <ctrl-3> may be configured by the user. All 10 are available in the Editor.
(ii) A printer device-name is pre-loaded into the Editor by the loader and may be configured to suit your own system.

(iii) A default Edit filename may be configured. If left blank the default DSKx. or the pre-existing filename will be set, where x is the default utility drive number.

(iv) The <fctn => system Quit key remains disabled at all times while in the Editor, including SD.

(v) SD handles disks with up to 127 filenames. Paging is controlled by the use of the single <N> or <B> keys. Pressing <=> causes the program to check and indicate the type of program files on the disk.

(vi) An asterisk after the file length shown in the directory indicates a fractured file, one stored in more than one contiguous block on the disk. A full or nearly full disk will usually have one or more fractured files because of the way the file system allocates space.

(vii) Files may be marked for the LF/SF function and <V>iew or <D>elF by pressing the number (1-9,0 for 1-10) of the entry as displayed on the current page. Pressing <O>ldF restores the workfile name existing on entry to SD.

(viii) If <P> is pressed SD prints the directory in two column format to the Editor print device. This may be a disk file, and the device is opened in Append mode. Program file check has to be called first if it is to be printed out as part of the directory.

(ix) If <V> is pressed SD will "View" the DV/80 file with name currently in the mailbox to the screen. Pressing any key pages through the file, and <BACK> or <ctrl-C> aborts the "View". This facility is handy for quick checks of the contents of other files. Remember to <O>ldF your workfile name before exit so that you do not save your workfile over another in a careless or hasty moment.

(x) Pressing <D> for <D>elF will delete, after checking, the currently marked file. It is inactive if no file has been marked. The file list and sector counts are immediately updated.

(xi) REDO causes SD to re-read the directory from the same drive. The <O>ldF buffer is not affected.
(xii) The CHARA1 file has been altered to improve legibility. If further changes are needed, say to support a language other than English, other files may be substituted.

(xiii) If either Editor is reloaded as first choice after quitting, re-entry does not need reload from disk. The character set is not changed and the text buffer is cleared. Reloading like this offers a way to switch between CHARA1/2 without altering the files on disk, which may be handy for bilingual usage.

(xiv) Screen rewriting has been speeded up slightly to give "crisper" screen scrolling. Delay in word-wrap has been reduced so that there is less problem with loss of keystrokes.

(xv) A right margin warning beep has been incorporated. The inset from the right margin may be set, or the beep canceled, but it requires disk sector editing of the second word in the ED file (after the 3 word file header). It is set to 5 (0005) in the distribution version. A null word here cancels the beep.

(xvi) Alpha case conversion is now available. <Ctrl-;> converts a lower case letter under the cursor to upper case, and <ctrl-.> converts upper to lower case, with auto-repeat.

(xvii) The End-of-File message has been replaced with a full width ruler line which shifts with window and line number selection.

(xviii) Page up and down are now also on keys <ctrl-Q> and <ctrl-A>. Used with the <ctrl-ESDX> cursor block and <ctrl-C> this gives convenient one-hand operation without the stretching needed for the <fctn-key> equivalents.

(2) Formatter

The Formatter functions normally with the following exceptions.

(i) The printer device-name is preconfigured by CF/CG and may be changed to suit your convenience.

(ii) The Formatter will display the filename last used or marked. If it cannot locate a name then DSKx. is the utility drive number default.

(iii) The <fctn-9> key now returns directly to the exit which allows return to Funnelweb's central menu screen, also reached with <enter> after a format job has finished. The Formatter does not need to reload from disk if reselected.

(iv) Pressing AID <fctn-7> invokes the QD directory routine. File QD must be already loaded or present on the boot disk when the Formatter loads for this to be available. File marking is inoperative here.

(v) The formatter may now have 4 disk files open at the same time.

Disk 61. Contents of file FWDOC/UTIL

Funnelweb Vn 4.10 UTILITIES

Utility files of various origins are included on the Funnelweb distribution disk(s). Files LGEN/S and LDSR/S from Vn 4.0 are not included as not a single comment has been received concerning their existence or use. This file contains notes on the following programs:

1 CF/CG
2 DP
3 MG/MH
4 CP
5 FMSAVE
6 LDFW
7 UL
8 CT8RAM
9 SL
10 LL
11 LH
12 XB4THLD

(1) Configuration CF

CF/CG is used to customize LOAD, UTIL1/FW and various User List files to your preference in the run-time setup of Funnelweb. The program makes extensive use of windowed displays and help screens are available at many points in the process. The editing process is tree structured and is easy to follow along to any particular item. Your system configuration details are saved for re-use in data files of which SYSCON is an example. CF loads as an Option 2 program file from Funnelweb, which is used as a source of default data but the program in memory is NOT altered.

(2) Disk sector editor DP

This is based on DPATCH aka DISKO, issued by TI to user groups after orphaning the 99/4A. DP has had extensive modifications to make it easier to use. Key functions have been changed around a little, for better or for worse. <Ctrl-C> parallels <Fctn-9> for some purposes. Both the original and DSKU (J. Birdwell) pattern are active. With Myarc controllers a read of sector #0 is forced so that 16/18 sectors/track can be distinguished.

If QD is already in memory when DP is loaded then a third choice for disk directory appears. This is just a bonus and its absence is NOT a bug. Exercise AID first if you want to make sure it appears.

(3) Disk manager DM1000 Vn 3.5

See -READ-ME for "fairware" advice. This has been reassembled from the Vn 3.5 source code from the Ottawa UG to improve its interface to Funnelweb. We have independently fixed the bugs in Type and Print (lockup on null records and it now always closes files), fixed a bug of unknown effects in file copy, improved printer codes entry (3 digit codes), high-CRU Horizon cards are supported except for initialization during Disk-Copy, and all DD operations with Myarc disk controllers are at 18 sectors per track in either SS or DS. The program has been shortened slightly without change to its structure, and <ctrl-C> added as alternative to <fctn-9>.

** WARNING ** With a Myarc disk controller present any operation involving formatting at double density (Initialize or DiskCopy) will go directly to the physical disk 1 to 4 and format that, ignoring any RAMdisk emulations, but the disk header sectors will be written to the emulating disk if any.

The option to reload Funnelweb first asks for primary and secondary drives with the ones at entry as default. and then tries to load the FW file or else UTIL1 from the secondary drive. If neither of these files is found then it will use XB LOAD in the primary drive as its source of the Funnelweb code. This allows use of XB with standard TI-Writer functions and DM-1000 without UTIL1 on the disk.

The program files have been renamed MG/MH to avoid confusion with the originals. Saving details to disk from DM-1000 will go to the Funnelweb boot drive, and defaults to drive #1 if MG/MH are loaded otherwise. This default may be modified with DPatch in the filename just after the start of the MG file.

(4) c-Compiler interface CP

Clint Pulley's Vn 4.0 c99 compiler files C99C/D/E may be loaded directly as Option 3 program files. File CP brings user convenience in working with c99 closer to Funnelweb standards . Instead of loading c99 directly, load CP as a Option 3 program file, and it will then load C99C/D/E from the same disk drive. The return from c99 reloads FW or UTIL1 from the E/A side boot drive and preserves the mailbox filename. If no filename was initially present, the c99 source code filename is installed.

(5) Save utilities FMSAVE FWSAVE

The E/A SAVE utility loads as absolute code in low memory. Funnelweb is compatible with SAVE, but does take up its own 6K share of high memory, so the FMSAVE utility has been prepared to allow SAVEing of object files loaded by Funnelweb, including into low memory. Refer to the E/A manual for general information.

FMSAVE and FWSAVE load as absolute code overlaying the Funnelweb (UL) system area. The start and first executable instruction should be DEFed with SFIRST and the last address DEFed by SLAST. Select entry point SAVE and enter the SAVE filename. FWSAVE, also provided, does not recognize Low-Load but will accept CS1. as a filename and has a LSAVE entry point for longer than standard files. The Funnelweb loader will accept program files up to >27D0 bytes long.

If the Loader has placed files so that SFIRST is in hi-mem and SLAST is in low memory, FMSAVE will SAVE high memory from SFIRST to the FFAHM indicated by the Loader at UTLTAB+2 and then proceed to SAVE low memory from >2676 (above the E/A utilities) to the FFALM. The utilities are not included so that the files will remain compatible with Funnelweb if reloaded under a different module.

When used with Low-Loaded (Opt 6) files, FMSAVE saves its first module from low-mem from SFIRST to the top of lo-mem, nominally >3FFA (at UTLTAB+4), and then from hi-mem from >A000 to SLAST. If SFIRST and SLAST both point to the same segment the SAVE is normal. Use E/A SAVE for addresses in the >6000 to >8000 cartridge space.

The MEMSAV entry point allows direct entry of hex address limits for the memory block to be SAVEd. The second entry is the address of the last word (inclusive) to be saved. MEMSAV ignores SFIRST and SLAST but these must have been DEFed, perhaps by a dummy object file, for correct LOAD/RUN operation.

FMSAVE indicates the actual length of the memory block saved in each file in the second word of the header block, to a maximum of >1FFA in each file. The TI E/A SAVE utility, amongst its other little foibles, adds a further 6 bytes, but the program file loader in the E/A module believes the byte count in the header. In normal usage the extra 6 bytes, falsely indicated by E/A SAVE, as read in from VDP to CPU RAM do not cause any problems. FMSAVE files will of course not cause any problems unless perhaps a loader incompatible with E/A is used.

(6) Basic loader LDFW

LDFW is an auxiliary load program in the form of an autostarting object file which may be executed from E/A, Minimem Basics, Myarc XBII by CALL LOAD("..."), E/A Load & Run, or most other object file loaders such as come with Myarc or CorComp disk controllers. It may be kept in Minimem cartridge RAM if you follow the MM instructions for forcing it to load there. The RUN name is LDFW.

(7) User List UL

Writing in of the 8 user selectable options is done with the CF/CG installer program. There is no entry corresponding to "9 <CRT ROM> 0" which looks for a cartridge ROM header at >6000 and "9" executes the first program listed there and "0" the second. This may be handy for owners of Vn 2.2 consoles who have disk controllers that will load Funnelweb.

Remember that UL is a normally executing Option 2 Program file and different UL files can be chained by specifying them in a UL type of file. Just remember to avoid file name clashes. File DS is a UL type file collecting various disk utilities together for Opt 3.

(8) Cartridge RAM loader CT8RAM

Object file CT8RAM is used to store Funnelweb in >6000 - >8000 cartridge RAM, if present, so that it comes up as a selection after the title screen (not on V2.2 consoles or in the presence of Myarc 128K OS). The code produced is ROMable. Load Funnelweb and then load CT8RAM as a Utility Load / Run option. Funnelweb is loaded into hi-mem on selection.

(9) Batch File Loader SL

SL loads a list of up to 15 object files specified in a DV/80 script file. When SL is loaded as Option 5 SCRIPTLOADERit will prompt for entry of a script file name, but when a User List entry of Script Load type is selected, SL will just load and process the script file specified. File SAVIT is a simple example that shows how to eliminate some of the tedium in preparation of E/A SAVE files. It is possible to load any combination of relocatable object files that E/A will load.

CAUTION .... If the collection of files is such that Funnelweb is overwritten, then return should be to the Title Screen. Even if Funnelweb is unmolested, if files load above >CE00 then 8 RESET should be invoked on return to Funnelweb before using AID for QD as SL does not reset the flag for QD for convenience in LOAD error conditions. c99 users beware!!

Comment lines in a Script start with an asterisk in the first column and blank lines are ignored. Comments may also follow entries except as noted. Details of the load process are controlled by directives. These fall into several classes, some illustrated in SAVIT.

(a) File specification

FILE "DSK.xx" .. Followed by a filename in quotes, single or double, FILE specifies one of the files to be loaded. No spaces are allowed between the quotes.

LAST .. Indicates the end of the script to be parsed by SL. If it is followed on its line by text, the first 6 characters will be read as a link name for RUNning the programs. If the end of the script is reached before a LAST directive is found, an error is issued.

(b) Loader control

AUTO .. If AUTOmatic running is specified the link name specified on the LAST directive will be used to autostart the programs. The default, with no AUTO or ALLM directive, stops for editing of the link name.

IAOF .. I(nternal) A(utostart) OFf cancels internal autostarting of object files (like Option 8).

LWLD .. LoWLoaDer sets up the load conditions as for Low-Loader (Option 6 of the Loaders screen). File LL is loaded from the boot disk after the Script file has been read but before object file loading begins. It should be issued at the start of a Script and over-rides ALLM.

(c) Disk source control

These directives control the assumed source of the object files. Each remains in force until updated by another. The filenames written to the screen have been modified by the directive in force. Unadorned BOOT is the default. BOOT refers to drives established by CF/CG or at boot time, while FIND allows for arbitrary specification.

BOOT .. SL looks for files following this directive on the Funnelweb boot disk. Filenames must be of the form DSKx. so that the boot drive number may be written over the "x". If BOOT is followed by the single letter P, S, or U the primary, secondary, or utility drive respectively is used.

FIND .. FILE-names following this directive are loaded exactly as specified. Use for RAMdisk (RD.) or volume/pathname access. A simple FIND may not be followed by a comment. If the FIND directive is followed by a single character, this is used as the drive number in the DSKx. form.

(d) Memory Control

ALLM .. ALL M(emory) sets the LFHM memory pointer to the standard E/A value >FFD7. Once set it cannot be revoked. It also sets AUTO and a link name must be specified on LAST or an error will be called. Returns are adjusted to be to the title screen.

(10) Low memory loader LL

Some well known utilities for the TI-99/4A, such as the Editor and Assembler occupy low memory and use high memory for extra code, but mostly as a single large data storage area. The only way that TI provided for users to load such files in object format was Minimem, and now Low-loader provides this function for general use, with automatic recognition by FMSAVE.

When LOW-LOADER is selected from the Load Environment screen or is signaled by the LWLD directive in Scriptload the LL file is loaded. This provides an alternate set of E/A utilities just below the Funnelweb program in high memory. Low memory is now used as the first block for loading relocatable object files (only 8K is available for this block). All E/A REFs are recognized, but E/A utilities REFed this way will not be available to program file loads of FMSAVEd versions. The predefined REF/DEF table ends at >E138 and new entries build down from there.

(11) Assembly line locator LH

LINEHUNTER is one of our working tools now made available for Funnelweb users, though it could well stand comparison with many commercial programs on its own. If you write substantial assembly programs you will be aware of the problems in tracking down assembly errors through multiple Copy files. Generation of List files is fine, but impractical for the home computer user.

Give Linehunter the name of your master source file and a line number and it will locate and display the line itself, and the line number in, and the name of the file in which it is located. It will conduct a similar search for a source code label if one is entered instead of a number. The search starts automatically when 4 digits of line number or 6 characters of label have been entered, or else with the <enter> key. Exit from the program is by the <ctrl-=> key.

(13) XB FORTH Loader XB4THLD

This program allows the standard TI FORTH disk to be loaded by TI XB. It works only with the XB and E/A modules, and its primary use would be from the XB User List.

Disk 61. Contents of file LDSR/S

```
*  LDSR/S source file
*  See FWDOC/UTIL for details


CRUBAS EQU  >0     CRU base for DSR
CRUDAT EQU  >0     MSB is byte for LDCR

       REF  UTLTAB

CRUSET EQU  $
       LI   R0,UTLTAB  Find position
       MOV  R0,R0      Where am I ?
       JGT  EXIT       Not for lo-mem
       AI   R0,>1A     Advance to CRU base flag
       LI   R1,CRUBAS  Load value
       MOV  R1,*R0+    Put in place
       LI   R1,CRUDAT  Byte for LDCR
       MOV  R1,*R0     Put in place
EXIT   RT              Slip back unnoticed

       END  CRUSET
```

Disk 61. Contents of file LGEN/S

```
*  LGEN/S
*  ~~~~~~
*  XB LOAD PROGRAM GENERATOR
*  A Funnelweb FARM UTILITY
*  -------------------------
*
*  INSTRUCTIONS
*  ~~~~~~~~~~~~
*
*  Bring up Replace String (RS)
*
*  Enter /XXXX/yyyy/ where yyyy
*   is the 4 digit hex address
*   for the start of the assembly
*   language area
*
*  SaveFile to disk under working
*   file name LG/S and assemble
*   with R option only as file LG
*
*  Clear out top of hi-mem above
*   DEBUG (optional) and QUIT
*   directly (not by .Q)
*
*  NEW out XB
*
*  RUN the following program
*
*   100 CALL INIT :: CALL LOAD
*   ("LG"):: CALL LINK("START")
*
*  SAVE to disk
*
*  OLD to check SIZE and fill
*   with machine code routine
*
*  Do NOT RESequence directly
*
*  Always acknowledge origin
*   of this program.

       IDT  'LOADGEN'

LNTAB  EQU  >XXXX->56

       DEF  START

A      DATA >0064        Line #100
       DATA LNTAB+5      Pointer to line
```

```
       BYTE >51          Length byte for XB line
       BYTE >83          XB ! token

       TEXT 'FUNNELWEB FARM TEMPLATE'
       TEXT 'AVAILABLE ADDRESSES ABOVE XB'
       TEXT '~~~~  >XXXX  TO  >FFD7  ~~~~'
       BYTE 0
B      EQU  $
       EVEN
       DATA >0,>0,>0,>0  Make end obvious
C      EQU  $

START  EQU  $
       LI   R0,LNTAB     Set XB's pointers
       MOV  R0,@>8330    Start of line # table
       MOV  R0,@>832E    Start of line # table
       AI   R0,3
       MOV  R0,@>8332    End of line # table

       LI   R0,A         Write to place
       LI   R1,LNTAB
MV1    MOVB *R0+,*R1+
       CI   R0,C
       JL   MV1
       MOVB @B,@>837C    Clear GPL status
       RT                Return to XB

       END
```

Disk 61. Contents of file SAVIT

```
*  Script-Load example for
*   preparation of SAVE files.
*  Use your own object file name
*   and E/A SAVE if desired.

       AUTO
       BOOT U
       FILE "DSK1.OBJFIL"
       BOOT S
       FILE "DSK1.FMSAVE"
       LAST SAVE
```

Disk 62. RLE Pictures #1

Version: *Author*:
Requires: *Language*: *Updated*: 07/20/86

A general collection of pictures for use with the RLE graphics system (disk #54).

```
dskdir. v2.0. 12-dec-96
Disk name               = PICS
Sectors total           = 360
Sectors used            = 350
Sectors available       = 8
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >008 BAMBI_P      25 PROGRAM    Y >0b0 024
002 >009 BASKET_P     25 PROGRAM    Y >0c8 024
003 >00a CARTUN_P     25 PROGRAM    Y >0e0 024
004 >00b CLOWNS_P     25 PROGRAM    Y >0f8 024
005 >00e ENTERP_P     25 PROGRAM    Y >140 024
006 >00c HOWDY_P      25 PROGRAM    Y >110 024
007 >00d NEEDLE_P     25 PROGRAM    Y >128 024
008 >00f NUDE3_P      25 PROGRAM    Y >158 016 >010 008
009 >002 SCROOG_P     25 PROGRAM    Y >020 024
010 >003 SHELLS_P     25 PROGRAM    Y >038 024
011 >006 SPCWLK_P     25 PROGRAM    Y >080 024
012 >004 SPRING_P     25 PROGRAM    Y >050 024
013 >005 TIGER_P      25 PROGRAM    Y >068 024
014 >007 UWATER_P     25 PROGRAM    Y >098 024
```

Disk 62. Contents of files (graphics)

BAMBI_P

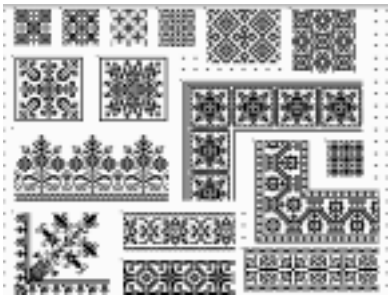
BASKET_P

CARTUN_P

CLOWNS_P

ENTERP_P

HOWDY_P



NEEDLE_P

NUDE3_P

SCROOG_P

SHELLS_P

SPCWLK_P

SPRING_P

TIGER_P

UWATER_P

Disk 63. RLE Pictures #2, Famous Logos

Version: *Author*:
Requires: *Language*: *Updated*: 08/15/86

Includes Coke, Corvette, The Wall, Rocky Horror, Mac(intosh), Middle Earth, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = RLE
Sectors total           = 360
Sectors used            = 353
Sectors available       = 5
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01a -README       3 DIS/VAR 80   >01b 002
002 >005 ABCNEW_P     25 PROGRAM    Y >06a 024
003 >002 BAMBI_P      25 PROGRAM    Y >022 024
004 >003 COKE_P       25 PROGRAM    Y >03a 024
005 >004 CORVET_P     25 PROGRAM    Y >052 024
006 >00b DROP_P       25 PROGRAM    Y >0fa 024
007 >006 FERRI_P      25 PROGRAM    Y >082 024
008 >00f GATOR_P      25 PROGRAM    Y >15a 014 >010 010
009 >009 LEDZEP_P     25 PROGRAM    Y >0ca 024
010 >00c M-EARTH_P    25 PROGRAM    Y >112 024
011 >00a MAC_P        25 PROGRAM    Y >0e2 024
012 >00d PAGODA_P     25 PROGRAM    Y >12a 024
013 >007 PIRATE_P     25 PROGRAM    Y >09a 024
014 >00e ROCKY-H_P    25 PROGRAM    Y >142 024
015 >008 THE-WALL_P   25 PROGRAM    Y >0b2 024
```

Disk 63. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public domain software disk

RLE Pictures

This disk contains a collection of graphic pictures stored in TI-Artist format. They may be loaded and printed with the MAXRLE program by Travis Watford, available from the BCS library. They may also be loaded, edited, and printed using TI-Artist, a commercial program. These pictures were provided by Barry Traver via a special interest group on CompuServe.

Disk 63. Contents of files (graphics)

ABCNEW_P

BAMBI_P

COKE_P

CORVET_P

DROP_P

FERRI_P

GATOR_P

LEDZEP_P

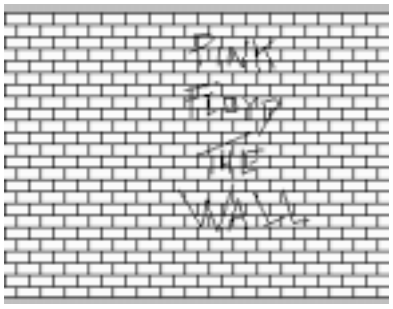
M-EARTH_P

MAC_P

PAGODA_P

PIRATE_P

ROCKY-H



THE-WALL_P

Disk 64. RLE Pictures #3, Space and Science Fiction

Version: *Author*:
Requires: *Language*: *Updated*: 08/15/86

Includes the Enterprise, space shuttle, Star Wars, Dr. Who, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = RLE
Sectors total           = 360
Sectors used            = 353
Sectors available       = 5
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01a -README       3 DIS/VAR 80   >01b 002
002 >002 ASTRO_P      25 PROGRAM      >022 024
003 >00c DALEK_P      25 PROGRAM    Y >112 024
004 >003 ENTERP_P     25 PROGRAM    Y >03a 024
005 >00e EVILF_P      25 PROGRAM    Y >142 024
006 >00f SEER_P       25 PROGRAM    Y >15a 014 >010 010
007 >004 SHUTTLE1_P   25 PROGRAM    Y >052 024
008 >005 SHUTTLE2_P   25 PROGRAM    Y >06a 024
009 >006 STAR1_P      25 PROGRAM    Y >082 024
010 >007 STAR2_P      25 PROGRAM    Y >09a 024
011 >008 STAR3_P      25 PROGRAM    Y >0b2 024
012 >009 STAR4_P      25 PROGRAM    Y >0ca 024
013 >00a STWAR_P      25 PROGRAM    Y >0e2 024
014 >00d UHURA_P      25 PROGRAM    Y >12a 024
015 >00b WOOKIE_P     25 PROGRAM    Y >0fa 024
```

Disk 64. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

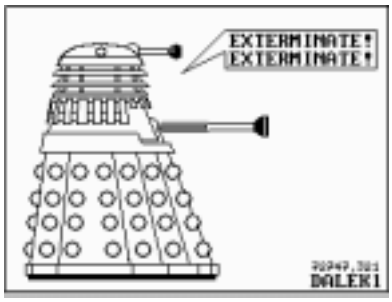
Public domain software disk

RLE Pictures

This disk contains a collection of graphic pictures stored in TI-Artist format. They may be loaded and printed with the MAXRLE program by Travis Watford, available from the BCS library. They may also be loaded, edited, and printed using TI-Artist, a commercial program. These pictures were provided by Barry Traver via a special interest group on CompuServe.

Disk 64. Contents of files (graphics)

ASTRO_P

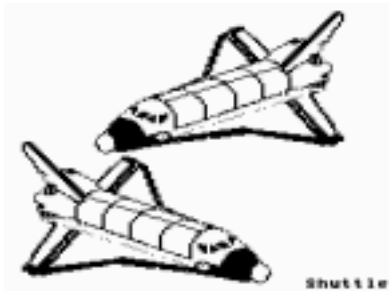


DALEK_P

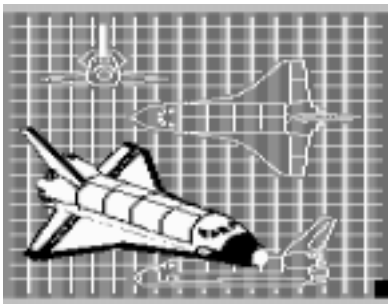
ENTERP_P

EVILF_P

SEER_P

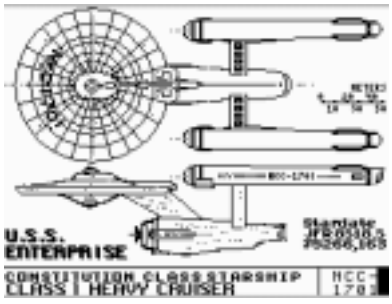


SHUTTLE1_P



SHUTTLE2_P

STAR1_P



STAR2_P

STAR3_P

STAR4_P

STWAR_P

UHURA_P

WOOKIE_P

Disk 65. RLE Pictures #4, Famous Folks

Version: *Author*:
Requires: *Language*: *Updated*: 08/15/86

Includes Bugs Bunny, Dracula, Wait Disney, rock stars, and more.

```
dskdir. v2.0. 12-dec-96
Disk name               = RLE
Sectors total           = 360
Sectors used            = 353
Sectors available       = 5
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01a -README       3 DIS/VAR 80   >01b 002
002 >00d ABBOTT+C_P   25 PROGRAM    Y >12a 024
003 >002 BUGSBUN_P    25 PROGRAM    Y >022 024
004 >003 CHRISTI_P    25 PROGRAM    Y >03a 024
005 >004 CINDI_P      25 PROGRAM    Y >052 024
006 >00f DISNEY_P     25 PROGRAM    Y >15a 014 >010 010
007 >005 DR-WHO_P     25 PROGRAM    Y >06a 024
008 >006 DRACULA_P    25 PROGRAM    Y >082 024
009 >007 GOSTBUST_P   25 PROGRAM    Y >09a 024
010 >008 HOWDY_P      25 PROGRAM    Y >0b2 024
011 >009 INDIAN_P     25 PROGRAM    Y >0ca 024
012 >00a MERLIN_P     25 PROGRAM    Y >0e2 024
013 >00b RODNEY_P     25 PROGRAM    Y >0fa 024
014 >00c SGTMAJ_P     25 PROGRAM    Y >112 024
015 >00e TINA_P       25 PROGRAM    Y >142 024
```

Disk 65. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public domain software disk

RLE Pictures

This disk contains a collection of graphic pictures stored in TI-Artist format. They may be loaded and printed with the MAXRLE program by Travis Watford, available from the BCS library. They may also be loaded, edited, and printed using TI-Artist, a commercial program. These pictures were provided by Barry Traver via a special interest group on CompuServe.

Disk 65. Contents of files (graphics)

ABBOTT+C_P
(actually Laurel and Hardy — Ed.)

BUGSBUN_P

CHRISTI_P

CINDI_P

DISNEY_P

DR-WHO_P

DRACULA_P

GOSTBUST_P

HOWDY_P

INDIAN_P

MERLIN_P

RODNEY_P

SGTMAJ_P

TINA_P

Disk 66. Music Disk 3

Version: *Author*: Terrence Murphy, others
Requires: *Language*: *Updated*: 09/13/86

Another in the collection of fantastic music for the TI. Includes 12th Street Rag, Axel F, Harrigan, Flag Day and July 4th by Terrence Murphy, and one of the best music tutorials around.

```
dskdir. v2.0. 12-dec-96
Disk name               = MUSIC-3
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01f -README       5 DIS/VAR 80   >0a1 002 >00b 001 >020 001
002 >00d 12/ST/RAG    89 INT/VAR254 Y >0d4 088
003 >003 AXELF/NEW    56 INT/VAR254 Y >024 055
004 >00e BOJANGLES    43 PROGRAM    Y >12c 042
005 >004 FLAG/DAY      2 PROGRAM    Y >05b 001
006 >002 FLAGLDR       9 PROGRAM    Y >022 002 >05c 005 >005 001
007 >006 FLAGLPD      23 PROGRAM    Y >061 022
008 >00f HARRIGAN     34 PROGRAM    Y >156 018 >010 015
009 >007 JULY4        33 PROGRAM    Y >077 032
010 >008 JULY4LDR      5 DIS/FIX 80 Y >097 004
011 >009 JULY4LNK      2 PROGRAM    Y >09b 001
012 >00a JULY5         6 PROGRAM    Y >09c 005
013 >00c TEACH/MUS    50 INT/VAR254 Y >0a3 049
```

Disk 66. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public domain software disk

EXTENDED BASIC PROGRAMS

12/ST/RAG
A very complete and complex version of Joplin's Twelfth Street Rag (by Eduay L. Bowman — Ed).

AXELF
One of the ultimate sound programs for the TI. Yes I know there is a copyright notice but the author himself uploaded this to CompuServe for the TI Community.

BOJANGLES
An animated version of this sixties tune.

HARRIGAN
Another animated tune. Lots of fun.

JULY4LNK
FLAG/DAY
Two holiday tunes with words and graphics by the prolific Terrence Murphy. Written in 9900 Basic. Loads from Extended BASIC. (Fix line 110 in JULY4LNK to point to DSK1.JULY4LDR — Ed).

TEACH/MUS
One of the nicest introductions to creating music and sound effects on the TI.

Disk 67. Music Disk 4

Version:
Requires: XB
Author: Ken Gilliland, others
Language:
Updated: 08/15/86

Two fantastically long selections from Wagner Opera by Ken Gilliland complete with colorful graphics. The pinnacle of musical achievement on the TI. Also thrown in is a unique Scottish Air.

```
dskdir. v2.0. 12-dec-96
Disk name               = OPERA
Sectors total           = 360
Sectors used            = 352
Sectors available       = 6
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >005 -README       4 DIS/VAR 80   >0cd 003
002 >009 6-RIBBONS    29 PROGRAM    Y >161 007 >00a 021
003 >003 SIEG-ART     39 PROGRAM    Y >046 038
004 >002 SIEG-LOAD    37 PROGRAM      >022 036
005 >004 SIEG-WORK    98 INT/VAR254 Y >06c 097
006 >006 TRIST        46 PROGRAM    Y >0d0 045
007 >008 TRIST2       99 INT/VAR254 Y >0ff 098
```

Disk 67. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public domain software disk

This disk contains three different musical works. The first two are programmed by Ken Gilliland. All programs may be run in Extended BASIC. The final program is Six Ribbons, which doesn't fit with the two operatic works, but there was some space left!

SIEG-LOAD
SIEG-ART
SIEG-WORK
These are the three files that make up the award winning Siegfried program.

TRISTAN
TRISTAN2
These two programs make up the impressive music and graphics experience taken from the Wagner opera.

6-RIBBONS
This is just a nifty little Scottish Aire that makes excellent use of the TI's sound capabilities.

Disk 68. C Programs

Version: *Author*:
Requires: *Language*: c99 *Updated*: 08/15/86

Programs written in c99 collected by Clint Pulley. Includes an archiver, C-Invaders, Tic-Tac-Toe, graphics demos and more. C source code to most programs is on disk.

```
dskdir. v2.0. 12-dec-96
Disk name               = CLINT
Sectors total           = 360
Sectors used            = 347
Sectors available       = 11
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00d -README       4 DIS/VAR 80   >0f9 002 >0fd 001
002 >002 AR1          33 PROGRAM    Y >022 032
003 >003 AR2          12 PROGRAM    Y >042 011
004 >004 AR;C         43 DIS/VAR 80 Y >04d 042
005 >005 AR;DOC       20 DIS/VAR 80 Y >077 019
006 >00f BITGRF;ARC   91 DIS/VAR 80 Y >108 090
007 >00c BOXDEM/C      7 DIS/VAR 80 Y >0f3 006
008 >009 BOXDEMO/C    10 DIS/VAR 80 Y >0ba 007 >0fb 002
009 >021 BOXES        14 DIS/FIX 80 Y >014 013
010 >006 CINVADER     33 PROGRAM    Y >08a 032
011 >007 CINVADES      4 PROGRAM    Y >0aa 003
012 >013 EXPLST;C     10 DIS/VAR 80 Y >162 006 >010 003
013 >008 GRFTST;C     14 DIS/VAR 80 Y >0ad 013
014 >00a TTT          30 DIS/FIX 80 Y >0c1 029
015 >00b TTT;C        22 DIS/VAR 80 Y >0de 021
```

Disk 68. Contents of file -README

This is a selection of c material received from Clint Pulley.

1) ar1, ar2 are program, ar;c is c.code, and ar;doc is doc for c archiver program

2) boxdem/c, boxdemo/c are c.code for box demos; load cfio, boxes, printf when you "load + run"

3) ttt is o.code and ttt;c c.code for 3-d tic-tac-toe

4) cinvader is game in prog. format

5) explist;c is expanded list program

6) grftst;c is demo of all grf1 functions

N.B. you cannot run mem.image format of c programs using FNLWRTR; must use E/A 5.

dlm (Donald L. Mahler — Ed).

Disk 68. Contents of file ARC;C

```
/*
**   AR -- c99 Text File Archiver VERSION 1.2
**
**   Ar collects text files into a single archive file.
**   Files can be extracted, added, replaced or deleted.
**   A list of the archive contents can be generated.
**
**   Processing modes :
**
**   d  delete named files from the library.
**   p  print named, or all, files on stdout.
**   t  table of contents of named, or all, files to stdout.
**   u  update the archive by adding/replacing files
**       (used to create a new library).
**   x  extract named, or all, files.
**
**   Clear (function 4) terminates execution.
**
**   This program was given as a class assignment in the
**   Computer Science Department at the University of Arizona.
**   It was contributed by Ernest Payne.
**
**   c99 version adapted by Clint Pulley
**
**   Last edit  86/07/17  1850
*/
#include "dsk1.stdio"
#asm
 REF PRINTF,FPRINT
#endasm
#define NAMESIZE 25
#define MAXLINE  81
#define MAXFILES 20
#define HDR      ">>>"
char date[20];
char aname[NAMESIZE];
char wname[NAMESIZE];
char tname[]="dskn.wrk$";
char devname[]="dskn.";
char line[MAXLINE];
char fnbuf[200];     /* MAXFILES*10 */
int fnptr[MAXFILES];
int fstat[MAXFILES];
int nfiles;
int errchk;

main()
{ int cmd,i,first;
  *date=*aname=0;
```

```
  tname[3]=' ';
  first=YES;
  while(1)
  { for (i=0 ; i<MAXFILES ; ++i) fstat[i]=0;
    nfiles=errchk=0;
    printf("\fc99 File Archiver v1.2\n\nDate : %s",date);
    printf("\nArchive Filename : %s\n",aname);
    printf("Work File Drive # : %c\n\n",tname[3]);
    if(first)
    { locate(3,8); gets(date);
      locate(4,20); gets(aname);
      if((tname[3]=getdn(5,21))==0)exit(0);
      first=NO;
    }
    puts("\n\nSelect mode :\n\nd Delete files\np Print files\nq Quit\n");
    puts("t Table of contents\nu Update/create archive\nx Extract files");
    puts("\n\nYour choice (dpqtux) : ");
    while(1)
    { cmd='P';
      switch(getchar() | 32)
      {
    case 'd': drop(aname);
              break;
    case 't': table(aname);
              break;
    case 'u': update(aname);
              break;
    case 'x': cmd='X';
    case 'p': extract(aname, cmd);
              break;
    case 'q': putchar('\n'); exit(0);
    default: putchar('\b \b'); continue;
      }
      break;
    }
    puts("\nPress any key to continue ");
    getchar();
  }
}
/* acopy - copy size characters from fpi to fpo
*/
acopy(fpi,fpo,size) int fpi, fpo,size;
{ int c;
  while(size-- > 0)
  { poll(YES);
    if((c = getc(fpi)) == EOF)
      break;
    putc(c,fpo);
  }
}
```

```
/* addfile - add file "name" to archive
*/
addfile(name,fp) char *name; int fp;
{ int nfp;
  if((nfp = fopen(bldfn(name),"r")) == NULL)
  { printf("%s : can't open\n",wname);
    errchk = 1;
  }
  if (errchk == 0)
  { fprintf(fp,"%s %-10s %6d %s\n",HDR,name,fsize(nfp),date);
    fcopy(nfp,fp);
    fclose(nfp);
    printf( " copied new %s\n", name);
  }
}
/* amove - rewrite archive from workfile
*/
amove()
{ int file1,file2;
  if(errchk)
  { printf("fatal errors - archive not altered\n");
    fdelete(tname);
    exit(1);
  }
  file1=mustopen(tname,"r");
  file2=mustopen(aname,"w");
  puts("\n rewriting archive file\n");
  if(!fcopy(file1,file2))
  { printf("Error on archive file rewrite\n");
    printf("Updated archive is on%s\n",tname);
    exit(1);
  }
  fclose(file2);
  fdelete(tname);
}
/* atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
/* cant - print file name and die
*/
cant(name) char *name;
{ printf("%s : can't open\n",name);
```

```
  exit(1);
}
/* bldfn - build filename string
*/
bldfn(name) char *name;
{ strcpy(wname,devname);
  strcat(wname,name);
  return wname;
}
/* drop - delete files from archive
*/
drop(aname) char *aname;
{ int afp, tfp;
  getfns();
  if(nfiles <= 0) /* protect innocents  */
    error("delete by name only");
  afp = mustopen(aname,"r");
  tfp = mustopen(tname,"w");
  replace(afp,tfp,'d');
  notfound();
  fclose(afp);
  fclose(tfp);
  amove();
}
/* error - print message and die
*/
error(msg) char *msg;
{ printf("\nError - %s\n",msg);
  exit(1);
}
/* extract - extract files from archive
*/
extract(aname,cmd) char *aname; int cmd;
{ int afp, efp;
  char ename[NAMESIZE], in[MAXLINE];
  int size;
  putchar('\f');
  afp = mustopen(aname,"r");
  if(cmd == 'P') efp = getf("List","w");
  getfns();
  while((size = gethdr(afp,in,ename)) >= 0)
  if(!fmatch(ename, YES)) fskip(afp,size);
  else
  { if(cmd != 'P')
    { efp = fopen(bldfn(ename),"w");
      if(efp == NULL)
      { printf("%s : can't create\n",wname);
        errchk = 1;
        fskip(afp,size);
      }
```

```
    }
    acopy(afp,efp,size);
    if(cmd == 'P')
      printf( "printed %s\n", ename);
    else
    { printf( "created %s\n", ename);
      fclose(efp);
    }
  }
  notfound();
  fclose(afp);
}
/* fcopy - copy file in to file out
*/
fcopy(in,out) int in, out;
{ while(fgets(line,MAXLINE,in))
  { poll(YES);
    if(!fputs(line,out))return 0;
  }
  return(1);
}
/* fmatch - check if name matches argument list
*/
fmatch(name, quit) char *name; int quit;
{ int i, done;
  char *fnp;
  if(nfiles <= 0) return(1);
  done = YES;
  for(i=0;i<nfiles;++i)
  { fnp = fnptr[i];
    if(strcmp(name,fnp) == 0)
    { fstat[i] = 1;
      strcpy(fnp,name);
      return(1);
    }
    if(fstat[i] == 0) done = NO;
  }
  if(quit & done) exit(0);
  return(0);
}
/* fsize - size of file in characters
*/
fsize(fp) int fp;
{ int i,j;
  i=0;
  do
  { if((j=fread(line,MAXLINE,fp))==ERR)break;
    i=i+j+1;
    poll(YES);
  }
  while(!feof(fp));
```

```
  rewind(fp);
  return(i);
}
/* fskip - skip n characters on file fp
*/
fskip(fp,n) int fp, n;
{ int i;
  do
  { if((i=fread(line,MAXLINE,fp))==ERR)break;
    n=n-i-1;
  }
  while(n>0);
}
/* getdn - get a drive # */
getdn(r,c) int r,c;
{ while(1)
  { locate(r,c);
    gets(line);
    if(*line>='1' & *line<='4')break;
    if(!*line) return 0;
    locate(r,c+3); puts("Error - retry");
  }
  locate(r,c+1); puts("               ");
  return(*line);
}
/* getf - returns unit # or 0 if null name entered
*/
getf(text,m) char *text,*m;
{ int unit; char fn[NAMESIZE];
  unit=0;
  while(1)
  { puts(text);
    puts(" filename? ");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
/* getfns - get file names into fname, check for duplicates
*/
getfns()
{ int i, j; char *ptr;
  nfiles = 0;
  ptr=fnbuf;
  while(1)
  { puts("\fFilename Entry:\nDrive #  ");
    if(devname[3]=getdn(2,8))break;
  }
```

```
  puts("\nFiles: (null entry to end)\n");
  while(1)
  { puts(devname);
    gets(ptr);
    if(!*ptr)break;
    fnptr[nfiles]=ptr;
    while(*ptr=toupper(*ptr++));
    if(++nfiles > MAXFILES)
    error("too many file names");
  }
  for(i = 0; i < nfiles-1; ++i)
    for(j = i+1; j < nfiles; ++j)
    { if(strcmp(fnptr[i],fnptr[j]) == 0)
      { printf("%s : duplicate file names\n",fnptr[i]);
        exit(1);
      }
    }
  puts("\fProcessing...\n");
}
/* gethdr - get header info from fp
*/
gethdr(fp,buf,name) int fp; char *buf, *name;
{ if(fgets(buf,MAXLINE,fp) == NULL)
    return(-1);
  buf = getwrd(buf,name);
  if(strcmp(name,HDR) != 0)
    error("archive not in proper format");
  buf = getwrd(buf,name);
  return(atoi(buf));
}
/* getwrd - copy first word of s to t
*/
getwrd(s,t) char *s, *t;
{ while(isspace(*s)) ++s;
  while(*s != '\0' & !isspace(*s))  *t++ = *s++;
  *t = '\0';
  return(s);
}
/* isspace - return true if space
*/
isspace(s) int s;
{ return(s==' ');
}
/* mustopen - open file or die
*/
mustopen(name,mode) char *name, *mode;
{ int fp;
  if(fp = fopen(name,mode)) return(fp);
  cant(name);
}
/* notfound - print "not found" message
```

```
*/
notfound()
{ int i;
  for(i=0;i<nfiles;++i)
    if(fstat[i] == 0)
    { printf("%s not in archive\n",fnptr[i]);
      errchk = 1;
    }
}
/* replace - replace or delete files
*/
replace(afp,tfp,cmd) int afp, tfp; char cmd;
{ char in[MAXLINE], uname[NAMESIZE];
  int size;
  while((size = gethdr(afp,in,uname)) >= 0)
  { if(fmatch(uname, NO))
    { if(cmd == 'u')    /* add new one */
        addfile(uname,tfp);
      fskip(afp,size);  /* discard old one */
      printf( "dropped old %s\n", uname);
    }
    else
    { fputs(in,tfp);
      acopy(afp,tfp,size);
    }
  }
}
/* strcat - concatenate t to end of s
*/
strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
}
/* strcmp - return <0,   0,  >0 according to
**          s<t, s=t, s>t
*/
strcmp(s, t) char *s, *t;
{ while(*s == *t)
  { if(*s == 0) return (0);
    ++s; ++t;
  }
  return (*s - *t);
}
/* strcpy - copy t to s
*/
strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
/* table - print table of archive contents
```

```
*/
table(aname) char *aname;
{ char in[MAXLINE], lname[NAMESIZE];
  int afp, fn, size;
  putchar('\f');
  afp = mustopen(aname,"r");
  fn=getf("List","w");
  putchar('\n');
  if(fn)
  { fprintf(fn,"\n Contents of Archive File :\n %-18s%s\n",aname,date);
    fprintf(fn,"\n Filename     Size   Date",date);
    fprintf(fn,"\n --------     ----   ----\n");
  }
  while((size = gethdr(afp,in,lname)) >= 0)
  { poll(YES);
    fprintf(fn,"%s\n",in+3);
    fskip(afp,size);
  }
  if(fn)fclose(fn);
  notfound();
  fclose(afp);
}
/* toupper - return upper-case if c lower case
*/
toupper(c) int c;
{ if((c>='a') & (c<='z')) return(c-32);
  return c;
}
/* update - update existing files, add new ones at end
*/
update(aname) char *aname;
{ int afp, i, tfp;
  char fn[NAMESIZE];
  afp = mustopen(aname,"u");
  tfp = mustopen(tname,"u");
  getfns();
  replace(afp,tfp,'u');   /* update existing */
  if(nfiles > 0)
  { for(i=0;i<nfiles;++i) /* add new ones */
      if(fstat[i] == 0)
      { addfile(fnptr[i],tfp);
        if(errchk) break;
        fstat[i] = 1;
      }
  }
  fclose(afp);
  fclose(tfp);
  amove();
}
```

Disk 68. Contents of file AR;DOC

Documentation file for AR;C 86/07/22

Ar collects text files into a single archive file. Files can be extracted, added, replaced or deleted. A list of the archive contents can be generated.

Processing modes:

d delete named files from the archive.
p print named, or all, files.
t table of contents of all files.
u update the archive by adding/replacing files (used to create a new archive).
x extract named, or all, files.

CLEAR (FCTN 4) terminates execution.

The original public domain Z80 small-c version of this program was given as a class assignment in the Computer Science Department at the University of Arizona.

The c99 version was adapted and extensively modified by Clint Pulley. As it is based on a public domain program, this version is also considered to be in the public domain.

AR is especially useful for maintaining collections of small text files. As input files are combined into a single archive file, considerable savings in disk space can result. The archive file is in display/variable 80 format with a header record for each archived file. The header contains four fields:

-a header flag (>>>)
-the filename without device specifier
-the number of bytes (including the c99 newline) in the file
-the date that was entered when AR was run

Since file size is specified by a byte count, it is essential that an archive file is NOT edited except by use of AR or a damaged archive will result. Files within an archive should not exceed 32,767 bytes in length.

Program Operation

This program is usually distributed as program image files named AR1 and AR2. It is loaded in the usual manner with E/A option 5.

After being loaded, AR will prompt for three parameters which will be used for all processing in this run of the program. These parameters are:

> The current date, which is placed in the header of all files added to, or replaced in the archive. It is suggested that the international date representation yy/mm/dd be used.
>
> The complete filename (dskn.fffff) of the new or existing archive file.
>
> The number of the drive where the temporary working copy of the archive will be written. This is normally the drive containing the permanent archive file, but users with multiple drives can speed up processing by using a different drive.

After entry of these parameters and after each operation the processing mode menu is displayed. Pressing a single key will result in execution of the selected mode.

Filename Entry

The modes d, p, u and x require a list of the filenames to be deleted, added, or extracted. This is accomplished by a special input screen designed to minimize the keystrokes required. The filename entry will first prompt for a drive number. This is the disk drive where the files to be archived are located or where the extracted files will be written. For the d and p modes this parameter is ignored. Next, the program will prompt for filenames by displaying "DSKn.". After all required names have been entered, the list is terminated by pressing <enter> immediately after the prompt. If p or x mode has been selected and <enter> is pressed in response to the very first filename prompt, then ALL files in the archive will be printed or extracted.

Output Filename

The p and t modes also prompt for an output filename, which will normally be a printer. Pressing <enter> in response to this prompt will result in output being displayed on the screen. If a disk filename is entered when using p mode, a single file containing all the selected files (without headers) will be generated.

Processing

Once all required input has been entered, processing begins. Information concerning the operations being performed is displayed on the screen. For all modes which change the contents of the archive, the final operation is the rewriting of the archive file, after which the working file is deleted.

A Request

This program is being distributed in source form. Make any changes to it that you wish, but PLEASE do not change the format of the archive file. Because of their compactness, archive files are a useful way of conveying collections of text files between TI owners, but this is only true if the file format is consistent.

Disk 68. Contents of file BITGRF;ARC

BITDOC 11958 86/07/18

PRELIMINARY NOTE: Earlier this year I uploaded initial versions of BITRTN and BITWRT. These had been based on some experimentation I had done while working with other programs. Finally I have gotten around to going over the material, improving documentation and making it easier to use. Please feel free to send me your comments.

Jay Holovacs 7/1/86

BITMAP GRAPHICS COMMANDS FOR C99 COMPILER

BITRTN REL2.0
BITWRT REL2.0
6/26/86

This material is written by Jay Holovacs as a support for Clint Pulley's C compiler. I believe the commands are compatible with various versions of his compiler.

The material may be used personally or in your programs to be distributed.

REVISIONS IN REL 2.0

Rel 2.0 of BITRTN and BITWRT contains several additional features:

1) A bug is corrected in bitmap() which interfered with the console KSCAN routine.

2) A circle function has been added.

3) The graphics primitives now have a toggle argument to provide an 'erase' function.

4) Console IO functions (comparable to gets, getchar etc.) have been added to allow easy writing of user interactive graphics and game programs.

5) Improved documentation with examples. At the end of this documentation is a short c program which illustrates use of various graphics commands and will enable you to interactively experiment.

The bitmap graphics extensions to c are divided into two "#include" files. The first, BITRTN, contains the tools for basic graphics manipulation. The second, BITWRT contains additional commands to mix text with bit mapped graphics. You can use these as they are or combine them into a library module (see below).

Once included in your program, the commands can be called as any c function.

VALID RANGES (similar to BASIC):

color: 1 to 16
row: 1-24
column: 1-32
ascii: 32-127
x (bitmap location): 1-256
y (bitmap location): 1-192
str: string pointer; string literal; base pointer of character array
toggle=0-->erase the primitive
toggle!=0-->draw the primitive

NOTE1: Calling plot() with out of bounds coordinates (off the screen) will result in the dot not being printed.

Calling circle() with coordinates such that all or part of the circle will be off the screen will result in the part of the circle visible being drawn.

Calling line() with invalid coordinates will result in an aborted command, no line or part of line will be drawn. Since rect() is based on calls to line, any lines of the rectangle which are partially or completely off screen will not be drawn.

It is highly recommended that your program monitor passed values for out of bounds coordinates when calling the line or rect functions.

NOTE2: Internal labels in the assembly language portions all include '$' to avoid conflict with normally named global variables in your program.

INCLUDED IN THE BITRTN FILE:

The toggle function: In the previous version the only way to get rid of a line etc was to draw it invisible or in background color. This left 'ghosts' which could reappear if that block of 8 pixels was color changed by another graphics or text item passing through it. Calling a function with toggle=0 will erase the primitive, calling with any other value in toggle will draw the primitive. Even when you are erasing, you must still pass a foreground color. This may seem irrelevant, however if other pixels in the set of 8 are turned on (from another graphics primitive) they will be displayed in whatever color is last specified when that block is accessed, so choice of color can be significant in some cases.

`bitmap(fore,back)`
: Must be called first to change the screen configuration from text mode (used by c) to bitmapped mode. The arguments are respectively the fore and background colors selected for the screen default.

`bitclr()`
: Clears the entire screen, but does not change any color defaults. If you wish to modify screen color defaults, the bitmap() call can be repeated.

`plot(x,y,color,toggle)`
: Will turn 'on' a single pixel at the screen location indicated.

`line(x1,y1,x2,y2,color,toggle)`
: Draws a line of the specified color between the points indicated.

`rect(x1,y1,x2,y2,color,toggle)`
: Draws a rectangle with opposite corners at the points indicated.

`circle(xcenter,ycenter,radius,color,toggle)`
: Draws a circle with the appropriate parameters. This version incorporates aspect ratio correction to produce a round shape even though the pixels on a monitor or TV screen are not equally spaced horizontally and vertically. There are a few disadvantages to aspect correction however (routine is slowed down, screen dumps to printer may not produce proper shapes and the fact that the sides are 'pulled in' by the correction means that pixel coordinates you might expect on the circle will miss it, for instance if a line is to meet the circle as a tangent). If desired the aspect correction can be altered (to better reflect your system characteristics) or defeated (see comments in the source code).

INCLUDED IN THE BITWRT FILE:

`bitxt()`
Call this function once, before calling bitmap(); this function copies all ascii character definitions into a buffer in CPU ram. Do not call it again even if you call bitmap several times to reinitialize the screen.

`bputch(ascii,row,column,color)`
Places the character at location indicated; similar to putchar().

`bputs(row,column,color,str)`
Places text on the screen in 24x32 format (works similar to 'puts()' in normal c).

`blanks(row,column)`
Places a blank on the screen erasing material below unlike bputch which will allow graphics or text underneath to show thru.

`btblanks(row,col,count)`
Blanks out a sequence of locations starting at row,col.

`bgetch(row,column,color)`
Waits for user input and returns the ascii of the key pressed. If printable, it is displayed on the screen, however any code (including break) will be returned regardless of whether it is echoed. The returned characters can be monitored by your program for a break or other special character; but the function will not exit() or abort() automatically.

`bgets(buffadr,size,row,column,color)`
Inserts printable characters in buffer specified. Unlike normal gets(), you must specify a maximum size. This prevents any chance of an overly verbose user overwriting the buffer at runtime.

`getky()`
Scans the keyboard once and simply returns an ASCII if a new key is pressed or else returns 0. This is similar to poll() in CSUP, however I have had difficulty getting poll to return an ASCII (on my console, at least) so this alternate is accessible for your use as well. Unlike poll it does not automatically exit on 'break'.

`nxtcur(&row,&column)` and `bkspc(&row,&column)`
Are used by bgets() etc. for incrementing and decrementing the next print position in an orderly manner. The BITWRT philosophy consistently passes screen locations at each call rather than an invisible system cursor (as is normally done for C text applications) on the assumption that screen printing locations will be widely varied on graphic applications. You can utilize nxtcur and bkspc to update row and column variables. Because these functions modify values pointed to by passed parameters, if you fail to use the 'address-of' operator '&' strange results will occur.

Creating your own BITSUP library module: Source code is provided here for reference and user modification, but it is generally more convenient to actually work with a library module. To generate library module called BITSUP, compile and assemble the following:

```
#asm

  DEF BITMAP,BITCLR,PLOT,LINE,RECT,CIRCLE
  DEF BITXT,BPUTCH,BPUTS,BGETCH,BGETS,GETKY,BTBLNK,NXTCUR
  DEF BLANKS,BKSPC

#endasm

#include DSK1.BITRTN
#include DSK1.BITWRT
```

To access these routines from your program, write this short file called BITACC:

```
#asm

  REF BITMAP,BITCLR,PLOT,LINE,RECT,CIRCLE
  REF BITXT,BPUTCH,BPUTS,BGETCH,BGETS,GETKY,BTBLNK,NXTCUR
  REF BLANKS,BKSPC

#endasm
```

Note that there is at least one space in front of each occurrence of "REF" or "DEF".

This would be then used in the form:

```
#include DSK1.BITACC
.
.
/* your program ....*/
```

The compiled and assembled BITSUP library would be loaded at runtime (just like CSUP).

FINAL NOTES:

SUPPORT: This is a spare time project on my part, and can only receive sporadic attention, however, I will try to address any bugs, problems or suggestions. If you are writing concerning a problem, be sure to send a copy of the portion of your code that seems to be giving trouble and explain any circumstances surrounding it. Address any comments/questions/problems via CompuServe Easyplex or paper mail. (I cannot guarantee to solve problems, but am interested in weeding out any remaining bugs and adding new features).

Assembly code in the files has been stripped of comments and R-register references to make it more compact and compatible with the same assembler options as C99. If you are an assembly hacker seriously interested in commented code, contact me as noted above.

Jay Holovacs
95 King George Rd.
Warren, NJ 07060

CIS 74756,413

The following program, SKETCH, is a simple implementation of the graphics commands. It will allow experimentation with the commands interactively to get a feel for their application. (To keep the purpose obvious, the code is a bit verbose for a c program. The local variables xcent, ycent etc could have been eliminated entirely by simply placing the calls to bgetint() as the arguments in circle() etc.)

NOTE: SKETCH assumes you have prepared the BITSUP and BITACC modules as above.

```
/* demonstration sketch program to illustrate BITMAP routines */
/* Jay Holovacs
   95 King George Rd.
   Warren, NJ 07060

   cis 74756,413

 Commands:
   L)ine enter starting and ending coordinates
   C)ircle enter coord of center and radius
   E)rase clear screen
   P)lot a point enter x,y coordinates
   R)ectangle enter corner points  */

int pts[4];
#include DSK1.BITACC
#include DSK1.CONV;C
main()
{ int q,chce,xcent,ycent,rad; q=0;
  bitxt();
  bitmap(16,2);
```

```
while(1)
{
  blanks(23,5,25);
  bputs(23,5,16,"Command");
  chce=bgetch(23,13,16);
  switch (chce)
  {
case 'L':
    q=0;
    do
    {
      blanks(23,5,15);
      bputs(23,5,16,"Line coord: x1,y1,x2,y2");
      pts[q]=bgetint(24,5,16);
    }
    while (q++<3);
    line(pts[0],pts[1],pts[2],pts[3],16,1);
    break;
case 'C':
    blanks(23,5,25);
    bputs(23,5,15,"Center coordinates");
    bputs(23,23,15,"X");
    xcent=bgetint(24,5,16);
    blanks(23,23,4);
    bputs(23,23,15,"Y");
    ycent=bgetint(24,5,16);
    blanks(23,5,25);
    bputs(23,5,15,"Radius");
    rad=bgetint(24,5,16);
    circle(xcent,ycent,rad,15,1);
    break;
case 'E':
    bitclr();
    break;
case 'P':
    blanks(23,5,25);
    bputs(23,5,15,"Point Coordinates x,y");
    xcent=bgetint(24,5,16);
    ycent=bgetint(24,5,16);
    plot(xcent,ycent,16,1);
    break;
case 'R':
    q=0;
    do
    {
      blanks(23,5,15);
      bputs(23,5,16,"Rect coord: x1,y1,x2,y2");
      pts[q]=bgetint(24,5,16);
    }
    while (q++<3);
```

```
      rect(pts[0],pts[1],pts[2],pts[3],16,1);

    }
  }

}

bgetint(row,col,color)
int row,col,color;
{ char bufr[6];
  bgets(bufr,6,row,col,color);
  return(atoi(bufr));
}

>>> BITRTN       5040 86/07/18
/* BITR;CS CONTAINS FULL COMMENTS, USE BITRTN
 FOR INCLUDE PURPOSES */
/* TEST  REL VERSION 2.0b 6/28/86
   Jay Holovacs
   95 King George Rd.
   Warren, NJ 07060
   CIS: 74756,413

   For Use With C99C Compiler by Clint Pulley */


/* Contents:

   BITMAP V1.2
   LINE V3.0A
   PLOT V3.0A
   RECT V2.0A
   CIRCLE V2.1 */


 bitmap(fore,back)
 /* V1.2 6/21/86 JAY HOLOVACS
  MUST BE CALLED BEFORE ANY BITMAP COMMANDS
 ARE INVOKED
 * * USES TI BASIC COLOR DEFINITIONS */

  int fore,back;
  #asm
 REF VWTR,VSBW,VMBW,VSBR
 MOV @4(14),0
 DEC 0
 SLA 0,4
 A @2(14),0
 DEC 0
 SWPB 0
 BLWP @BITMP$
```

```
 B @B$MP2
BITMP$  DATA PLW$P,B$MP1 USES PLOT WORKSPACE
B$MP1  MOV *13,2
 LIMI 0
 LI 0,>0002
 BLWP @VWTR
 LI 0,>E001
 MOVB 0,@>83D4
 SWPB 0
 BLWP @VWTR
 LI 0,>0206
 BLWP @VWTR
 LI 0,>03FF
 BLWP @VWTR
 LI 0,>0403
 BLWP @VWTR
 LI 0,>0570
 BLWP @VWTR
 LI 0,>0607
 BLWP @VWTR
 LI 0,>3800
 LI 1,>D000
 BLWP @VSBW
 MOV 2,0
 SWPB 0
 ANDI 0,>000F
 ORI 0,>0700
 BLWP @VWTR
 LI 0,>1800
 CLR 1
SILP$  BLWP @VSBW
 INC 0
 AI 1,>0100
 CI 0,>1B00
 JNE SILP$
 BLWP @B$CL1
 LI 0,>2000
 MOV 2,1
CLP$  BLWP @VSBW
 INC 0
 CI 0,>3800
 JNE CLP$
 RTWP
B$CL1  DATA B$CL2,B$CL3
B$CL2  BSS 32
B$CL3  LIMI 0
 CLR 0
 CLR 1
CLP2$  BLWP @VSBW
 INC 0
```

```
 CI 0,>1800
 JNE CLP2$
 RTWP
B$MP2
 #endasm


bitclr()
/* V1.0B BY JAY HOLOVACS CLEARS THE SCREEN */

 #asm
 BLWP @B$CL1
 #endasm



line(x1,y1,x2,y2,color,toggle)
/* V3.1A 6/28/86 JAY HOLOVACS */
/* USE TI BASIC COLOR AND LOCATION FORMATS
  COLOR=1-16
  X LOC=1-256
  Y LOC=1-192
  toggle: 0=erase
          !0=draw */

 int x1,y1,x2,y2,color,toggle;

 #asm
 MOV @2(14),@T$GL
 MOV @4(14),4
 DEC 4
 MOV @6(14),3
 DEC 3
 MOV @8(14),2
 DEC 2
 MOV @10(14),1
 DEC 1
 MOV @12(14),0
 DEC 0
 BLWP @L$NE14
 B @L$NE15
L$NE14  DATA L$WSP,L$NE1
L$WSP  BSS 32
X1$  BSS 2
X2$  BSS 2
Y1$  BSS 2
Y2$  BSS 2
INCX$  BSS 2
INCY$  BSS 2
XP$T  BSS 2
YP$T  BSS 2
```

```
L$NE1  MOV 13,12
 MOV *12+,@X1$
 MOV *12+,@Y1$
 MOV *12+,@X2$
 MOV *12+,@Y2$
 MOV *12,0
 MOV @X1$,3
 CI 3,255
 JH L$NE13
 MOV @X2$,5
 CI 5,255
 JH L$NE13
 S 3,5
 MOV 5,11
 ANDI 11,>8000
 MOV @Y1$,6
 CI 6,191
 JH L$NE13
 MOV @Y2$,8
 CI 8,191
 JH L$NE13
 S 6,8
 MOV 8,10
 ABS 10
 MOV 5,9
 ABS 9
 C 9,10
 JLT YGRT$
 MOV 5,1
 ANDI 1,>8000
 JEQ L$NE5
 LI 1,>FF00
 JMP L$NE6
L$NE5  LI 1,>0100
L$NE6  MOV 1,@INCX$
 MOV 8,11
 ABS 8
 SLA 8,8
 CLR 7
 DIV 9,7
 MOV 11,11
 JGT L$NE12
 NEG 7
L$NE12  MOV 7,@INCY$
 JMP L$NE10
YGRT$  MOV 8,1
 ANDI 1,>8000
 JEQ L$NE7
 LI 1,>FF00
 JMP L$NE8
```

```
L$NE7  LI 1,>0100
L$NE8  MOV 1,@INCY$
 CLR 4
 MOV 5,11
 ABS 5
 SLA 5,8
 DIV 10,4
 MOV 11,11
 JGT L$NE11
 NEG 4
L$NE11  MOV 4,@INCX$
L$NE10  SLA 3,8
 MOV 3,@XP$T
 SLA 6,8
 MOV 6,@YP$T
L$NE9  MOV @XP$T,2
 SRL 2,8
 MOV @YP$T,1
 SRL 1,8
 C 2,@X2$
 JNE L$NE4
 MOV 9,9
 JNE L$NE13
 C 1,@Y2$
 JNE L$NE4
L$NE13  RTWP END
L$NE4  BLWP @PL$T3
 A @INCX$,@XP$T
 A @INCY$,@YP$T
 JMP L$NE9
L$NE15
 #endasm


plot(x,y,color,toggle)
 int x,y,color,toggle;
/* v 3.0A 6/27/86 */
/* USES TI BASIC COLORS AND LOCATIONS
   COLOR=1-16
   X LOCATION=1-256
   Y LOCATION=1-192
   TOGGLE 0=ERASE
          !0=DRAW

NOTE: USES WKS IN CPU PAD--IF YOU NEED THIS SPACE
 FOR OTHER PURPOSES, CHANGE:
PLW$P EQU >8330
 to:
PLW$P BSS 32 */

 #asm
```

```
 MOV @2(14),@T$GL
 MOV @4(14),0
 DEC 0
 MOV @6(14),1
 DEC 1
 MOV @8(14),2
 DEC 2
 BLWP @PL$T3
 B @PL$T4
PL$T3  DATA PLW$P,PL$T1 PLOT ENTRY POINT
T$GL  BSS 2
PLW$P  EQU >8330
PL$T1  MOV 13,12
 LIMI 0
 MOV *12+,10
 SLA 10,12
 MOV *12+,2
 MOV *12,1
 CI 1,255
 JH PL$T2
 CI 2,192
 JH PL$T2
 MOV 1,8
 MOV 2,6
 SRA 1,3
 SRA 2,3
 MOV 1,4
 SLA 4,3
 S 8,4
 NEG 4
 INC 4
 MOV 2,5
 SLA 5,3
 S 6,5
 NEG 5
 MOV 2,3
 SLA 3,5
 A 1,3
 SLA 3,3
 A 5,3
 MOV 4,0
 LI 7,>0100
 SRA 7,0
 MOV 3,0
 BLWP @VSBR
 SWPB 7
 MOV @T$GL,@T$GL
 JEQ PL$T5
 SOCB 7,1
 JMP PL$T6
```

```
PL$T5  SZCB 7,1
PL$T6  BLWP @VSBW
 AI 0,>2000
 BLWP @VSBR
 ANDI 1,>0F00
 SOC 10,1
 BLWP @VSBW
PL$T2  RTWP
PL$T4
 #endasm

rect(x1,y1,x2,y2,color,toggle)
/* v2.0A 6/28/86 */
 int x1,y1,x2,y2,color,toggle);
{
 line(x1,y1,x1,y2,color,toggle);
 line(x1,y2,x2,y2,color,toggle);
 line(x2,y2,x2,y1,color,toggle);
 line(x2,y1,x1,y1,color,toggle);
}



circle(xcent,ycent,r,col,toggle) int xcent,ycent,r,col,toggle;
/* V2.1 6/28/86 based on an algorithm by Mike Higgins in
   Computer Language 5/86
   --corrected for .8 aspect ratio
   TO REMOVE ASPECT CORRECTION for more speed or
   other purposes delete lines noted and change all
   references of xx to x and yy to y */

   {
   int x,y,r1;
   int xx,yy; /* delete this line */
   x=r; y=0; r1=x/2;
   do
       {
       y++;
       r1=r1-y;
       if (r1<0)
           {
           x--;
           r1=r1+x;
           }
       xx=(8*x)/10;  /* delete this line */
       yy=(8*y)/10;  /* delete this line */
       plot(xcent+xx,ycent+y,col,toggle);
       plot(xcent+yy,ycent+x,col,toggle);
       plot(xcent-yy,ycent+x,col,toggle);
       plot(xcent-xx,ycent+y,col,toggle);
       plot(xcent-xx,ycent-y,col,toggle);
```

```
       plot(xcent-yy,ycent-x,col,toggle);
       plot(xcent+yy,ycent-x,col,toggle);
       plot(xcent+xx,ycent-y,col,toggle);
       }
   while(x>y);
   }



>>> BITWRT       4165 86/07/18
/* Source file with comments, use BITWRT for include */
/* REL 2.0B 6/29/86 ADDS TEXT CAPABILITY
CONTENTS:
  bitxt 1.1A
  bputch 2.2
  bputs 1.1
  btblnk 1.0
Added to REL 2.0
  bgets 1.1
  bgetch 1.0
  getky 1.0
  nxtcur 1.0
  bkspc 1.0
  blanks 1.0

TO BITRTN
  By:  Jay Holovacs
       95 King George Rd
       Warren, NJ 07060
       CIS 74756,413

Required files:

       C99C compiler by Clint Pulley
       BITRTN support                        */

#define CURSOR 95
bitxt()
/* v1.1A 12/8/85 must be called before
bitmap()
copies letter definitions into
storage buffer for use in bitmap
mode */

 #asm
 REF VMBR,VMBW
 B @T$T2
T$T  BSS 768
T$T1  BSS 8
T$T2  LI 0,>900
```

```
 LI 1,T$T
 LI 2,>300
 BLWP @VMBR
 #endasm


bputch(ascii,row,colmn,color)
 int ascii,row,colmn,color;


/* v2.2 6/26/86 or's char def onto
screen image ROW=1-24, COL=1-32 */
/* COLORS TIBASIC FORMAT */

 #asm
 MOV @6(14),0
 DEC 0
 SLA 0,5
 A @4(14),0
 DEC 0
 SLA 0,3
 LI 1,T$T1
 LI 2,8
 BLWP @VMBR
 MOV @8(14),3
 AI 3,-32
 JLT B$LP3
 CI 3,96
 JGT B$LP3
 SLA 3,3
 AI 3,T$T
 LI 5,T$T1
 LI 4,6
B$LP  SOC *3+,*5+
 DECT 4
 JOC B$LP
 BLWP @VMBW
 MOV @2(14),6
 DEC 6
 MOV 6,7
 SLA 6,12
 SLA 7,4
 SOC 7,6
 AI 0,>2000
 BLWP @VMBR
 LI 5,T$T1
 LI 4,6
 LI 3,>F0F0
B$LP2  SZC 3,*5
 SOC 6,*5+
 DECT 4
```

```
 JOC B$LP2
 BLWP @VMBW
B$LP3
 #endasm

bputs(row,colmn,color,str)
 int row,colmn,color;
 char *str;
/* v1.1 6/26/86 */

 {
 int ascii;
 while (*str!=0)
   {
    ascii=*str++;
    bputch(ascii,row,colmn,color);
    nxtcur(&row,&colmn);
   }
 }


 btblnk(row,colmn)
 /* v1.0 12/11/85 blanks a character space */

 int row,colmn;

 #asm
 B @BL$1
B$BF  DATA 0,0,0,0
BL$1  MOV @4(14),0
 DEC 0
 SLA 0,5
 A @2(14),0
 DEC 0
 SLA 0,3
 LI 1,B$BF
 LI 2,8
 BLWP @VMBW
 #endasm

bgetch(row,col,color) int row,col,color;
/* v1.0 6/26/86
   returns next key, echoes to bitmap screen if printable */
    {
    int ky;
    while ((ky=getkey())==0);
    if ((ky>31)&(ky<128))
       {
       btblnk(row,col);
       bputch(ky,row,col,color);
```

```
       }
    return(ky);
    }

bgets(buff,size,row,col,color) int row,col,color,size; char *buff;
/* v1.0 6/26/86
   returns printable characters to buffer specified
   places ascii 0 at end of string
   size=size of buffer to prevent overwriting at run time */
   {
   int ky,; char *bufpt, *endbf; bufpt=buff; endbf=bufpt+size-2;
   blanks(row,col,size);
   bputch(CURSOR,row,col,color); /* initial cursor */
   while ((ky=bgetch(row,col,color))!=13)
       {
       if (ky==8)
           {
           if (bufpt!=buff)
               {
               *bufpt--;
               btblnk(row,col);
               bkspc(&row,&col);
               btblnk(row,col);
               bputch(CURSOR,row,col,color);
               }
           }
       if ((ky>31)&(ky<127)&(bufpt!=endbf))
           {
           *bufpt++=ky;
           nxtcur(&row,&col);
           bputch(CURSOR,row,col,color);
           }
       if (bufpt==endbf)
           btblnk(row,col);
       }
   btblnk(row,col);
   *bufpt=0;
   return(buff);
   }

getkey()
/* v1.0 6/26/86
   scans keyboard, returns ascii
of key pressed, 255 if no key pressed */
   #asm
 REF KSCAN
 CLR 8
 MOVB 8,@>8374
 BLWP @KSCAN
 MOVB @>837C,8
 ANDI 8,>2000
```

```
 JEQ CHK1$
 MOVB @>8375,8
CHK1$  SWPB 8
   #endasm


nxtcur(rowpt,colpt) int *rowpt, *colpt;
/* v1.0 6/26/86
   increments cursor position when in bitmap text mode
   rowpt and colpt are pointers to current row and
   column values  format: nxtcur(&row,&col) */

   {
   if (*colpt==32)
       {
       *colpt=1;
       if (*rowpt==24)
           *rowpt=1;
       else
           (*rowpt)++;
       }
   else
       (*colpt)++;
   }

bkspc(rowpt,colpt) int *rowpt, *colpt;
/* v1.0 6/26/86
   decrements cursor location--call with the
   the format: bkspc(&row,&col) */
   {
   if (*colpt==1)
       {
       if (*rowpt==1)
           return;
       else
           (*rowpt)--;
           *colpt=32;
           return;
       }
   else
       (*colpt)--;
   }


blanks(row,col,cnt)
/* v1.0 6/29/86
   will blank out cnt character spaces */
int row,col,cnt;
   {
   while(cnt-->0)
```

```
        {
        btblnk(row,col);
        nxtcur(&row,&col);
        }
    }
```

Disk 68. Contents of file BOXDEM/C

```
/* This is a test of the box functions */

#asm
    REF MAKBOX,PUTBOX,ENDBOX,CHSET2,IVIDEO,PRINTF
#endasm
#include "DSK2.STDIO"
strcat(s,t)
char *s,*t;
{
  --s;
  while(*++s);
  while(*s++=*t++);
}
main()
{
    int c;
    char *strg1,*strg2,*strg3,*strg4;
    strg1="This is a test of the boxes routines.";
    c=0;
    locate(1,1);
    while(c<24){
      printf("%s \n",strg1);
      c++;
    }
    chset2();
    strg1="********** This is** a test **********";
    strg2="########## This is## a test ##########";
    strg3="                    this is a larger       BOX            ";
    strg4="   and consist of       two character         arrays.      ";
    strcat(strg3,strg4);
    ivideo(strg1);
    ivideo(strg3);
    makbox(10,4,strg1);
    makbox(10,4,strg2);
    makbox(10,4,strg1);
    makbox(20,6,strg3);
    strg2="This is a window that will appear on the screen.   And when popped";
    strg3="  off will restore the former screenbelow it.        ";
    strcat(strg2,strg3);
    ivideo(strg2);
    makbox(17,7,strg2);
    putbox(1,7,10);
    putbox(2,10,10);
    putbox(3,12,15);
    putbox(4,8,8);
    putbox(5,17,15);
    locate(15,1);
    c=getchar();
```

```
    endbox(4);
    c=getchar();
    endbox(3);
    c=getchar();
    endbox(2);
    c=getchar();
    endbox(1);
    c=getchar();
    endbox(5);
}
```

Disk 68. Contents of file EXPLST;C

```
/* explst - expanded listing program v1.1
**
** this program is used to list assembly language or
** c source files. The COPY and #include directives
** are executed to produce a complete listing. As in
** their respective languages, the directives cannot
** be nested. Null filename inputs default to the
** keyboard and screen. Pressing a key will pause the
** program, CLEAR (FCTN-4) will abort.
**
** Clint Pulley  86/06/14
*/
#include "dsk1.stdio"
#define LEN 81
char fn[40],line[LEN];
int c,inp,out,cpy;
char delim[]="\n-  -  -  -  -\n\n";
main()
{ while(1)
  { putchar(FF);
    puts("c99 explst v1.1\n\n");
    inp=getfn("Input","r");
    out=getfn("Output","w");
    putchar('\n');
    while(fgets(line,LEN,inp))
    { outline();
      chklin();
    }
    fclose(inp);
    fclose(out);
    puts("\nmore copies?");
    c=getchar();
    if((c=='N')|(c=='n'))break;
  }
  exit(7);
}
/* getfn returns unit # or 0 if null name entered
*/
getfn(text,m) char *text,*m;
{ int unit;
  unit=0;
  while(1)
  { puts(text);
    puts(" filename?");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
```

```
  }
  return(unit);
}
/* output a line and check keyboard
*/
outline()
{ fputs(line,out);
  if(!out)putchar(EOL);
  poll(YES);
}
/* scan line for COPY or #include
** if found, copy referenced file
*/
chklin()
{ int i;
  char *ptr;
  i=0;
  while(line[i]==' ')++i;
  ptr=line+i;
  if(strmat("COPY ",ptr))copy2(i+4);
  if(strmat("#include ",ptr))copy2(i+8);
}
/* open secondary file and copy
*/
copy2(i) int i;
{ char *ptr;
  while(line[++i]==' ');
  ptr=line+i;
  if(*ptr=='"') {++ptr; ++i;}
  while(*ptr!='"' & *ptr>' ')++ptr;
  *ptr=NULL;
  if((cpy=fopen(line+i,"r"))==0)
  { puts("\nCan't open ");
    puts(line+i);
    putchar(EOL);
    return;
  }
  fputs(delim,out);
  while(fgets(line,LEN,cpy))outline();
  fclose(cpy);
  fputs(delim,out);
}
/*
** return true if s matches t up to
** the NULL byte
*/
strmat(s, t) char *s, *t;
{ while(*s == *t)
  { if(*t == 0) return (1);
    ++s; ++t;
  }
```

```
  if(*s==0)return 1;
  return 0;
}
```

Disk 68. Contents of file GRFTST;C

```
/* test of graphics 1 library
*/
#include "dsk1.grf1rf"
main()
{ puts("In text mode\n\n");
  chrdef(42,"183C7EFFFF7E3C18");
  pause(" hchar test");
  hchar(11,1,42,120);
  pause(" vchar test");
  vchar(1,20,42,72);
  pause(" gchar test");
  hchar(20,24,126,1);
  putchar('\n');
  putchar(gchar(20,24));
  putchar('\n');
  pause(" chrset test");
  chrset();
  pause(" patcpy test");
  patcpy(36,42);
  pause(" grf1 mode");
  grf1();
  chrdef(42,"183C7EFFFF7E3C18");
  puts(" In graphics 1 mode\n\n");
  pause(" magenta screen");
  screen(14);
  pause(" red spaces");
  color(4,2,10);
  pause(" redefine space");
  chrdef(32,"000000000000001");
  pause(" clear screen");
  screen(5);
  color(4,2,1);
  clear();
  pause(" hchar test");
  hchar(11,1,42,96);
  pause(" vchar test");
  vchar(1,15,42,96);
  pause(" gchar test");
  hchar(20,24,126,1);
  putchar('\n');
  putchar(gchar(20,24));
  putchar('\n');
  pause(" key test");
  keytst();
  pause(" joyst test");
  jsttst();
  clear();
  pause(" sprite test 1");
```

```
  sprt1();
  chrdef(32,"0");
}
keytst()
{ int cv,st;
  while(cv!=18)
  { cv=key(1,&st);
    if(st==1)putchar(cv);
  }
}
jsttst()
{ int xj,yj,xv,yv,cv,st;
  clear();
  xv=16;
  yv=12;
  while(key(1,&st)!=18)
  { joyst(1,&xj,&yj);
    xv=xv+(xj>>2);
    yv=yv-(yj>>2);
    if(xv<1)xv=32;
    if(xv>32)xv=1;
    if(yv<1)yv=24;
    if(yv>24)yv=1;
    vchar(yv,xv,42,1);
  }
}
sprt1()
{ int i,j,r,c;
  char cb[7];
  sprite(5,64,8,60,100);
  delay(30);
  sprite(0,80,6,60,60);
  delay(30);
  spmag(4);
  i=0;
  while(++i<=16)
  { spcolr(5,i);
    spcolr(0,17-i);
    delay(10);
  }
  spcolr(0,16); spcolr(5,2);
  sppat(0,84);
  delay(30);
  i=j=0;
  while(j<192)
  { sploct(0,++j,++i);
    sploct(5,160,i);
    delay(1);
  }
  screen(8);
```

```
  sploct(0,60,60); sploct(5,120,120);
  i=0;
  while(++i<5)
  { spmag(i);
    delay(30);
  }
  screen(12);
  sploct(0,80,0); sploct(5,80,120);
  spmotn(0,0,4);
  spmct(1);
  i=j=0;
  while(j<255)
  { spposn(0,&i,&j);
    locate(23,2);
    puts("row="); puts(itod(i,cb,5));
    puts(" col="); puts(itod(j,cb,5));
    puts(" dsp="); puts(itod(spdist(0,5),cb,7));
    locate(24,2);
    puts("drc="); puts(itod(spdrc(0,96,128),cb,7));
    PUTS("  ");
    if(spcnc(0,5,4)) puts("spcnc"); else puts("     ");
    if(spcrc(0,80,180,8)) puts(" spcrc"); else puts("      ");
    if(spcall()) puts(" spcall"); else puts("       ");
    poll(1);
  }
  spdel(0);
  pause(" done!\n");
  spdall();
}
pause(c) char *c;
{ puts(c);
  getchar();
}
delay(n) int n;
{ n=1000*n;
  while(n--);
}
/*
** str=itod(nbr,str,sz) -
**  convert nbr to signed decimal string of width sz
**  right justified, blank filled, result in str[].
**  sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
```

```
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
}
```

Disk 68. Contents of file TTT;C

```
/*
* 3D Tic Tac Toe.
*
* Adapted to c99 by Clint Pulley
*
* Last edit 86/03/01 1830
*/
#include "dsk1.conio"
#define  EMPTY  0
#define  PLAYER 1
#define  PLT2   2
#define  PLT3   3
#define  PLT4   4
#define  BEAST  5
#define  BET2  10
#define  BET3  15
#define  WDIM 304
/*
* This is a table of all winning
* combinations, stolen from
* Kilobaud, April 78.
* You can look there to see how it
* is ordered.
*/
int w[WDIM] = {
  0,  1,  2,  3,
  4,  5,  6,  7,
  8,  9, 10, 11,
 12, 13, 14, 15,
  0,  4,  8, 12,
  1,  5,  9, 13,
  2,  6, 10, 14,
  3,  7, 11, 15,
  0,  5, 10, 15,
  3,  6,  9, 12,
 16, 17, 18, 19,
 20, 21, 22, 23,
 24, 25, 26, 27,
 28, 29, 30, 31,
 16, 20, 24, 28,
 17, 21, 25, 29,
 18, 22, 26, 30,
 19, 23, 27, 31,
 16, 21, 26, 31,
 19, 22, 25, 28,
 32, 33, 34, 35,
 36, 37, 38, 39,
 40, 41, 42, 43,
```

```
 44, 45, 46, 47,
 32, 36, 40, 44,
 33, 37, 41, 45,
 34, 38, 42, 46,
 35, 39, 43, 47,
 32, 37, 42, 47,
 35, 38, 41, 44,
 48, 49, 50, 51,
 52, 53, 54, 55,
 56, 57, 58, 59,
 60, 61, 62, 63,
 48, 52, 56, 60,
 49, 53, 57, 61,
 50, 54, 58, 62,
 51, 55, 59, 63,
 48, 53, 58, 63,
 51, 54, 57, 60,
  0, 16, 32, 48,
  1, 17, 33, 49,
  2, 18, 34, 50,
  3, 19, 35, 51,
  4, 20, 36, 52,
  5, 21, 37, 53,
  6, 22, 38, 54,
  7, 23, 39, 55,
  8, 24, 40, 56,
  9, 25, 41, 57,
 10, 26, 42, 58,
 11, 27, 43, 59,
 13, 29, 45, 61,
 12, 28, 44, 60,
 14, 30, 46, 62,
 15, 31, 47, 63,
  0, 21, 42, 63,
  4, 21, 38, 55,
  8, 25, 42, 59,
 12, 25, 38, 51,
  1, 21, 41, 61,
 13, 25, 37, 49,
  2, 22, 42, 62,
 14, 26, 38, 50,
  3, 22, 41, 60,
  7, 22, 37, 52,
 11, 26, 41, 56,
 15, 26, 37, 48,
  0, 20, 40, 60,
  0, 17, 34, 51,
  3, 18, 33, 48,
  3, 23, 43, 63,
 12, 24, 36, 48,
```

```
 12, 29, 46, 63,
 15, 30, 45, 60,
 15, 27, 39, 51
};
/*
* This is the board.
*/
int b[64];
char  *sep  = "    ------- ------- ------- -------\n";
/*
* Working storage
*/
#asm
 AORG >8330
#endasm
int i, j, t;
int s, bs, bt;
#asm
 RORG
#endasm
int v[76];
char buf[20];
/*
*/
main()
{
  puts("Do you want the rules? ");
  gets(buf);
  if(buf[0]=='Y' | buf[0]=='y')
    rules();
  for(i=0 ; i<64 ; ++i)b[i]=EMPTY;
  puts("Do you want to go first? ");
  gets(buf);
  if(buf[0]=='Y' | buf[0]=='y')
    user();
  for(;;) {
    beast();
    user();
  }
}
/*
 * Print the rules of the
 * game.
 */
rules()
{
  puts("Three  dimensional tic-tac-toe is played");
  puts("on a 4X4X4 board.  To win you must get 4");
  puts("in a row.  Your moves are specified as a");
  puts("3  digit number;  the first digit is the");
  puts("level,  the second the row and the third");
```

```
  puts("the column.  Levels and columns go  from");
  puts("left to right from 0 to 3.  Rows go from");
  puts("top to bottom with 0 on the top.\n\n");
}
/*
 * Accept a user move.
 * Exit if he wins.
 */
user()
{
  board();
  for(;;) {
    puts("\nYour move? ");
    if(gets(buf) == NULL) {
      puts("\nChicken.\n");
      exit(0);
    }
    i = 16*(buf[0]-'0') + (buf[1]-'0') + 4*(buf[2]-'0');
    if(i>=0 & i<=63 & b[i]==EMPTY)
      break;
    puts("\nEh?\n");
  }
  b[i] = PLAYER;
  for(i=0; i<WDIM; i=i+4) {
    t = 0;
    for(j=0; j<4; ++j)
      t = t+b[w[i+j]];
    if(t == PLT4) {
      puts("\nYou win.\n");
      exit(0);
    }
  }
}
/*
 * Display the board.
 * Not as easy as it sounds.
 */
board()
{
  puts("\f\n");
  for(i=0; i<4; ++i) {
    if(i != 0)
      puts(sep);
    puts("    ");
    for(j=0; j<64; j=j+4) {
      psq(i+j);
      if(j==12 | j==28 | j==44)
        putchar(' ');
      else if(j >= 60)
        putchar('\n');
```

```
      else
        putchar('|');
    }
  }
}
/*
 * Format and put out square
 * `s' of the board.
 */
psq(s) int s;
{
  switch(b[s])
  {
  case PLAYER :
    putchar('U');
    return;
  case BEAST :
    putchar('C');
    return;
  default :
    putchar(' ');
  }
}
/*
 * Move for the machine.
 * Just exit on machine wins
 * and draws.
 */
beast()
{
  for(i=0; i<WDIM; i=i+4) {
    t = 0;
    for(j=0; j<4; ++j)
      t = t+b[w[i+j]];
    v[i>>2] = t;
    if(t == BET3)
      break;
  }
  if(i < WDIM) {
    for(j=0; j<4; ++j)
      if(b[w[i+j]] == EMPTY) {
        b[w[i+j]] = BEAST;
        break;
      }
    board();
    puts("\nI win.\n");
    exit(0);
  }
  bt = 0;
  for(s=0; s<64; ++s) {
    if(b[s] != EMPTY)
```

```
      continue;
    t = 0;
    for(i=0; i<WDIM; i=i+4) {
      for(j=0; j<4; ++j)
        if(w[i+j] == s)
          break;
      if(j != 4) {
        if(v[i>>2] == PLT3) {
          b[s] = BEAST;
          return;
        }
        t = t+weight(v[i>>2]);
      }
    }
    if(t > bt) {
      bt = t;
      bs = s;
    }
  }
  if(bt != 0)
    b[bs] = BEAST;
  else {
    for(s=0; s<64; ++s)
      if(b[s] == EMPTY)
        break;
    if(s == 64) {
      puts("\nDraw.\n");
      exit(0);
    }
    b[s] = BEAST;
  }
}
/*
 * Given a total along a winning
 * combination, return the weight
 * value.
 */
weight(at) int at;
{
  switch(at)
  {
  case PLAYER :
    return(1);
  case PLT2 :
    return(4);
  case BEAST :
    return(1);
  case BET2 :
    return(2);
  default :
```

```
    return(0);
  }
}
```

Disk 69. Colossal Caves

Version: 1.1 *Author*:
Requires: Adventure *Language*: *Updated*: 11/19/86

The original adventure game finally translated to the TI. This game was the precursor to the classic Zork series. Hours of text adventure solving fun.

```
dskdir. v2.0. 12-dec-96
Disk name               = COLCAVE#69
Sectors total           = 360
Sectors used            = 107
Sectors available       = 251
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CAVE1        53 PROGRAM    Y >020 052
002 >003 CAVE2        43 PROGRAM    Y >054 031 >078 011
003 >004 CAVEINFO     11 DIS/VAR 80 Y >073 005 >083 005
```

Disk 69. Contents of file CAVEINFO

THE COLOSSAL CAVE INFORMATION

This is the original Adventure from which all other versions, both micro and mainframe, evolved. The game of Adventure understands English words and phrases and as such will attempt to do what you command.

The object of the game is to explore a cave, find a quantity of treasures and deposit them back in the building. All exits from a location are not necessarily visible! Simple? Good luck! (You'll need it to get past the snake.)

WELCOME TO ADVENTURE

Somewhere nearby is Colossal Cave, where others have found fortunes in treasure and gold, though it is rumored that some who enter are never seen again. Magic is said to work in the cave. I will be your eyes and hands. Direct me with commands of 1 or 2 words. I should warn you that I only look at the first five letters of each word.

Saving, quitting and scoring this game work exactly as they do in all Scott Adams' Adventure series.

Load CAVE1 to begin and then follow instructions in the room thereafter. This game contains all 145 locations of the original adventure, including the MASTER'S GAME. The game is currently at 100% capacity, so you will have to do some SAVEing and LOADing in order to work between the mazes and the main cave.

This version doesn't have the complex score keeping system of the original either, but I hope to include that in a future edition.

If you think you've found all the treasures, just keep exploring for a while. All of the treasure now exist somewhere in the cave and when you have found them all, you will be further advised what to do in order to get to the MASTER'S GAME.

No donation is expected or asked for. This has just been a lot of fun. Please feel free to put this on any TIBBS or add it to any TIUG library. I would appreciate any comments you have on the work. Please send mail to:

CIS 72257,3722
STC TI6010

P.S. Hints are also available at the above addresses.

Disk 70. Mass Transfer

Version: 4.1 *Author*: Stu Olsen
Requires: XB, EA *Language*: AL *Updated*: 08/16187

A great terminal emulator program including multiple file transfers, XMODEM file transfers, telephone number list, large incoming text buffer, and in general a much more user friendly program than the now classic Fast-term.

```
dskdir. v2.0. 12-dec-96
Disk name               = MT43
Sectors total           = 360
Sectors used            = 354
Sectors available       = 4
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >006 MASS         33 PROGRAM      >0f4 032
002 >003 MASSDOCS1    88 DIS/VAR 80   >07f 087
003 >002 MASSDOCS2    94 DIS/VAR 80   >022 093
004 >007 MAST         32 PROGRAM      >114 031
005 >008 MTDOCS4/2    75 DIS/VAR 80 Y >133 053 >009 021
006 >005 PHONE1       10 DIS/FIX 80   >0eb 009
007 >004 PHONEMAKE    22 PROGRAM      >0d6 021
```

Disk 70. Contents of file MASSDOCS1

"MASS-TRANSFER" Modem Utility Version 4.1

If you are using this program, and have not yet paid for it, please send $10 check or money order to the author for his work.

I'm allowing you to try this software before you decide to buy it. As such, I am trusting you to pay for it if you should decide to keep it. If you do not like it, please pass it along to other users, as they may find a need for this program. Good response from you will encourage programmers like myself to provide you with software at a reasonable price and without copy protection. Like they say, you can start paying us now or pay a software dealer 3 times the price for their programs.

STUART OLSON SYSOP: TI-85033 (602) 848-6200
6625 W. COOLIDGE ST.
PHOENIX, AZ. 85033

MASS-TRANSFER is a terminal emulator program written in 9900 assembly language. Through its menu driven screens, it supplies the user with a variety of options with which to enhance the telecommunications of the TI computer.

SYSTEM REQUIREMENTS

The minimum system necessary for operation is the TI-99/4A console, 32k Memory Expansion, RS232 card, disk drive, and a modem. Either the Editor/Assembler or Extended Basic cartridge may be used.

GENERAL INFORMATION

This software employs a unique feature provided by the TI-99/4A computer. The RS232 card contains its own programming inside a special Device Service Routine (DSR) rom. This rom normally handles the computers file management. It also allows for the use of Circular Interrupt Buffer (CIB) operation. This routine interrupts the running program each time a character is received and places that character into a special buffer in memory. This program periodically checks the CIB and upon finding new characters, processes them as either control characters or data. The CIB overwrites itself if more than 255 characters are received and not processed. This type of programming results in an interrupt driven input for the computer, allowing the user to leave the terminal mode and return to the main menu for short periods of time without any data loss.

MASS-TRANSFER maintains its own user managed telephone directory. The accompanying extended basic program (PHONEMAKE) allows the user to load a phone directory with up to 20 of the most frequently called numbers. The directory can be easily updated with the program's editing feature. The desired phone number can be accessed via the auto-dial feature. This auto-dialing can only be accessed if a smart-modem with autodial capacity is being used. Each of the 20 listings allow the user to send up to 50 characters to the modem. This allows the user to send special modem control codes prior to the actual dialing of the desired phone number.

Note: In order to use the auto-dial phone directory option, the extended basic program that creates the directory should first be used. To do this, load and run the following program OLD DSK1.PHONEMAKE. If this program is being ran for the very first time, then select option 1 from the menu, which will create a blank directory on the disk you have in drive #1. To provide for proper modem operation, you must answer all the prompts in the "change modem commands" section. This program is also menu driven. Follow the screen prompts to load the directory with your phone numbers and modem commands.

INTRODUCING "MXT" (Multiple Xmodem Transfer)

The MXT file transfer option is an ALL NEW concept for the TI and its growing xmodem file transfer capabilities. The MXT supervisor routine allows the user to select any number of files from a disk, and send them without the boring task of typing in numerous filenames and other associated key combinations as used by other xmodem transfer programs. If used with a smartmodem, the program will even disconnect itself from the phone line after the transfer is complete or should more than 10 retries be encountered on any one record.

If the MXT system is used on both sending and receiving computers, the modified xmodem transfer is automatic, allowing both users to walk away from the computer, returning later to turn off the equipment.

MXT is capable of SENDING your selected disk files to OTHER xmodem terminal emulators also. The receiving system will still have to type in filenames, however the MXT supervisor sends the other computer a message containing the proper filename of the file to be transferred. This allows MXT to work with such programs as Fast-Term and 4A/Talk.

MXT uses a modified file header to convey the necessary info for the receiving MXT system, including the proper filename and status as to any more files to be sent after the present one is finished. Although the file header is modified, it still works with those other programs previously mentioned. Since the other xmodem emulator programs do not send the modified file header, MXT can NOT be used for receiving files from these other programs.

ERRORS

Mass-Transfer uses on screen I/O error messages if problems are encountered while accessing a peripheral. For your convenience, the standard TI error codes are listed below.

Code	*Meaning*
00	Bad device name
01	Device is write protected
02	Bad open attribute such as incorrect file type
03	Illegal operation; ie., an operation not supported on the peripheral or a conflict with the open attributes
04	Out of table or buffer space on the device
05	Attempt to read past end of file
06	Device error. Covers all hard device errors
07	File error such as program/data file mismatch or file not found.

SYSTEM DEFAULTS

Upon entering the program, the following are in effect:

RS232 card #1, port #1
300 baud rate
8 bit character length (terminal mode masked to 7)
1 stop bit
No parity check
Auto Hangup Off
Local echo Off
Remote echo Off
Buffer capture On
Log file Off
Linefeed Off
Print spooler off, set for PIO

CONTROL and FUNCTION KEYS

The following control and function keys are active along with auto repeating of all keys:

CTRL

G	Bell	Produces a beep
H	BS	Move cursor one space left, deleting the previous character
J	LF	Scrolls the display screen one line
R	-	Changes some RS232 parameters when at main menu
L	FF	Clears the screen and homes cursor
/	-	Causes the program to generate a break character and also CANCELS the AUTO-DIAL routine once it has been started
=	-	Changes screen color

FCTN

S	BS	Left arrow, provides same function as Control H
X	LF	Down arrow, provides same function as Control J
1	-	Changes text color.
4	-	FCTN CLEAR allows the user to abort most functions
7	-	FCTN AID exits the Terminal mode and returns to the main menu screen
=	-	Prompts you for exiting the program. Toggles print spooler on/off in terminal mode
D	-	Right arrow, provides same function as the space bar.
9	-	When in terminal mode, allows a screen dump to be done.
Enter	CR	Positions cursor at the beginning of the line.

LOADING THE PROGRAM

Editor/Assembler using option 5: Enter the filename as: DSK1.MASS

Extended Basic module: Enter the filename as: DSK1.LOAD

CUSTOMIZING THE PROGRAM

Note: It is assumed the user of this info has an understanding of how a sector editor works, especially the one that will be used to alter the MASS file. You should make a copy of the MASS file first, and attempt your changes on that copy. If a mistake is made, you will not destroy your only "working" version of the program. If you do not know how to use a sector editor, I would recommend that you contact a local user group or an individual who is knowledgeable in the use of sector modifications.

Directions:

1. Using a sector editor, locate the first sector of the MASS file. This is the sector in which all modifications will be made.

2. Locate the byte(s) within the sector to change. The following tables gives the location and function of the bytes and defaults that are available.

3. After you have changed the desired bytes, write the edited sector back to disk.

Note: I count bytes assembly style. In other words, the first byte in a sector is 0 and the last byte is 255. I am using decimal numbers for the byte #'s and hex numbers ">" for the default and optional numbers.

Note: Most RS232 cards have 1 PIO port and 2 serial ports. Most users only have 1 RS232 card. If so, the serial ports are referred to as RS232/1 and RS232/2. If two RS232 cards are installed in your system, the second card then becomes RS232/3 and RS232/4, and the second PIO port becomes PIO/2. Since each card has two serial ports, the computer addresses these by using two separate notations; the card address and the port address. The following tables show you which hex codes to edit into the program for setting up both the card and port addresses, depending on your system configuration.

Byte #	*Default*	*Function and Optional values*
10-11	>1300	Selects first or second RS232 CARD.
13	>F4	Selects character and screen colors. The first digit is character color. Second digit is screen color.
15	>40	Selects the RS232 port within the card.
16-17	>04D0	Selects the modem baud rate.
42-43	>03E8	Delay time before auto-repeat for keyboard (>03E8=1000)
44-45	>003C	Delay between characters once keyboard has started auto-repeating. (>003C=60)
48-49	>0000	Print spooler type. >0000=TI card PIO >0002=TI, CorComp, Myarc card serial
50-51	>1300	Select first or second RS232 card for print spooler.
52-53	>0040	Selects the RS232 serial print spooler port within the card.
54-55	>0034	Selects the baud rate for serial print spooler.
56-57	>8300	Selects number of data bits, parity, and stop bits for serial print spooler

Color codes (in hex)

0	transparent
1	black
2	medium green
3	light green
4	dark blue
5	light blue
6	dark red
7	cyan
8	medium red
9	light red
A	dark yellow
B	light yellow
C	dark green
D	magenta
E	gray
F	white

Baud rate codes (in hex)
>0638 = 110 baud
>04D0 = 300 baud
>0341 = 600 baud
>01A0 = 1200 baud
>00D0 = 2400 baud
>0068 = 4800 baud
>0034 = 9600 baud

RS232 card and port codes (in hex)
>1300 = RS232 card #1
>1500 = RS232 card #2
>40 = RS232/1 or /3
>80 = RS232 /2 or /4

Data bits, parity, stop bits
>8200 = 7N1 >8300 = 8N1
>A200 = 7E1 >A300 = 8E1
>B200 = 7O1 >B300 = 8O1

SCREEN DUMP OPERATION

While in the terminal mode, a screen dump can be done without having to go to the view buffer mode. To dump a screen, the FCTN and 9 keys are used. It is advised to dump a screen when the sending computer, BBS, etc. has stopped sending data. If this is not handy, and the other computer responds to XON/XOFF control keys (control S and Q), send a control S to stop the other computer's data flow. Then, dump your screen to any device you wish to enter (disk, printer, etc.). After the dump is finished, enter a control Q to tell the other computer to continue the data transmission.

PRINT SPOOLER OPERATION

The print spooler in M/T shares the same memory space as the buffer. As such, a few things should be noted while using the print spooler. First, you must make sure the 1st sector of MASS contains the proper code to match the program with your type of printer. If you have not already done so, refer to the previous steps on sector editing and make the necessary changes.

Note: I HAVE NOT BEEN ABLE TO OBTAIN THE NECESSARY INFO NEEDED TO ALLOW M/T TO WORK WITH THE MYARC AND CORCOMP PIO PORTS. THEY ARE ACCESSED DIFFERENTLY THAN THE TI PIO PORT. UNTIL SOMEONE IS KIND ENOUGH TO PROVIDE ME WITH THE CRU ADDRESSING INFO, I'M AFRAID I CAN NOT IMPLEMENT PRINT SPOOLING FOR USE WITH CORCOMP AND MYARC PARALLEL USE.

The print spooler is toggled on and off by simply press the FCTN and = keys while in the terminal mode.

Your printer can fall behind the currently received modem data if your modem is running at speeds faster than 300 baud. The print spooler is designed to allow the printer to lag the incoming data by up to 12.5K bytes. If this amount is exceeded, the program will initiate certain steps to prevent data loss. The buffer will close, preventing any more incoming data from being stored. The print spooler will continue to dump the remaining 12.5K of data to your printer. After the print spooler is empty, you can once again turn on your spooler and resume normal use.

Since the print spooler and buffer share the same memory space, M/T will automatically turn on your buffer if you turn on the printer spooler. If the buffer is already on, it will make no difference to the program's operation.

If you should have both the buffer and print spooler on, and decide to turn off the buffer, the print spooler will turn off also. Your printer will continue to print any data still left in the spooler. If you should then turn on both the buffer and the print spooler, you will re-initialize the print spooler. Any remaining unprinted data will be lost unless you wait for the spooler to empty. Turning the print spooler on and off will cause no problems.

Since many BBSs often pause while accessing the message base, your printer should be able to keep up with the incoming data without falling too far behind. You usually have the option of pausing somewhat, before reading the next message. This will give your printer a chance to catch up somewhat.

PHONEMAKE AND NEW PHONE FILES

The PHONEMAKE program for version 4.1 is the same as the one used in version 4.0. Nothing in it has been changed, except the version number itself.

You now have the option of using up to EIGHT PHONE files with M/T. M/T now provides you with a separate menu screen from which to make your selection. You have the option of selecting the "not using phone files". If this is selected, no data is loaded and any part of M/T that uses this data is made non-useable, to prevent any program crash.

You can load the PHONE file from almost any storage device; Myarc ramdisk, New Horizon ramdisk, or floppy disk drive. Just select the appropriate option from the menu and M/T will do the rest. If you receive an error while doing this, you most likely do not have the PHONE file available on that device, or the PHONE file was from a version earlier than 4.0.

NOTE: M/T EXPECTS THE PHONE FILES TO BE NAMED EITHER:

PHONE1, PHONE2, PHONE3, ... or PHONE8

Use your disk manager to rename your old PHONE file to and one of the above filenames.

NEW HORIZON RAMDISK SUPPORT

Although I do not have an HRD, I've been told that they can emulate DSK5. I have included in all parts of the program that require you to input a drive # the ability to enter a 5. This should allow full use of your HRD with this program.

Disk 70. Contents of file MASSDOCS2

MAIN MENU INSTRUCTIONS

<R> Reconfigure I/O port

This option allows the user to change the RS232 port and also the baud rate for that port. To reconfigure your system for a different setup, select R from the menu. The program will take you to the Configuration Setup screen. You will then be able to enter the desired communication parameters.

Although the RS232 port can be changed along with baud rate, several users requested an easier way to change data bits, parity, and stop bits. M-T is pre-programmed with the 3 standard protocols; those being 8,N,1; 7,O,1; 7,E,1. These refer to number of data bits (7 or 8), type of parity (none, odd, or even), and number of stop bits, respectively. To access this feature, just press CONTROL R (CTRL R) on the main menu. You will notice the 8N1 that was underneath the buffer size counter has now changed. Pressing CTRL R again will again bring up the next option, and one more time will bring you back to where you started. Once the selection is made, the RS232 port is automatically reset to your new specifications.

<M> Multiple Xmodem Transfer (MXT)

This option allows the user to send and receive an unlimited number of disk files without having to attend to the computer during the transfers.

When you select the send option, the program will catalog the disk and display the directory on the monitor, 10 files at a time. To select a file for transfer, position the cursor next to the desired filename and press the <ENTER> key. You may use the up and down arrow keys to move the cursor. To delete a previously selected file, position the cursor in front of the filename, and press the FCTN and 1 keys (FCTN DELETE). To proceed on to the next screen of directory entries, press the FCTN and 6 keys (FCTN PROCEED). When finished with the last directory screen, press FCTN PROCEED one more time and the MXT transfer will begin.

To abort during a transfer, press the FCTN and 4 (FCTN CLEAR) keys. The abort will cancel the current file being transferred and any other files still remaining in the supervisor list.

When using MXT to send files, the sending computer will pause for 5 seconds between files. This is so that the receiving system (if non-MXT capable) can type in the next filename as it appears on his monitor. If you wish to bypass the 5 second pause, just press the <ENTER> key.

Note: IF USING MXT IN THE RECEIVE MODE, USE A CLEAN DISK. THE MXT SUPERVISOR ROUTINE WILL CREATE THE FILENAMES ON YOUR DISK JUST AS THEY WERE LABELED ON THE SENDING DISK. IF YOU HAVE FILES ON YOUR DISK, THEY COULD BE ERASED AND OVERWRITTEN WITH THE NEWLY RECEIVED FILE IF THE FILENAMES ARE THE SAME.

<H> Hangup after MXT status = OFF

This option allows the user to pre-select the auto hangup feature upon completion of an MXT transfer.

The auto hangup will be executed when the transfer is either successfully completed OR aborted. This feature allows the user to leave the computer unattended for a period of time, and return later to see the status of the disk transfer. This option toggles ON and OFF by pressing the H key.

Note: THE DATA NEEDED BY M/T TO HANGUP YOUR MODEM IS CONTAINED IN THE PHONE FILE. FURTHER, A MODEM CAPABLE OF HANGING ITSELF UP THROUGH EITHER PROGRAM OR KEYBOARD CONTROL IS REQUIRED. IF YOUR MODEM IS NOT ABLE TO HANG ITSELF UP, THEN M/T WILL NOT BE ABLE TO PERFORM THIS FUNCTION. BE SURE TO RUN THE PHONEMAKE PROGRAM AND ENTER YOUR MODEM HANGUP COMMANDS IN THE APPROPRIATE SECTION.

<C> Clear download buffer

This option clears the buffer. Any previously received data will be lost. After the buffer has cleared, the program returns to the main menu.

<E> Echo remote status = OFF, monitor status = OFF

This option allows the user to select the necessary echo as needed. The user has control over both the remote and monitor echo. Varied use of this option will permit the user to interface with almost any other computer, regardless of what the remote computer provides/requires in the area of character echo.

To change either the remote or monitor echo status, press the E key. A four choice menu appears, allowing the user to turn on/off each echo. A beep will be heard, when a valid choice is selected from the menu. When all choices have been made, press ENTER. The program will return to the main menu, with the new echo status displayed.

<1> Remote echo - ON

Echo all received data back to the sending computer.

<2> Remote echo - OFF

Normally used when communicating with BBS's.

<3> Monitor echo - ON

This mode shows what you are typing but does not echo the received characters back to the remote computer.

<4> Monitor echo - OFF

Suppresses the echo of your typed characters to your monitor.

Note: The above combinations of echoes gives the user an excellent choice to use under various situations. However, one should keep in mind the following idea. If both computers are using an echo ON mode, the program usually starts typing a string of feedback characters since both computers are re-sending the same character over and over. If this should occur, it can be stopped by returning to the menu and changing the remote echo to off. Auto-dialing should not be done with the remote echo on. This echoing of modem responses usually causes the modem to misdial or lock up.

<U> Upload DIS/VAR 80 file

This option allows the user to send a display variable 80 file that is loaded in from disk and stored in the buffer. It will erase the contents of the buffer. Be sure to save anything of importance in the buffer before you select this option.

NO ERROR CHECKING is provided in the program as to the SIZE of the file being loaded. Since the file is placed in the buffer, care should be taken not to exceed the size of the buffer. A DV-80 file of about 40 to 50 sectors in length will easily fit into the available buffer space. Since this text sending feature is normally used to send a pre-composed message to a BBS, this size limitation should be of no real problem.

After the file is loaded, you must determine who will control the sending of the text. You can let the computer manage it, or you can take control and send it one line at a time. It is suggested that if you are uploading to a BBS, the line by line option may be the best choice. Experimentation on your part may be necessary. Follow the instructions on the screen which will tell you how to send the data. As the data is sent, a carriage return is added to the end of each line automatically.

While sending, any prompts or text that are sent or echoed by the BBS will also appear on your monitor. When the last line has been sent, you will be placed back into terminal mode.

<A> Auto-Dial from directory

After selecting the "[A]" option from the main menu, you will see your phone file on screen with three options available for use.

[D]ial is used to dial any phone number you have included in your phone list. It will dial that number only once, and return you to the terminal mode. To dial a number, simply select the desired letter (A-T). Your smart modem will now dial the number for you.

[R]edial is used when you wish M/T to dial a possibly busy phone number. It will continue to redial the busy number until the line is no longer busy or you cancel the dialing process manually. To redial any number, simply select the desired letter (A/T). After you select the first number to dial, M/T prompts you for an alternate number. Yes, M/T is able to redial two different numbers until it gets through to one of them. To use this feature, just enter your second choice. If you do not wish to dial an alternate number, just press the enter key, and M/T will skip the alternate redial feature.

While M/T is redialing numbers, it will display the current number being dialed, along with the status (busy or dial) and the number of redialing attempts for that number.

[P]c-Pursuit is the last option available in the auto-dialer. It allows you to let the computer REDIAL any PCP area code you desire. On my system, I have found that M/T will redial a busy area code listing 10 times per minute. At that rate, it is quite easy to get an access line.

In order to make use of the PCP feature, you'll need a special file(s) on disk that contain the necessary info required by PCP. (ie, password, ID, area code, and baud rate). You can use a standard D/V80 file editor to create these files, or make use of the editor provided in PHONEMAKE (option 6). If you use an editor such as TI-Writer, be sure to use the "PF" option to save your file to disk, as M/T will not work properly with the margin info normally found at the end of a TI-Writer file.

If you use the PHONEMAKE editor to create your file, all the info needed by PCP is display on your monitor. Just enter your specific data. After that, your PCP file will be saved to disk, under whatever filename you wish. Personally, I use filenames such as 312/12 which indicates area code 312 and 1200 baud.

Assuming you have made up the necessary PCP files, you can auto-dial an area code by first connecting with your local PCP access line. After you arrive at the PCP prompt where you would type in the usual C DIAL 312/12.....etc, just select the autodialer function from the main menu, followed by the [P]c-Pursuit option. You will be prompted for the file name (remember the ones you made using one of the DV80 editors).

Enter the name, and M/T will now faithfully redial that "always busy" area code for you. When it connects, you are signaled with a beep from the program (just like before).

At this point, you could use the auto-dialer to dial the BBS number you are trying to logon to. To do so, return to the main menu, select the autodial option again, and this time use the redial feature to dial the BBS you desire.

Note: If you have PC-Pursuit but do not have a smart-modem, you can still make use of the auto-dialer features once you are connected to a PC-Pursuit access line!

CANCELING THE AUTO-DIALER

To cancel the auto-dialer, press the break key (CTRL /). This signals M/T to stop auto-dialing and searching for any modem responses.

One last comment on using the auto-dialer. The two responses you supply in the PHONEMAKE program are what is used by M/T for modem response comparison. My Hayes modem gives a NO CARRIER for a busy response while my use of PCP shows that BUSY is returned for a busy number. Keep these modem responses in mind as you enter them in the PHONEMAKE program. Without the proper responses and commands, M/T will not be able to control your smart-modem.

<D> Dump download buffer

This option copies the buffer to a storage device. This device can be any legal device name that supports a maximum file length of 80 characters, except cassette. Upon selection of this option, enter the appropriate filename, such as PIO, DSK1.SAVEFILE, RS232/2.BA=4800.DA=8, etc. You can also abort the dump by following the screen prompts. In order to allow the same buffer contents to be saved to several devices, the buffer is NOT CLEARED after the dump routine.

Note: This buffer is shared with the LOG file. However, even if you have the LOG file on, you may still dump the buffer contents to any device you desire.

<S> Set up Log File status = OFF

This option allows you to assign a peripheral device to accept the contents of the buffer when the buffer fills. You will be asked for a filename. Most users would use either the printer or a disk in order to keep a copy of the buffer contents for later review.

Once the log file has been toggled on, you may turn it off by simply pressing the "S" key from the main menu. If there is data in the buffer, it will be sent to the device you had named as the log file before it closes the file. If you do not desire to have the buffer contents sent to the log file, then clear the buffer before closing the log file. Likewise, if you should happen to quit the program while the log file is open, you will be asked if you wish to save the buffer contents.

The log file uses an append format which allows the computer to add on to the previous file contents, if any. This means that a file on disk will have new contents added to it each time the buffer fills and dumps to the disk. THE LOG FILE WILL NOT OVERWRITE AN EXISTING FILE ON DISK.

Since the file append mode can create a file too large to be loaded by TI-Writer, another way exists in which you can print the file. The E/A module allows you to print a file directly from the disk drive. To use this feature, select the EDIT option from the E/A main menu. Next select the Print feature.

At this point, you will be asked to type in the filename, and then the printer' filename. Upon entering your printer's filename, the file will be read off of the disk and sent to your printer.

Note: M/T sends an XOFF (control S) to the other computer when logging the buffer contents to the assigned peripheral. After the buffer is emptied, the program sends an XON (control Q) to allow the sending computer to continue. This is commonly referred to as XON/XOFF protocol. A BBS that supports this protocol (most of them do) allows M/T to dump the buffer and not miss any incoming data since the BBS will not be sending any during this time.

<V> View buffer contents

This option allows the user to view the contents of the buffer without destroying it. As an added feature, any portion of the buffer can be saved to any peripheral via the screen dump option. This allows the user to read or save a message without having to dump the entire buffer. Scrolling of the displayed text is controlled by the space bar. Press the space bar once to stop the scrolling and again to start it. Pressing FCTN 4 (CLEAR) will abort the viewing option and return to the main menu. To use the screen dump option, allow the screen to scroll to the desired text. Stop the scroll and press the "P" key to start the screen dump. You will be prompted for the output device name. After the screen dump is completed, you will be returned to the viewed screen. You may continue the viewing, printing more screens if necessary, or abort back to the main menu with the FCTN 4 keys.

<B> Buffer capture status = ON

This option allows the user to control the flow of data into the buffer. When the status indicates ON, all data is stored in the buffer. By pressing "B", the buffer will toggle between ON and OFF.

<L> Line feed toggle status = OFF

This option allows the program to provide a linefeed after each received carriage return. Pressing the L key will toggle the option between ON and OFF. These linefeeds are not stored in the buffer, since they are not received data, but rather generated locally by the program. This option is very handy when accessing a BBS that does not supply linefeeds after carriage returns or when receiving an uploaded file from another computer. When toggled ON, it also causes double spacing of the buffer contents when using the view option.

<X> Xmodem File Transfer

This option allows the user to transfer a file to another computer using the xmodem protocol. As with the other xmodem file transfer programs available for the TI computer, this routine provides the user with compatibility with these other programs. It supports both CRC and checksum error detection. The filename for the transfer is assigned after entering this routine. Retry, record, and sector (current and total) indicators are updated constantly (files other than DIS/FIX 128) on the monitor, along with disk access errors. Transfer status is indicated upon completion or aborting of the file transfer. You may use the FCTN 4 keys to abort the transfer.

Note: TO SEND A DISPLAY FIXED 128 (DF-128) FILE IN LARGE BLOCKS RATHER THAN THE USUAL 1 SECTOR AT A TIME, USE THE MXT FUNCTION. THIS WILL ALLOW THE RECEIVING TI COMPUTER TO PROCESS THE FILE AS THOUGH IT WERE A "REGULAR" TI DISK FILE. (SEE COMMENTS ON MXT SENDING AND RECEIVING)

<F> Files (catalog disk) Selection this option from the main menu allows the user to catalog a disk. M/T will prompt you for the drive you wish to display. I use this catalog feature on my Myarc ramdisk. I can not guarantee it will work on other versions of Myarc ramdisks, or any other brand of ramdisk either.

Note: MASS-TRANSFER was originally designed for use with the smartmodems. Although it will work with other modems, one item must be remembered. Some modems (specifically the TI) will not allow the RS232 card to send it data when it is not linked with another modem. During this time, if the program attempts to send the modem data, the computer will appear to lock up. Actually, the program is attempting to send the modem a character, but the modem refuses to accept it. The easiest fix for this problem is to try turning the modem on or off. This will usually prevent/cure this "lock-up" condition.

Your modem may differ in other ways.

I'd like to thank the many users of my old TI-North BBS (312-395-4618) who provided me with many ideas on how to improve my program.

You can call my new BBS at (602) 848-6200.

I'd also like to thank my wife, who I've made a computer widow during the countless hours spent working on this program.

Disk 70. Contents of file MTDOCS4/2

"MASS-TRANSFER" Modem Utility
Version 4.2

If you are using this program, and have not yet paid for it, please send $10 check or money order to the author for his work.

I'm allowing you to try this software before you decide to buy it. As such, I am trusting you to pay for it if you should decide to keep it. If you do not like it, please pass it along to other users, as they may find a need for this program. Good response from you will encourage programmers like myself to provide you with software at a reasonable price and without copy protection. Like they say, you can start paying us now or pay a software dealer 3 times the price for their programs.

STUART OLSON SYSOP: TI-85033 (602) 848-6200
6625 W. COOLIDGE ST.
PHOENIX, AZ. 85033

INFO ON VERSION 4.2

This is the only DOC file for version 4.2. Since the changes in this version are quite simple, in terms of user interfacing, there will not be an update to the complete Mass-Transfer document file, MASSDOCS1 and MASSDOCS2.

This release of M/T has a couple of new items added, and a couple of bugs fixed that I ran across in 4.1. The disk catalog routine in 4.1 identified write protected files incorrectly, and is now corrected.

Thanks to Paul Charlton, I have been able to add the print spooler routines for Axiom, CorComp, and Myarc PIO cards. The instructions following this section describe what bytes can be edited in order to allow use of this version with your particular card. I have released this version with the TI PIO as the print spooler default. Note, someone before you may have modified it, so be sure to check it out.

Since I do not have an Axiom, CorComp, or Myarc card, I have no way of verifying these routines. I hope I have installed them correctly such that they will require no more work.

INTRODUCING YMODEM TRANSFERS FOR THE TI-99/4A

Version 4.2 also includes a new file transfer protocol called YMODEM. No folks, I did not invent this routine. It has been in use for many years on other computer systems. I have adopted it to work on our computer.

First, some background on what Ymodem is, and how it works:

Ymodem is very similar to Xmodem, except that each record sent contains 1024 bytes of data vice the 128 byte blocks used in Xmodem. This increase in record size will give a speed increase in the file transfer, since less time is spent checking small blocks of data and more time is spent sending larger ones.

I have tested Ymodem, both on a local basis, and also using the PC-Pursuit data service available through Telenet. The speed increase when transferring locally is measurable, but not totally earth shattering. However, when transferring files via Telenet, (or other packet switching networks), file transfer time is cut by over 50%. . . yep, that is correct, the transfer takes less than 1/2 the time when compared with Xmodem.

I will not bore you with the details as to how this is possible, but simply state that it is possible because of the way the packet networks process blocks of data. Simply put, Ymodem provides a format better suited to the data handling format available on these networks.

Another item I would like to mention is that this version of Ymodem is not compatible with other computers versions of Ymodem. Before you get all bent out of shape about it, let me explain why.

First off, I did my research when I decided to implement Ymodem on the TI computer. I talked to several professional programmers who make a living writing telecommunication software. Through my discussions with them, I found out that there are "several" versions of Ymodem in use throughout the computer community. Although they are all fundamentally the same, minor differences in each version can cause BIG problems during a file transfer. What is worse is that they are all called Ymodem.

Since our memory space is very limited, I decided it was not possible to attempt to make the TI's use of Ymodem acceptable with all the other mutations floating around, so I kept the same general format, and added a couple of things that greatly improves the ease of use of Ymodem for our computer. Also, since we already have xmodem, it in itself will provide the common link necessary for file transfers between the TI and other brands of computers.

I think most everyone realizes that a TI file header is sent as the first record in an Xmodem transfer. It is also sent as the first record in a Ymodem transfer. However, instead of the first record being 1024 bytes, it has been kept at 128, just like the Xmodem header. After the header has been received, both sending and receiving computers will switch over to the 1024 byte records for the remainder of the transfer.

In order to prevent two users from attempting a transfer using both xmodem and ymodem formats, I have allowed the receiving computer system to automatically detect and adjust itself to the proper format of the sending computer....yep, you guessed it, I am trying to protect you from yourself!

The receiving computer always assumes it will be doing an xmodem transfer. It will look at the received TI file header record, checking the first byte of the record for either a >07 or >08. As usual, if it finds a >07, it continues the transfer in the normal xmodem style. If it finds a >08, it will then shift into the ymodem format, and continue the file transfer.

As you can see, doing it this way makes it very easy for the user, since only the person sending the file need make the decision as to which transfer method to use. The receiving system does not have to worry about it.

To use the Ymodem protocol, just start your transfer the same way as you normally do, with either the X or M keys. If you are sending the file, you will be prompted during the process as to your choice of either X or Ymodem. As I said before, the receiving computer will not have to enter the transfer method to be used. (Ymodem can also be used in MXT transfers.)

OK. . . the last thing I want to inform you about is which method to pick. . . you way, "Heck, I'll go with the faster Ymodem".

That may not always be the proper choice. . . and here is why. When Xmodem encounters an error, it resends the last 128 byte record again. Likewise, when Ymodem encounters and error, it also resends the last record, however, you must remember that the length of the record is 8 times longer. As such, it will take longer to send it. If you are on a noisy phone line, you could easily end up taking longer on a Ymodem file transfer since do to the excessive time required to send the retry blocks. This is something you will have consider when you start and transfer. If the line is usually nice and clean, use Ymodem. If you have had a lot of previous trouble with noise, sticking with xmodem may be the better choice. It is all in your hands!

Last, I would also like to announce that this will most likely be my last update to the Mass-Transfer program. I have been working on it for more than 2 years now, trying to improve it, debug it, include user suggestions, etc. The program is by no means perfect. I am the first to admit that. However, I am not a professional programmer, never tried to be, and never will attempt to be. I have done this terminal emulator program only because I wanted something for my computer which was not available on the open market.

I would like to think that I have provided a valuable program to the TI community. I know that I have supported this program longer than many other fairware authors have theirs, and now I believe it is time for me to let it stand by itself and make room for a bigger and better TE program to come along.

I would like to thank those of you who have taken the time to send financial support for this software. To those of you who have been using it without paying for it. . . well, I guess that is your choice. I have talked with a lot of fairware programmers, and must admit that your attitude of "I'll pay for it later" is causing YOU more harm than you realize. Lets face it, a fairware author can write software for his own use. . . a lot of them do. Your attitude of not paying simply means that the program will stay in the author's disk files, and not yours. If you can afford this for only $10, then I hope you are taking some programming classes since you will need them.

If that last paragraph sounds a bit harsh. . . remember, I have been that author, not you. I was the one who received letters, BBS messages, and phone calls complaining about some feature of Mass-Transfer that the person didn't like. A statement such as "Well, I've been using it for the last year and I don't know why you can't. . .". Gee, maybe that person would have gotten a better response from me had they paid for it instead of just complaining about it.

Folks. . . I am about programmed out, at least for now. I am looking forward to "relaxing" with a computer, running some of those downloads I have never had a chance to really get to use. Like I said earlier, if you have been putting off paying for Mass-Transfer because you expected another version to come out. . . you are correct. Another version has come out, this one. Now you can go ahead and pay for it. . . I doubt there will be another one for you wait for.

Thanks for the support. . . Stu Olson

CUSTOMIZING THE PROGRAM:

Note: It is assumed the user of this info has an understanding of how a sector editor works, especially the one that will be used to alter the MASS file. You should make a copy of the MASS file first, and attempt your changes on that copy. If a mistake is made, you will not destroy your only "working" version of the program. If you do not know how to use a sector editor, I would recommend that you contact a local user group or an individual who is knowledgeable in the use of sector modifications.

Directions:

1. Using a sector editor, locate the first sector of the MASS file. This is the sector in which all modifications will be made.

2. Locate the byte(s) within the sector to change. The following tables gives the location and function of the bytes and defaults that are available.

3. After you have changed the desired bytes, write the edited sector back to disk.

Note: I count bytes assembly style. In other words, the first byte in a sector is 0 and the last byte is 255. I am using decimal numbers for the byte #'s and hex numbers ">" for the default and optional numbers.

Note: Most RS232 cards have 1 PIO port and 2 serial ports. Most users only have 1 RS232 card. If so, the serial ports are referred to as RS232/1 and RS232/2. If two RS232 cards are installed in your system, the second card then becomes RS232/3 and RS232/4, and the second PIO port becomes PIO/2. Since each card has two serial ports, the computer addresses these by using two separate notations; the card address and the port address. The following tables show you which hex codes to edit into the program for setting up both the card and port addresses, depending on your system configuration.

Byte #	*Default*	*Function and Optional values*
10-11	>1300	Selects first or second RS232 CARD.
13	>F4	Selects character and screen colors. The first digit is character color. Second digit is screen color.
15	>40	Selects the RS232 port within the card.
16-17	>04D0	Selects the modem baud rate.
42-43	>03E8	Delay time before auto-repeat for keyboard (>03E8=1000)
44-45	>003C	Delay between characters once keyboard has started auto-repeating. (>003C=60)
48-49	>0000	Print spooler type. >0000=TI card PIO >0001=TI, CorComp, Myarc card serial >0002=AXIOM card PIO >0003=CorComp card PIO >0004=Myarc card PIO
50-51	>1300	Select first or second RS232 card for print spooler.
52-53	>0040	Selects the RS232 serial print spooler port within the card.
54-55	>0034	Selects the baud rate for serial print spooler.
56-57	>8300	Selects number of data bits, parity, and stop bits for serial print spooler

Color codes (in hex)

0	transparent
1	black
2	medium green
3	light green
4	dark blue
5	light blue
6	dark red
7	cyan
8	medium red
9	light red
A	dark yellow
B	light yellow
C	dark green
D	magenta
E	gray
F	white

Baud rate codes (in hex)
>0638 = 110 baud
>04D0 = 300 baud
>0341 = 600 baud
>01A0 = 1200 baud
>00D0 = 2400 baud
>0068 = 4800 baud
>0034 = 9600 baud

RS232 card and port codes (in hex)
>1300 = RS232 card #1
>1500 = RS232 card #2
>40 = RS232/1 or /3
>80 = RS232 /2 or /4

Data bits, parity, stop bits
>8200 = 7N1 >8300 = 8N1
>A200 = 7E1 >A300 = 8E1
>B200 = 7O1 >B300 = 8O1

Disk 71. STAR — Super TI Assembly Routines

Version:
Requires: XB
Author: Michael Riccio
Language: AL
Updated: 02/18/87

A collection of 53 assembly routines for use with Extended BASIC. Routines include disk access, graphics, and many others to make your XB programs much fancier. Great documentation. Source code available from the author.

```
dskdir. v2.0. 12-dec-96
Disk name               = STAR
Sectors total           = 360
Sectors used            = 357
Sectors available       = 1
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 HELP          3 PROGRAM    Y >020 002
002 >003 LOAD         36 PROGRAM    Y >022 035
003 >004 LOAD1        54 PROGRAM    Y >045 053
004 >005 LOAD2         3 PROGRAM    Y >07a 002
005 >006 MULTI         3 PROGRAM    Y >07c 002
006 >007 MUSICTURNS    3 PROGRAM    Y >07e 002
007 >008 SCREEN1      10 PROGRAM    Y >080 009
008 >009 SCREEN2      10 PROGRAM    Y >089 009
009 >00a SCREEN3      10 PROGRAM    Y >092 009
010 >00b SCREEN4      10 PROGRAM    Y >09b 009
011 >00c SCREEN5      10 PROGRAM    Y >0a4 009
012 >00d SCREEN6      10 PROGRAM    Y >0ad 009
013 >00e SCREEN7      10 PROGRAM    Y >0b6 009
014 >00f SCREEN8      10 PROGRAM    Y >0bf 009
015 >010 SCREENS       2 PROGRAM    Y >0c8 001
016 >011 SIDEWAYS      3 PROGRAM    Y >0c9 002
017 >012 SPEECH        7 PROGRAM    Y >0cb 006
018 >013 SPRITES       3 PROGRAM    Y >0d1 002
019 >014 STARDOC1    100 DIS/VAR 80 Y >0d3 099
020 >015 STARDOC2     49 DIS/VAR 80 Y >136 048
021 >016 VDPTEST       7 PROGRAM    Y >166 002 >017 004
022 >01b VDPUTIL2      4 DIS/VAR163 Y >01c 003
```

Disk 71. Contents of file STARDOC1

STAR (Super TI Assembly Routines) Version 1.0 Revision 08/24/86

Instructions

INTRODUCTION:

STAR is a package containing 53 TMS9900 assembly language routines to be used in conjunction with your TI Extended Basic programs. The routines are accessed via the CALL LINK statement. These routines perform tasks that would be difficult or impossible to do with Extended Basic code. Because all of the routines are written in the native tongue of the computer, they execute extremely faster than they would if the code was written in Extended Basic. Also, these routines reside in an area of memory that is not used by your Extended Basic programs, so they do not reduce the maximum size of your programs, and, by performing tasks with one statement instead of several, STAR saves memory. Routines are provided in such areas as character sets, sounds, cassette control, color, screen access, VDP memory access, control, keyboard, character definitions, disk access, string handling, and text mode (40 columns).

FAIRWARE INFORMATION:

STAR was written by Michael Riccio, COM-LINK Enterprises, and was released to the public as Fairware. Fairware is the "try before you buy" concept. This is how it works: You get a copy of the program FREE, then you take it home and use the program. If you find the program useful, you send your payment to the address in the program. The program IS NOT FREE; the author is relying on the honesty of the public to pay for the program.

You may copy the program and distribute it as often as you wish, as long as you follow the following rules:

- NO FEE may be charged for the program. The only exception to this is a copying fee/disk fee, but ONLY IF this fee is charged by a User Group or similar organization as a standard library fee. This fee MAY NOT EXCEED $2. This fee is NOT a substitute for the Fairware donation.

- When the disk is copied, ALL FILES MUST BE INCLUDED. This includes the documentation, as well as any sample programs.

- The files MAY NOT BE ALTERED in any way. This includes the documentation and title screens. However, additional files may be added to the disk, such as expanded documentation or sample programs.

- STAR, its documentation, and sample programs are copyright (C)1986 by Michael Riccio. All rights reserved. Any violation of the above rules is a violation of the Federal Copyright Act.

The requested Fairware donation for this program is $10 to $15. You may send this amount, or any other which you think is fair. Keep in mind that this is not a high price to ask, considering that a comparable package would cost upwards of $50. Send payment to this address:

COM-LINK Enterprises
953 Fillmore Street
Phila, PA 19124

Please include a note explaining that you are paying for STAR.

IMPORTANT: MAKE ALL CHECKS PAYABLE TO MICHAEL RICCIO.

LOADING:

The program loads extremely fast through an Extended Basic LOAD program.

IMPORTANT: For the program to load correctly, files LOAD and LOAD1 must not be altered, and their names must not be changed. Line 1 of the program LOAD2 may be changed (this will be discussed shortly), but the filename must not be altered.

STAR requires a minimum system of the following to operate:

- TI-99/4A console (some routines will not work on a TI-99/4 console)
- Monitor or TV set and RF Modulator (a color set is preferred)
- TI Extended BASIC (STAR has not been tested with third-party versions)
- 32K Memory Expansion (WILL work with Myarc 128K/512K and with RAMDISK)
- TI Disk Controller (has not been tested with CorComp or Myarc controllers)
- One or more disk drives (any configuration)
- Cassette recorder and cable optional for cassette control routines.
- All other hardware is optional, but should work without conflicts.

To load STAR, place the disk in drive 1 (must be in drive 1). If you are not in Extended Basic, you may now enter it, and STAR will load automatically. If you are in Extended Basic, type RUN "DSK1.LOAD" from command mode and press ENTER (you can also have a running program execute this statement.)

After the program loads, you will see a graphic title screen. When you are done looking at it, press any key. You will now see the Fairware information screen. When you are done reading it, press any key. The screen will clear, and you will be placed in command mode of Extended Basic. No program will be in memory, so it will be just like you entered Extended Basic from the title screen. STAR is now loaded into low memory expansion, where it will stay loaded unless you execute CALL INIT or turn off the computer. Entering NEW, BYE, or pressing QUIT (Function-equals) will not affect STAR.

Later on in these instructions, you will learn how to modify the LOAD program so it will do something other than return to command mode (such as run a program that needs the STAR routines.) Now that STAR is loaded, you can learn how to use each of its 53 powerful routines.

PROGRAM ROUTINES:

IMPORTANT: When the syntax is shown for a routine, it will follow these guidelines:

Items in CAPITALS are to be typed exactly as seen. Also, any punctuation except brackets ([]), ellipses (...), or dashes (-) is to be typed as seen.

Items in lower case describe a value, variable, or expression that is to be substituted where indicated.

Items in [brackets] are optional, and may be omitted.

Items followed by ellipses (...) may be repeated.

These are the same guidelines used in the Editor/Assembler manual.

All of STAR's 53 routines are accessed through the CALL LINK statement. This Extended Basic command has the following syntax:

CALL LINK("program-name"[,parameter1 ... ,parameter15])

The program-name is from 1 to 6 characters long, and identifies the routine you are accessing. Parameters 1 through 15 are optional, and vary from routine to routine. The program-name MUST be enclosed in quotation marks, and parameters MUST be separated by commas. The program-name and parameter(s) MUST be enclosed in parentheses. All program names are in capital letters, and two contain the numeral 1 in their names. You must type the program-name exactly as given for the routine to work correctly. Consult the error section of these instructions if you receive an error message when executing a routine.

All of the routines may be executed from a program or from command mode. However, upon return to Extended Basic, the computer resets many video items to the defaults, therefore you may not obtain the expected results. The following routines will not work as expected from command mode: LOW, LGCAPS, FLASH, SCROFF, GETSCR, POKEV, VDPREG, COLORS, MAG, INVERT, ROTATE, FLIP, MIRROR, COPY, TEXT, PRINT, and SCREEN. There is a way to defeat this problem. If you follow the CALL LINK with :: ACCEPT A$, and press FUNCTION-4 when the computer asks for input after the routine executes, the defaults will not be restored. For example, if you enter: CALL LINK("LOW") the lower case letters will be redefined briefly, but then Extended Basic will redefine them. If you enter:

```
CALL LINK("LOW"):: ACCEPT A$
```

the lower case letters will be redefined, then Extended Basic will be waiting for your input. When you press FUNCTION-4, you will be returned to command mode, and Extended Basic will not redefine the letters. NOTE: You do not have to do this if you use the routines inside a program.

CHARACTER SETS:

The four character set routines allow you to use different character sets other than the standard Extended Basic set.

CALL LINK("SMCAPS")

Loads the standard upper case character set (characters 32-95). Same as CALL CHARSET, but does not reset character colors.

CALL LINK("LGCAPS")

Loads the title screen upper case character set (characters 32-95). The title screen characters are larger than the standard character set.

CALL LINK("STDLOW")

Loads the standard lower case character set (characters 96-126). The lower case letters appear as small capitals. CALL CHARSET does not reset these characters.

CALL LINK("LOW")

Loads a true lower case character set (characters 97-122). Also redefines the zero to be slashed, and makes the capital O round.

SOUNDS:

The three sound routines allow you to generate commonly used sounds with one command.

CALL LINK("BEEP")

Emits the valid response tone. It is the input tone used in Extended Basic.

CALL LINK("HONK")

Emits the bad response tone. It is the error tone used in Extended Basic.

CALL LINK("CHIMES")

Plays a chimes sound. It is the same as the one used in the FAST-TERM program.

CASSETTE CONTROL:

The two cassette control routines allow you to turn the motor on or off on the cassette in port 1. This is done through the small black plug. You can control music or other taped material through your programs.

CALL LINK("CS1ON")

Turns on the motor control for cassette port 1.

CALL LINK("CS1OFF")

Turns off the motor control for cassette port 1.

COLORS:

The four color routines allow you to control the colors of different character sets in new ways.

CALL LINK("COLORS",foreground,background)

Changes the colors of all 15 character sets (0-14) to foreground and background. Foreground and background are numbers from 1 to 16, which represent the Extended Basic color code.

CALL LINK("BURST",cycles)

Rapidly changes the screen color through all possible colors, creating an unusual optical illusion. Upon completion, the screen color will be white. Cycles can be a number from 1 to 30000, with 1 being very short, and 30000 being a very long duration.

CALL LINK("FLASH",foreground,background)

Copies the character definitions for the upper case letters (65-90) on to the lower case letters (97-122). In other words, the lower case letters appear as upper case. Any lower case letters on the screen will flash with foreground and background colors. Flashing will continue without further program control until NORMAL is executed (see below.)

CALL LINK("NORMAL")

Stops the flashing caused by the above routine, resets TI's standard lower case letters, and resets the colors to black on transparent.

VDP ACCESS:

The three VDP routines allow you to directly access VDP (Video Display Processor) RAM memory. For details as to what this memory contains, consult the Editor/Assembler and Explorer manuals.

CALL LINK("PEEKV",address,variable1[,variable2 ... ,variable15])

Allows you to read up to 15 consecutive VDP memory addresses. Address should be a number from 0 to 16383.

CALL LINK("POKEV",address,byte1[,byte2 ... byte15])

Allows you to write to up to 15 consecutive VDP memory addresses. Address should be a number from 0 to 16383, and byte1 through byte15 should be values from 0 to 255.

CALL LINK("VDPREG",register,byte)

Writes to a VDP write only register. Register should be a number from 0 to 7, and byte should be a value from 0 to 255.

SCREEN ACCESS:

The seven screen access routines allow you to control what is on the screen.

CALL LINK("SCROFF")

Disables screen display. All that is visible on the screen is the screen color. This is the same as when a key is not pressed for five minutes. After disabling the screen display, you can build your screen, piece by piece, then enable the screen display (see below) to make it all appear at once.

CALL LINK("SCRON")

Enables the screen display.

CALL LINK("ALL",character)

Fills the entire screen with given character code. It is the same as CALL HCHAR(1,1,character,768) but it executes much faster.

CALL LINK("RDSCR",string-array)

Reads in the contents of the screen into string-array, assigning one line to each element. If you are in text mode, 21 elements must be dimensioned, otherwise, 24 elements must be dimensioned. Each array element will contain one line of characters from the screen. Element zero is not used by this routine.

CALL LINK("WTSCR",string-array)

Writes the contents of the string-array to the screen, with each element of the array producing one line on the screen. All elements must have the same length as the lines on the screen. Element zero is not used by this routine. This routine may be used with the one above to save and restore screens using string arrays as temporary storage locations.

CALL LINK("SAVSCR",filename[,screen-color])

Saves the contents of a screen, including color, definitions, sprites, and motion to the device specified by filename. The Myarc 128K/512K RAM-DISK may be used as a valid filename for rapid screen saving. If screen-color is included, the color of the screen (1-16) will also be saved to the device, otherwise it will not be saved.

CALL LINK("GETSCR",filename)

Restores a screen saved by the above routine. If the screen color was saved, it will be restored, otherwise it will remain unaltered. The Myarc 128K/512K RAM-DISK is a valid filename for rapid screen displaying.

CONTROL ROUTINES:

The seven control routines allow you to perform tasks normally not possible in Extended Basic.

CALL LINK("BYE")

Returns to the master title screen. Functions the same as the command BYE, but this routine may be used in a program. Note: This routine does NOT close files like the BYE command does, and all unsaved data will be lost. This routine will not affect STAR (you do not have to re-load.)

CALL LINK("NEW"):: END

Erases the current program in memory. Functions the same as the command NEW, (without clearing the screen) but this routine may be used in a program. *Note*: This routine MUST be followed :: END or unpredictable results may occur. This routine will not affect STAR (you do not have to re-load.)

CALL LINK("NOQUIT")

Disables the QUIT key (function-equals). When QUIT is pressed, nothing will happen.

CALL LINK("QUIT")

Enables the QUIT key (function-equals). When QUIT is pressed, the computer will return to the master title screen. Do not confuse this routine with BYE.

CALL LINK("NOMOVE")

Disables sprite motion. When this routine is executed, all sprites will stop. Any sprites created with motion, or any sprites told to move will not do so. Motion may be enabled with MOVE (see below.)

CALL LINK("MOVE")

Enables sprite motion. All sprites will move when directed to. This routine, along with NOMOVE above, may be used to create large moving objects made up of several sprites. First, execute NOMOVE. Place the sprites on the screen, telling them to move as desired. When all sprites are set, execute MOVE. All sprites will begin to move at the same instant, so no gaps will exist between them.

CALL LINK("ERROR",error-code)

Issues an Extended Basic error or warning message designated by error-code. DO NOT use the codes in the Extended Basic manual, but use a code from this list:

2 Numeric Overflow 24 Line Too Long
3 Syntax Error 25 Can't Continue
4 Illegal After Subprogram 26 Command Illegal In Program
5 Unmatched Quotes 27 Only Legal In Program
6 Name Too Long 28 Bad Argument
7 String-Number Mismatch 29 No Program Present
8 Option Base Error 30 Bad Value
9 Improperly Used Name 31 Incorrect Argument List
10 Image Error 32 Input Error
11 Memory Full 33 Data Error
12 Stack Overflow 34 File Error
13 NEXT Without FOR 35
14 FOR-NEXT Nesting 36 I/O Error (gives meaningless code)
15 Must Be In Subprogram 37 Subprogram Not Found
16 Recursive Subprogram Call 38
17 Missing SUBEND 39 Protection Violation
18 RETURN Without GOSUB 40 Unrecognized Character
19 String Truncated 41 Warning: Numeric Overflow
20 Bad Subscript 42 Warning: String Truncated
21 Speech String Too Long 43 Warning: No Program Present
22 Line Not Found 44 Warning: Input Error
23 Bad Line Number 45 Warning: I/O Error
46 Warning: Line Not Found

KEYBOARD DETECTION:

The five keyboard detection routines allow you to check for key presses that would be difficult or impossible to do from Extended Basic.

CALL LINK("LOCK",variable)

Checks to see if the ALPHA LOCK key is down (on). If it is, then variable will equal one. If it is up (off), then variable will equal zero. This is useful for game programs that require joysticks, because the joysticks will not operate properly if the ALPHA LOCK key is down.

CALL LINK("SHIFT",variable)

Checks to see if either of the two SHIFT keys are down. If it is, then variable will equal one. Otherwise, variable will equal zero.

CALL LINK("CTRL",variable)

Checks to see if the CONTROL key (marked "CTRL") is down. If it is, then variable will equal one. Otherwise, variable will equal zero.

CALL LINK("FCTN",variable)

Checks to see if the FUNCTION key (marked "FCTN") is down. If it is, then variable will equal one. Otherwise, variable will equal zero.

CALL LINK("KEY",valid-keys,key-code)

Scans the keyboard waiting for a key press. Valid-keys is a string containing a list of acceptable key presses. If this is a null string (""), then all keys will be allowed. The key code of the key pressed will be returned in the variable key-code.

CHARACTER DEFINITIONS:

The six character definition routines allow you to modify the definitions of characters in unique ways. All of the character definition routines allow you to access characters 30 through 143. Character 30 is the cursor, and character 31 is the edge character. Normally in Extended Basic, it is not possible to modify these characters, but now you can.

CALL LINK("COPY",character1,character2)

Copies the character definition from character1 over to character2 so they appear identical. This would be the same as: CALL CHARPAT(character1,X$):: CALL CHAR(character2,X$) except this routine is faster and saves memory. To change the definition of the cursor (character 30), define an unused character, then COPY the definition to character 30. Character1 and character2 are character codes from 30 to 143.

CALL LINK("MAG",character1,character2)

Magnifies character1 to twice its size, and defines character2 and the next 3 characters as this magnified version. This comes in handy for using magnification factors 3 and 4 with sprites. Character1 is magnified in the same way that sprites are. Character1 is a character code from 30 to 143. Character2 is a character code from 32 to 140, and it must be evenly divisible by four.

CALL LINK("ROTATE",turns,character[,number])

Rotates one or more character definitions, starting at character, and continuing for the specified number. Turns is a number from 1 to 3 which means the following: 1: Character(s) will be rotated 90 degrees clockwise. 2: Character(s) will be rotated 180 degrees. 3: Character(s) will be rotated 270 degrees clockwise (or 90 degrees counter-clockwise, which is the same.) Character is the starting character code, and is a number from 30 to 143. Number is the number of characters to rotate. If number is omitted, it is assumed to be one, and only character is rotated.

CALL LINK("INVERT",character[,number])

Inverts (changes "on" dots to "off", and "off" dots to "on", making a "negative" image) one or more character definitions, starting at character, and continuing for the specified number. Character is the starting character code from 30 to 143. Number is the number of characters to invert. If number is omitted, it is assumed to be one, and only character is inverted.

CALL LINK("FLIP",character[,number])

Flips one or more character definitions upside down (*Note*: This is NOT the same as rotating them 180 degrees, there is a difference!), starting at character, and continuing for the specified number. Character is the starting character code from 30 to 143. Number is the number of characters to flip. If number is omitted, it is assumed to be one, and only character is flipped.

CALL LINK("MIRROR",character[,number])

Creates a sideways mirror image of one or more character definitions (a sideways flip), starting at character, and continuing for the specified number. Character is the starting character code from 30 to 143. Number is the number of characters to mirror. If number is omitted, it is assumed to be one, and only character is mirrored.

DISK ACCESS:

The six disk access routines allow you to access many of the features of the Disk Manager module from the Extended Basic environment, without losing the program in memory. All of these disk routines EXCEPT CAT will work with the Myarc 128K RAM-DISK if it is emulating a disk drive.

CALL LINK("CAT"[,drive])

Catalogs a disk to the screen. Drive is the drive number from 1 to 5. If drive is omitted, it will catalog the most-recently cataloged drive. The disk will be cataloged to the screen in the same format as Disk Manager uses. The listing can be paused by pressing a key, and it can be resumed by pressing a key again.

IMPORTANT: DO NOT use this routine to catalog the Myarc RAM-DISK. If you do so, unpredictable results will occur, and the computer may lock-up, with the only solution being to shut the computer off (which may erase STAR.) To catalog the RAM-DISK, use the command CALL RDDIR.

Disk 71. Contents of file STARDOC2

CALL LINK("PRO",drive,filename)

Places write protection on filename on the disk in the specified drive. With this "Disk Manager" protection, the file can not be written to, changed, or deleted. Drive is the disk drive number from 1 to 5. Filename is the name of the file to be protected. It may not be longer than 10 letters, and it may not contain any periods or spaces. This protection may be removed with UNPRO (see below.)

CALL LINK("UNPRO",drive,filename)

Removes write protection on filename on the disk in the specified drive. Without this "Disk Manager" protection, the file may be written to, changed, or deleted. Drive and filename are the same as for PRO (see above.)

CALL LINK("RENAME",drive,old-filename,new-filename)

Renames the file called old-filename on the disk in the specified drive as new-filename. Drive is the disk drive number from 1 to 5. Both old-filename and new-filename may be up to 10 letters long, and may not contain any periods or spaces.

CALL LINK("RSEC",drive,sector,string-variable1,string-variable2)

Reads a sector off the disk in the specified drive, and places its contents into the string variables. This routine should be used with extreme caution. Drive is the disk drive number from 1 to 5. Sector is the sector number (sectors are numbered starting with zero.) After execution, String-variable1 will contain the first 128 bytes of the sector, and string-variable2 will contain the last 128 bytes of the sector. For more information on the contents of disk sectors, consult the Advanced Diagnostics manual.

CALL LINK("WSEC",drive,sector,string1,string2)

Writes a sector to the disk in the specified drive. This routine should be used with extreme caution, for ruining one sector can destroy an entire disk! Drive is the disk drive number from 1 to 5. Sector is the sector number to write to (sectors are numbered starting with zero.) String1 contains the first 128 bytes to be placed in the sector, and string2 contains the last 128 bytes. Both string1 and string2 MUST be 128 bytes long.

TEXT MODE:

The four text mode routines allow you to access 40 columns from Extended Basic. In 40 column mode, only lines 1 through 21 may be used, and there are 40 characters per line. Only two colors are allowed in text mode: text color and screen color. When defining characters for text mode, they are only 6X8 instead of 8X8, so the right-most two dots of each dot-row are ignored. Sprites are not available in text mode. The routines SCRON, SCROFF, SAVSCR, GETSCR, RDSCR, and WTSCR will work correctly in text mode. These routines should only be used within a program for proper operation. If your program stops while in text mode (because of an error, break, etc.), you MUST execute CALL LINK("GRAPH") IMMEDIATELY, or the text mode routines may not function as expected.

CALL LINK("TEXT")

Enters text mode (40 columns). The screen is cleared, and the screen colors are set to white on dark blue. If you were already in text mode, the screen is cleared, but your screen colors are retained. Line 22 is always left blank in text mode. Lines 23 and 24 contain valuable Extended Basic system information. This was retained so your program may stop without crashing the computer. You MUST ALWAYS return to graphics (32 column) mode when you are done using text mode.

CALL LINK("GRAPH")

Returns to graphics mode. The screen is cleared, and its color is set to cyan. If you are already in graphics mode, nothing happens. When you return from text mode, all character set colors will be unaltered, but all sprites will be destroyed.

CALL LINK("PRINT",row,column,string)

Displays string on the screen, starting at row and column in text mode. This routine will do nothing in graphics mode. If the string goes off the edge of the screen, it will wrap-around to the next line. If it goes off the bottom of the screen, the screen will scroll up one line. Row is the starting row number from 1 to 21. Column is the starting column number from 1 to 40. String is the string to be printed.

CALL LINK("SCREEN",text-color,screen-color)

Changes the text color and screen color in text mode. This routine will do nothing in graphics mode. Both text-color and screen-color are color codes from 1 to 16.

STRING HANDLING:

The two string handling routines change strings in convenient ways that are cumbersome and slow in Extended Basic.

CALL LINK("CAPS",string-variable)

Changes all the lower case letters in string-variable to upper case letters. All the other characters in the string-variable remain unaffected.

CALL LINK("REVRSE",string-variable)

Reverses the order of the characters in string-variable. For example, if string-variable contained "ABCDE", after execution, it would contain "EDCBA".

These are the 53 routines contained in the STAR package. I hope these descriptions enable you to use them to their fullest potential. If you have any questions concerning the operations of these routines, feel free to write me at the address on the title screen. Be sure to include a self-addressed stamped envelope for a reply. I am sorry to say that I cannot reply to letters which do not contain a self-addressed stamped envelope.

ERROR MESSAGES:

The 53 STAR routines issue 4 error messages, their descriptions follow:

SUBPROGRAM NOT FOUND

This error will be issued if you misspelled the name of the routine (all routine names are in capital letters.) Also, this error will be issued if the package was not loaded correctly (see the beginning of this documentation.)

FILE ERROR

This error is issued by routines which access devices. It is impossible to get Extended Basic to issue an I/O error with the proper error code, so a file error is issued instead. The first digit of the I/O error code does not matter, since no files were opened in the first place, but the second digit may be obtained by executing the following IMMEDIATELY after the error:

```
CALL PEEK(-24576,X):: PRINT X
```

The second digit of the I/O error code will then be printed (for a description of error codes for the disk routines, consult the disk controller manual.) The RSEC and WSEC routines return a zero for the error code, but the error code should be a six, because a sector cannot be read only when the disk is not initialized.

BAD VALUE

The routines issue this error when you supply a value that is beyond the acceptable limits. This error is also given when the string length for WSEC is not 128.

STRING TRUNCATED

This error is issued when a supplied string is beyond the maximum acceptable length. Note that this is an error, not a warning.

For other errors, consult the Extended Basic manual, for they were not issued by STAR. You may issue any Extended Basic error or warning by use of the ERROR routine. You can use an ON ERROR statement to handle these error conditions. Note: CALL ERR will return the Extended Basic error code for an error, not the code listed for the ERROR routine.

MODIFYING THE LOAD PROGRAM:

The files LOAD and LOAD1 should not be altered in any way. Doing so will cause improper loading of the STAR package. Line 1 of the file LOAD2 may be altered to suit your needs. Altering line 0 will cause improper loading of STAR. Line 1 currently reads:

```
1 CALL LINK("NEW"):: END
```

As you know, this will end the program and erase it from memory. You can change this line to do anything you want, such as run another program that needs the STAR routines. You may add lines to the end of this program, but do NOT modify line 0 or resequence the program, or STAR will not load correctly.

IMPORTANT: Programs that use the STAR routines will not run properly unless the STAR package is loaded and in memory.

SOURCE CODE:

The original source code, which is fully documented line-by-line and is 74 pages long, is available from COM-LINK Enterprises. It is of the utmost value to beginners and experts alike, because it provides many programming tips, short-cuts, and methods of interfacing to Extended Basic programs. The source code is NOT available as Fairware, and may only be obtained by purchase.

The source code costs $20 if you send a blank disk, mailer, and postage, or $25 if you do not. Make sure you include a note indicating that you are purchasing the STAR source code, and be sure to include your name and address.

IMPORTANT: The fee for the source code is in ADDITION to the Fairware donation of $15. If you have not paid for STAR, your request will be denied.

MAKE ALL CHECKS PAYABLE TO MICHAEL RICCIO. Send your payment to the address on the title screen.

SAMPLE PROGRAMS:

Several sample programs have been provided on the disk to illustrate the use of the STAR routines.

HELP

Prints out these instructions. Routines used: None.

MULTI

Demonstrates multi-color mode. Routines used: LGCAPS, LOW, SMCAPS, STDLOW, VDPREG, and ALL.

MUSICTURNS

Plays with words to the beat of the song "Upside Down." Routines used: LOW, FLIP, MIRROR, ROTATE and SMCAPS.

SCREENS

Displays 8 screens, waiting for a key press before going on to the next one. The screens are from the files SCREEN1 through SCREEN8. Routines used: KEY and GETSCR.

SIDEWAYS

Prints items sideways on the screen. Routines used: LOW, CHIMES, ROTATE, KEY, and COPY.

SPEECH

A comical demonstration of the computer with speech. Routines used: GETSCR, KEY, and COLORS.

SPRITES

Demonstrates the use of MOVE and NOMOVE. Routines used: MOVE and NOMOVE.

VDPTEST

Demonstrates use of the VDP utilities. Routines used: PEEKV, POKEV, and VDPREG.

VDPUTIL2

A modified version of VDP UTILITY. To use it, MERGE it in with a TI Basic program. The program will now run correctly in Extended Basic, because character sets 15 and 16 will be available.

I hope you enjoy STAR, and I would like to see many Extended Basic programs make use of the routines. I hope you find them helpful. If I get a good response from this release, I might write another package with many more useful routines. Remember, feel free to write to me! Enjoy it!

Michael Riccio
COM-LINK Enterprises

Disk 72. TI-Keys

Version: *Author*:
Requires: *Language*: *Updated*: 12/12/86

A macro key facility for XB which allows you to define almost all CONTROL key combinations to be any string up to 31 characters long. Complete assembly source code and documentation are included. Programs like this are commercially available for about $20. This one is fairware!

```
dskdir. v2.0. 12-dec-96
Disk name               = DM99&GKSTF
Sectors total           = 360
Sectors used            = 186
Sectors available       = 172
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README      21 DIS/VAR 80 Y >022 020
002 >003 KEYLOAD      20 PROGRAM    Y >036 019
003 >005 KEYLOAD/O    48 DIS/FIX 80   >095 047
004 >004 KEYLOAD/S    77 DIS/VAR 80 Y >049 076
005 >006 LOAD/JPH     20 PROGRAM      >0c4 019
```

Disk 72. Contents of file -README

TI-Keys by Wes Johnston

TI-Keys is a program allows the user to define 36 keys so that when typed as control keys, they will display up to 31 characters of text or code. The program is menu driven, disables the quit key, and changes the cursor shape when it is active. Other features include saving the user defined keys to disk, editing the keys, and the ability to turn "off" the program so that true control characters can be typed.

LOADING INSTRUCTIONS

TI-Keys is loadable in two different ways. It is saved in Extended BASIC program format and is named LOAD. RUN this program, and TI-Keys will load quickly. The disadvantage to this method is it will erase any BASIC program already in memory.

The other format is the standard CALL LOAD format. Type CALL INIT :: CALL LOAD("DSKx.MAC")::
CALL LINK("MACRO"). The disadvantage to this method is the time involved loading the program.

USING TI-Keys

Once the program is loaded, BASIC's cursor will be a hollow box. This indicates TI-Keys is loaded and functional. Now press control, and you will notice that the cursor is not blinking. The program is waiting for a key to be pressed. Now press the 'A' key while holding down control. ACCEPT will be printed on the screen. Now release control, and BASIC's cursor will begin blinking again. Make-Keys works with all letter keys A - Z, and the number keys 0 - 9. Any key can be redefined by the user, and saved to disk at any time.

By pressing control =, the menu will appear. The program options are listed on the screen as follows:

1 to EDIT, 2 to SAVE, 3 to LOAD, 4 to TURN OFF KEYS, 5 to RETURN TO BASIC.

1 EDIT
The program will ask `KEY TO CHANGE?'. Press the key you wish to change, and the `PRESENT VALUE' of the key will be displayed. Now TI-Keys asks `CHANGE TO?'. Simply type in the string as it will appear in BASIC, and press enter.

2 SAVE
The program will ask `SAVE FILENAME'. Type in any valid filename except CS1. If an error occurs, FILE ERROR will be displayed. Press a key to get back to the menu. If no error occurs, the menu will be displayed as soon as the file is saved.

3 LOAD
The instructions for load are the same as option.

4 TURN OFF KEYS
Will turn off TI-Keys so that true control characters can be typed, or so that other assembly programs can be loaded with out lock up. **WARNING** If CALL INIT is preformed, and another assembly program is loaded, the computer may lock up. This is prevented by turning TI-Keys OFF, and then typing CALL INIT, and loading the other program.

5 RETURN TO BASIC
Does just what it says. When you are finished with the menu, press 5 to get back into BASIC.

PREDEFINED KEYS

When the program is loaded, the keys have the following text strings stored in them.

A	ACCEPT	S	SAVE "DSK
B	BEEP	T	TAB(
C	CALL	U	U
D	DELETE "DSK	V	VCHAR(
E	END	W	CALL INIT
F	FOR	X	CALL LOAD("DSK
G	GOSUB	Y	CALL LOAD(-
H	HCHAR(	Z	CALL LINK("
I	IF	1	RUN
J	JOYST(	2	\
K	KEY(	3	|
L	LINPUT	4	|
M	MERGE "DSK	5	\ NO PREDEFINED
N	NEXT	6	/ VALUE
O	OPEN	7	|
P	PRINT	8	|
Q	Q	9	|
R	RUN "DSK	0	/

This is a "Fairware Program". Try it! If you like it please send the author your $10.00 thanks. I'm a high school student trying to upgrade to DSDD and other nice things! Thanks!

Wes Johnston
404 Furman Lane
Ladson, SC 29406

Disk 72. Contents of file KEYLOAD/S

```
  +--------------------------+
  |                          |
  |         TI-KEYS          |
  |            BY            |
  |       WES JOHNSTON       |
  |                          |
  |       VERSION  3.0       |
  |      COPYRIGHT 1986      |
  +--------------------------+

   TI-KEYS is to be distributed
  with the FREE WARE concept.
  If you think this program is
  worth $10, please send it to
  me at 404 Furman ln, Ladson
  SC 29456.  Thank you for keeping
  FREE WARE alive!
++++++++++++++++++++++++++++++++
       DEF  MACRO
VMBW   EQU  >2024
VSBW   EQU  >2020
VMBR   EQU  >202C
VSBR   EQU  >2028
KSCAN  EQU  >201C
*
* 26 PRE-DEFINED KEYS. MAX LEN=15 CHRs.
* byte 1=len, bytes 2-15=text
*
MSG1   BYTE 7
       TEXT 'ACCEPT         '
       BYTE 5
       TEXT 'BEEP           '
       BYTE 5
       TEXT 'CALL           '
       BYTE 11
       TEXT 'DELETE "DSK    '
       BYTE 4
       TEXT 'END            '
       BYTE 4
       TEXT 'FOR            '
       BYTE 6
       TEXT 'GOSUB          '
       BYTE 6
       TEXT 'HCHAR(         '
       BYTE 3
       TEXT 'IF             '
       BYTE 6
       TEXT 'JOYST(         '
```

```
       BYTE 4
       TEXT 'KEY(           '
       BYTE 7
       TEXT 'LINPUT         '
       BYTE 10
       TEXT 'MERGE "DSK     '
       BYTE 5
       TEXT 'NEXT           '
       BYTE 5
       TEXT 'OPEN           '
       BYTE 6
       TEXT 'PRINT          '
       BYTE 1
       TEXT 'Q              '
       BYTE 8
       TEXT 'RUN "DSK       '
       BYTE 9
       TEXT 'SAVE "DSK      '
       BYTE 4
       TEXT 'TAB(           '
       BYTE 1
       TEXT 'U              '
       BYTE 6
       TEXT 'VCHAR(         '
       BYTE 9
       TEXT 'CALL INIT      '
       BYTE 14
       TEXT 'CALL LOAD("DSK '
       BYTE 11
       TEXT 'CALL LOAD(-    '
       BYTE 11
       TEXT 'CALL LINK("    '
*
* TEN USER DEFINABLE KEYS. MAX LEN=31 CHRs
* byte 1=len, bytes 2-15=text
*
USRKEY BYTE 0
       TEXT '                                '
       BYTE 4
       TEXT 'RUN                             '
       BYTE 0
       TEXT '                                '
       BYTE 0
       TEXT '                                '
       BYTE 0
       TEXT '                                '
       BYTE 0
       TEXT '                                '
       BYTE 0
       TEXT '                                '
```

```
       BYTE 0
       TEXT '                                '
USRKE1 BYTE 0
       TEXT '                                '
       BYTE 0
       TEXT '                                '
       EVEN
*******************
KEYSTO DATA 0
CURSTO DATA 0
NOKEY  DATA >2000
SCRREG BSS  32
MYREG  BSS  32
LINE   BSS  32
BOTLIN DATA >7F7F,>8080,>8080,>8080
       DATA >8080,>8080,>8080,>8080
       DATA >8080,>8080,>8080,>8080
       DATA >8080,>8080,>8080,>7F7F
KEYCHR DATA >007C,>4444,>4444,>447C

MACRO
       LI   R1,>1000
       MOVB R1,@>83C2    DISABLE QUIT
       LI   R1,PRINT1
       MOV  R1,@>83C4
       BL   @>6A
PRINT1
       LWPI MYREG
*****************************
* check if cntrl is pressed *
*****************************
       LI   R0,1008
       LI   R1,KEYCHR
       LI   R2,8
       BLWP @VMBW
LOOP07 LI   R12,36
       LI   R1,>800
       LDCR R1,4
       LI   R12,6
       STCR R1,8
       CI   R1,>BE00
       JNE  LOOP14
       B    @MENU        CNTRL= pressed
LOOP14 CI   R1,>BF00
       JEQ  LOOP12       not pressed
       B    @RETURN
*********************************
* IF CNTL IS PRESSED, CHECK K-BRD*
*********************************
LOOP12 LI   R1,>0000     scan key board
       MOVB R1,@>8374
```

```
       CLR  @>837C
       BLWP @KSCAN
       CB   @>837C,@NOKEY
       JNE  LOOP07        NO OTHER KEY PRESSED, CHECK IF F-C IS STILL DOWN
       CLR  R1
       MOVB @>8375,R1      put key in r1
       MOVB @>8375,@KEYSTO
       SRL  R1,8           swpb
       CI   R1,158
       JEQ  LOOP10   KEY 8
       CI   R1,159
       JEQ  LOOP11   KEY 9
       CI   R1,176
       JL   LOOP08   MUST BE A LETTER
       CI   R1,184
       JL   LOOP09   MUST BE A # KEY
LOOP08
       CI   R1,129       less than 'A'?
       JL   LOOP07       yes
       CI   R1,154       gt'er than 'Z'?
       JH   LOOP07       yes
       AI   R1,-129      r1=r1-129
       SLA  R1,4         mpy x 16
       LI   R3,MSG1
       A    R1,R3        add for text location
       JMP  INT01
LOOP09 AI   R1,-176
       SLA  R1,5
       LI   R3,USRKEY
       A    R1,R3
       JMP  INT01
LOOP10 LI   R3,USRKE1
       JMP  INT01
LOOP11 LI   R3,USRKE1+32
       JMP  INT01
INT01
       MOVB @>8362,R0    GET CURSOR POS.
       SRL  R0,8
       MOVB @>8361,R0    COMPUTE SCREEN LOCATION.
       CLR  R1
       BLWP @VSBR        SEE IF CURSOR IS ON OR OFF.
       MOV  R1,@CURSTO   STORE IT FOR LATER.
       MOVB *R3,R2
       INC  R3
       SRL  R2,8
       MOV  R2,R2
       JEQ  RETURN
LOOP   MOVB *R3,R1       COPY
       AI   R1,>6000     ADD FOR BASIC OFFSET.
       BLWP @VSBW        PUT ON SCREEN
```

```
INT05  INC  R0           INC SCR LOC
       CI   R0,>2FE      CHECK IF
       JNE  INT06         OFF SCR.
       BL   @SCROLL
       JMP  INT04
INT06  BLWP @VSBR
       CI   R1,>7F00
       JEQ  INT05
INT04
       INC  R3           INC TEXT POINTER.
       DEC  R2           DEC LEN
       JNE  LOOP         NOT DONE
       LI   R1,>7E00
       C    R1,@CURSTO   WAS CURSOR ON?
       JNE  INT02        NO
       LI   R1,>8000     YES
       MOVB R1,@>8301    PUT SPACE IN CHAR BUF.
       LI   R1,>7E00
       BLWP @VSBW        PUT CURSOR ON SCREEN.
       JMP  INT03        RETURN
INT02  LI   R1,>7E00     PUT CURSOR IN CHAR BUF.
       MOVB R1,@>8301
       LI   R1,>8000
       BLWP @VSBW        PUT SPACE ON SCREEN.
**************************
* ADJUST BASIC'S POINTERS *
**************************
INT03
       MOV  R0,@>832A   END OF TEXT.
       SLA  R0,8
       MOVB R0,@>8362   CURSOR POSITION.
KLOOP1 BLWP @KSCAN
       CB   @>8375,@KEYSTO
       JEQ  KLOOP1
       LI   R1,>0000
       MOVB R1,@>8375
*******************
* RETURN TO BASIC *
*******************
RETURN
       LWPI >83E0        REAL KSCNA ADDR.
       RT                GOTO REAL K-SCAN ROUTINE.
*******************
* SCROLL           *
*******************
SCROLL
       LWPI SCRREG
       LI   R0,32
       LI   R1,LINE
       LI   R2,32
SCRL1  BLWP @VMBR
```

```
       AI   R0,-32
       BLWP @VMBW
       AI   R0,64
       CI   R0,768
       JL   SCRL1
       LI   R0,768-32
       LI   R1,BOTLIN
       LI   R2,32
SCRL2  BLWP @VMBW
       MOV  @>8320,R0
       AI   R0,-32
       MOV  R0,@>8320
       LWPI MYREG
       LI   R0,>2E2
       RT
TITLE  TEXT '  +--------------------------+  ' 2
       TEXT '  |                          |  ' 3
       TEXT '  |         TI-KEYS          |  ' 4
       TEXT '  |            BY            |  ' 5
       TEXT '  |       WES JOHNSTON       |  ' 6
       TEXT '  |                          |  ' 7
       TEXT '  |       VERSION  3.0       |  ' 8
       TEXT '  |      COPYRIGHT 1986      |  ' 9
       TEXT '  +--------------------------+  ' 10
       TEXT '  PRESS                        ' 1
       TEXT '        1 TO EDIT KEYS         ' 2
       TEXT '        2 TO SAVE KEYS         ' 3
       TEXT '        3 TO LOAD KEYS         ' 4
       TEXT '        4 TO TURN OFF MACRO    ' 5
       TEXT '        5 TO RETURN TO BASIC   ' 6
ETEXT1 TEXT 'KEY TO CHANGE?'
ETEXT2 TEXT 'PRESENT VALUE:'
ETEXT3 TEXT 'CHANGE TO:'
CURCHR DATA >007E,>7E7E,>7E7E,>7E7E
MLIST  DATA MEDIT,MSAVE,MLOAD,MCLAR,MRTRN
ALPFLG DATA 0
SCRBUF BSS  448
       EVEN
MENU
       LI   R0,320
       LI   R1,SCRBUF
       LI   R2,448
       BLWP @VMBR
MENU2  LI   R0,0
       BL   @CLEAR
       LI   R0,32
       LI   R2,TITLE
       LI   R3,480
       BL   @PBASIC      DISPLAY ALL OPTIONS.
MENU1  CLR  @>837C
```

```
       BLWP @KSCAN
       CB   @>837C,@NOKEY
       JNE  MENU1
       CLR  R1
       MOVB @>8375,R1
       CI   R1,>3100
       JL   MENU1
       CI   R1,>3500
       JH   MENU1
       AI   R1,->3100
       SRL  R1,8
       SLA  R1,1          MPY x 2
       LI   R2,MLIST
       A    R1,R2         ADD R1 TO LIST OF DATA
       MOV  *R2,R3
       BL   *R3
MEDIT  LI   R0,320
       BL   @CLEAR
       LI   R0,354
       LI   R2,ETEXT1
       LI   R3,14
       BL   @PBASIC       PRINT 'KEY TO CHANGE'
MEDIT1 CLR  @>837C
       BLWP @KSCAN
       CB   @>837C,@NOKEY
       JNE  MEDIT1
       CLR  R1
       MOVB @>8375,R1
       CI   R1,>3000
       JL   MEDIT1
       CI   R1,>3900
       JH   MEDIT2
       CLR  @ALPFLG       FLAG=0=NUMBER
       AI   R1,>6000
       BLWP @VSBW
       AI   R1,->9000
       SRL  R1,3
       LI   R2,USRKEY
       JMP  MEDIT3
MEDIT2 CI   R1,>4100      CHECK FOR A
       JL   MEDIT1        LESS THAN A
       CI   R1,>5A00      CHECK FOR Z
       JH   MEDIT1        HIGHER THAN Z
       INC  @ALPFLG       FLAG=1=LETTER
       AI   R1,>6000
       BLWP @VSBW
       AI   R1,->A100
       SRL  R1,4
       LI   R2,MSG1
MEDIT3 A    R1,R2
       MOV  R2,@BUFFLO
```

```
       MOVB *R2+,R3
       SRL  R3,8
       LI   R0,481
       BL   @PBASIC
       LI   R0,418
       LI   R2,ETEXT2
       LI   R3,14
       BL   @PBASIC
       LI   R0,546
       LI   R2,ETEXT3
       LI   R3,10
       BL   @PBASIC
       MOV  @ALPFLG,@ALPFLG
       JNE  MEDIT4       IF FLAG=1, LETTER
       LI   R0,609+32
       LI   R1,>8D00
       LI   R2,31
       BL   @HCHAR
       BL   @INPUT
       DATA 609
       DATA 640
       DATA >207E
       B    @MENU2
MEDIT4 LI   R0,609+32
       LI   R1,>8D00
       LI   R2,15
       BL   @HCHAR
       BL   @INPUT
       DATA 609
       DATA 624
       DATA >207E
       B    @MENU2
MCLAR  CLR  @>83C4
       LI   R0,1008
       LI   R1,CURCHR
       LI   R2,8
       BLWP @VMBW
       B    @MENU2
MRTRN  LI   R0,320
       LI   R1,SCRBUF
       LI   R2,448
       BLWP @VMBW
       CLR  @>8375
       LWPI >83E0
       B    @>6A
HCHAR  BLWP @VSBW
       INC  R0
       DEC  R2
       JNE  HCHAR
       RT
```

```
CLEAR
       MOV  R0,R2
       AI   R2,-768
       NEG  R2
       LI   R1,>8000
CLEAR1 BLWP @VSBW
       INC  R0
       DEC  R2
       JNE  CLEAR1
       RT
PBASIC
       MOV  R3,R3
       JEQ  PBAS1
PBAS2  MOVB *R2+,R1
       AI   R1,>6000
       BLWP @VSBW
       INC  R0
       DEC  R3
       JNE  PBAS2
PBAS1  RT
*
* VAR'S FOR INPUT SUB-PRGM
*
LOKEY  BYTE 32
HIKEY  BYTE 127
ENTR   BYTE 13
STARLO DATA 0   TEMP STORAGE FOR SCREEN STRT LOC.
STOPLO DATA 0   "                    " STOP LOC.
BUFFLO DATA 0   "              " WHERE BUFFER IS IN CPU.
SAVERT DATA 0   "              " WHERE TO RETURN TO.
BSPC   BYTE 8
       EVEN
*
* BEGIN OF INPUT SUB-PRGM.
*
* STORE DE-FAULT DATA
*
INPUT
       MOV  *R11,@STARLO
       INCT R11
       MOV  *R11,@STOPLO
       INCT R11
       MOV  *R11,@LOKEY
       INCT R11
       MOV  @BUFFLO,R3
       INC  R3
       MOV  R11,@SAVERT
       MOV  @STARLO,R0
       CLR  R2
       MOVB R2,@>8374
KEYONE
```

```
       LI   R4,>40
       LI   R1,>7E00
       LI   R5,1
       BLWP @VSBW
*
* CHECK KEY-BRD FOR K PRESS.
*
KEYTWO
       DEC  R4
       JNE  KEYTHR       NOT =0 YET
       MOV  R5,R5        FLAG SET?
       JNE  KEYFOU       YES
       LI   R1,>8000     CUR OFF
       LI   R4,>50
       INC  R5           SET FLAG
       JMP  KEYFIV
KEYFOU LI   R1,>7E00     CUR ON
       LI   R4,>50
       CLR  R5           CLR FLAG
KEYFIV BLWP @VSBW
KEYTHR CLR  @>837C
       BLWP @KSCAN
       CB   @>837C,@NOKEY
       JNE  KEYTWO
       CB   @>8375,@BSPC
       JEQ  BACK
       CB   @>8375,@ENTR
       JEQ  ENTER
       C    R0,@STOPLO
       JEQ  KEYTWO
       CB   @>8375,@LOKEY
       JL   KEYTWO
       CB   @>8375,@HIKEY
       JH   KEYTWO
       MOVB @>8375,R1
       AI   R1,>6000
       BLWP @VSBW
       MOVB @>8375,*R3
       INC  R0     INC SCREEN LOC.
       INC  R2     INC LEN OF STRING.
       INC  R3     INC BUFFER LOC.
       JMP  KEYONE
*
* BACK SPACE.
*
BACK   C    R0,@STARLO
       JEQ  KEYTWO
       LI   R1,>8000
       BLWP @VSBW
       DEC  R0     DEC SCREEN LOC.
```

```
       DEC  R2     DEC LEN OF STRING.
       DEC  R3     DEC BUFFER LOC.
       JMP  KEYONE
*
* ENTER PRESSED. ADJ. NESS. DATA & PUT LEN OF STR IN BYTE 0 OF 'BUFFER'.
*
ENTER
       MOV  R2,R2
       JEQ  KEYTWO     IF NULL STRING, GOTO KEYTWO
       LI   R1,>8000
       BLWP @VSBW
       MOV  @BUFFLO,R1
       SWPB R2
       MOVB R2,*R1
       MOV  @SAVERT,R11
       RT
FTEXT1 TEXT 'SAVE FILENAME?'
FTEXT2 TEXT 'LOAD FILENAME?'
FTEXT3 TEXT 'FILE ERROR!!'
MSAVE  LI   R0,>0600
       MOVB R0,@PAB
       CLR  @FLAG
       LI   R4,FTEXT1
       B    @MFILE
MLOAD  LI   R0,>0500
       MOVB R0,@PAB
       INC  @FLAG
       LI   R4,FTEXT2
       B    @MFILE
PAB    DATA >0500,>1000,0,736    FILE CHAR'S FOR PROGRAM FORMAT.
       BYTE 0                    REQUIRED DO NOT MOVE!
FNAME  BSS  34          DO NOT PUT ANYTHING BETWEEN HERE AND FILE.
       EVEN
VDPBUF BSS  736
FACBUF BSS  >24
REGBUF BSS  >6
FLAG   DATA 0
       EVEN
MFILE
       LI   R0,320
       BL   @CLEAR
       LI   R0,354
       MOV  R4,R2
       LI   R3,14
       BL   @PBASIC
       LI   R1,FNAME
       MOV  R1,@BUFFLO
       BL   @INPUT       GOSUB TO INPUT SUBPROGRAM
       DATA >16C+4       FIRST POSITION TO ACCEPT AT.
       DATA >17B+4       MAX POSITION TO ACCEPT AT.
       TEXT '!~'         ACCEPT ALL ASCII CHAR'S FROM ! TO ~.
```

```
MFILE1
       LI   R0,>FE0
       LI   R1,VDPBUF
       LI   R2,736
       BLWP @VMBR         SAVE VDP
       LI   R0,>FE0
       LI   R1,PAB
       LI   R2,32
       BLWP @VMBW         PUT FILE IN VDP.
       LI   R0,>834A
       LI   R1,FACBUF
       LI   R2,>24
       BL   @MEMMOV
       LI   R0,>83DA
       LI   R1,REGBUF
       LI   R2,>6
       BL   @MEMMOV
       MOV  @FLAG,@FLAG
       JNE  MFILE3
       LI   R0,>1000
       LI   R1,MSG1
       LI   R2,736
       BLWP @VMBW         IF SAVE, PUT KEYS IN VDP
MFILE3 LI   R0,>FE9
       MOV  R0,@>8356     SET UP TO READ FROM IT.
       BLWP @DSRLNK       READ 8K INTO TOP OF VDP.
       DATA 8             REQUIRED.
       JEQ  MERROR
       MOV  @FLAG,@FLAG
       JEQ  MFILE2
       LI   R0,>1000
       LI   R1,MSG1
       LI   R2,736
       BLWP @VMBR         READ KEYS FROM VDP TO A LOCATION IN CPU.
MFILE2
       LI   R0,>FE0
       LI   R1,VDPBUF
       LI   R2,736
       BLWP @VMBW
       LI   R0,FACBUF
       LI   R1,>834A
       LI   R2,>24
       BL   @MEMMOV
       LI   R0,REGBUF
       LI   R1,>83DA
       LI   R2,>6
       BL   @MEMMOV
       B    @MENU2
MERROR LI   R0,16*32+9
       LI   R2,FTEXT3
```

```
       LI   R3,12
       BL   @PBASIC
MERR1  CLR  @>837C
       BLWP @KSCAN
       CB   @>837C,@NOKEY
       JNE  MERR1
       B    @MFILE2
MEMMOV MOVB *R0+,*R1+
       DEC  R2
       JNE  MEMMOV
       RT
************************
* SUBROUTINE DSRLNK     *
* BLWP @DSRLNK.         *
* ENTER WITH POINTER TO *
* NAME LENGTH IN >8356  *
************************
MASK   BYTE >20
PERIOD BYTE >2E
VALID  BYTE >AA
REG1   BSS  >20
REG2   BSS  >20
DSRBUF BSS  8
DSRLNK DATA REG2,DSRL1

DSRL1  MOV  *R14+,R5     GET DATA TO R5 & INC PMG CNTR
       SZCB @MASK,R15    RESET EQUAL BIT
       MOV  @>8356,R0    NAME LENGTH BYTE TO R0
       MOV  R0,R9        THEN TO R9
       AI   R9,>FFF8     R9 POINTS TO BEGINNING OF PAB
       BLWP @VSBR        GET LENGTH BYTE INTO R1
       MOVB R1,R3        MOVE TO R3
       SRL  R3,8         ALIGN TO RIGHT BYTE
       SETO R4           R4 = >FFFF
       LI   R2,DSRBUF    R2 POINTS TO BUFFER
DSRL2  INC  R0           R0=R0+1
       INC  R4           R4=R4+1
       C    R4,R3        DONE YET?
       JEQ  DSRL3        YES, GO ON
       BLWP @VSBR        NO, GET NEXT BYTE
       MOVB R1,*R2+      AND PUT IT IN BUFFER
       CB   R1,@PERIOD   WAS IT A PERIOD?
       JNE  DSRL2        NO, GET NEXT BYTE
DSRL3  MOV  R4,R4        R4=0?
       JEQ  DSRL10       YES, RETURN W/ERROR
       CI   R4,>0007     R4 GREATER THAN 7?
       JGT  DSRL10       YES, RETURN W/ERROR
       CLR  @>83D0       >83D0=0
       MOV  R4,@>8354    NAME LENGTH TO >8354
       INC  R4           R4=R4+1
       A    R4,@>8356    >8356 POINTS TO DATA AFTER NAME
```

```
       LWPI >83E0        USE GPL REGS
       CLR  R1           R1=0
       LI   R12,>0F00    R12 = BEGINNING OF DSR BLOCKS
DSRL4  MOV  R12,R12      CHECK FOR 0
       JEQ  DSRL04       YES, GO ON
       SBZ  >00          TURN CARD OFF
DSRL04 AI   R12,>0100    POINT TO NEXT DSR BLOCK
       CLR  @>83D0       >83D0=0
       CI   R12,>2000    DONE?
       JEQ  DSRL9        YES, INCORRECT DEVICE NAME
       MOV  R12,@>83D0   STORE BLOCK AT >83D0
       SBO  >00          TURN CARD ON
       LI   R2,>4000     R2 POINTS TO BEGINNING OF DSR ROM
       CB   *R2,@VALID   CHECK FOR VALID CARD
       JNE  DSRL4        NOT ONE HERE, TRY NEXT BLOCK
       AI   R2,8         R2 POINTS TO DSR ENTRY FIELD
       JMP  DSRL6        SKIP OVER THE FIRST TIME THROUGH
DSRL5  MOV  @>83D2,R2    NEXT DEVICE FIELD TO R2
       SBO  >00          TURN CARD ON
DSRL6  MOV  *R2,R2       ENTRY FIELD POINTER TO R2
       JEQ  DSRL4        TRY NEXT BLOCK IF THIS IS LAST ONE
       MOV  R2,@>83D2    SAVE POINTER TO NEXT ENTRY
       INCT R2           POINT TO ENTRY ADR FOR THIS DEVICE
       MOV  *R2+,R9      MOVE TO R9
       MOVB @>8355,R5    LENGTH BYTE TO R5
       JEQ  DSRL8        GO IF ZERO
       CB   R5,*R2+      LENGTH THE SAME?
       JNE  DSRL5        NO, TRY NEXT DEVICE
       SRL  R5,8         YES, ALIGN COUNT TO RIGHT BYTE
       LI   R6,DSRBUF    R6 POINTS TO NAME BUFFER
DSRL7  CB   *R6+,*R2+    CHECK FOR MATCH & INC POINTERS
       JNE  DSRL5        NO MATCH, TRY NEXT DEVICE
       DEC  R5           COUNT -1
       JNE  DSRL7        TRY NEXT BYTE
DSRL8  INC  R1           R1=R1+1
       BL   *R9          BRANCH TO DSR ROM
       JMP  DSRL5        THIS INSTRUCTION IS SKIPPED!
       SBZ  >00          TURN CARD OFF
       LWPI REG2         USE PROGRAM REGISTERS
       MOV  R9,R0        R0 POINTS TO FLAG/STATUS BYTE
       BLWP @VSBR        GET STATUS
       SRL  R1,13        ALIGN ERROR BITS
       JNE  DSRL11       GO IF ERROR
       RTWP              RETURN NO ERROR
DSRL9  LWPI REG2         USE PROGRAM REGISTERS
DSRL10 CLR  R1           R1=0
DSRL11 SWPB R1           ERROR CODE TO RIGHT BYTE
       MOVB R1,*R13      PUT IN R0 OF CALLING PMG
       SOCB @MASK,R15    SET ERROR BIT
       RTWP              RETURN W/ERROR
```

```
END
```

Disk 73. TI-Sings

Version: *Author*:
Requires: *Language*: *Updated*: 12/12/86

A complete system for creating and editing songs using the TE-II cartridge and a speech synthesizer. Many examples are included with an excellent tutorial on how it all works. Fairware from Trio+ Software. Requires TE-II cartridge and Speech Synthesizer.

```
dskdir. v2.0. 12-dec-96
Disk name               = TI-SINGS
Sectors total           = 360
Sectors used            = 302
Sectors available       = 56
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 <CAROLINE    14 INT/VAR254   >022 013
002 >003 <DAISY        4 INT/VAR254   >02f 003
003 >004 <HERECOMES    8 INT/VAR254   >032 007
004 >005 <OCTOPUS     11 INT/VAR254   >039 010
005 >006 <SCALE        2 INT/VAR254   >043 001
006 >007 <SOVAIN      19 INT/VAR254   >044 018
007 >008 <STANDING    11 INT/VAR254   >056 010
008 >009 CHELP        37 PROGRAM      >060 036
009 >00a CREATESONG   22 PROGRAM      >084 021
010 >00b HELP         18 PROGRAM      >099 017
011 >00c HINTS        17 PROGRAM      >0aa 016
012 >00d REFINESONG   23 PROGRAM      >0ba 022
013 >00e RHELP        42 PROGRAM      >0d0 041
014 >00f SPEECH       69 DIS/VAR 80   >0f9 063 >13c 005
015 >010 TI-SINGS      5 INT/VAR 80   >138 004
```

Disk 73. Contents of file SPEECH

The double & signs in the program lines should be left as they are. Since the TI-Writer sees the & sign as a formatter command, the only way to get one to print correctly is to use two of them together.

!I AM COMPUTER, HEAR ME ROAR!"

By Barb Berg

If you have the Speech Synthesizer and a speech-accessible module, such as Extended BASIC, Terminal Emulator II, or a game such as Parsec or Alpiner, you have probably heard your computer talk to you at some time or another. Writing a program that includes speech is really not that hard, as you may have discovered. The TE II module allows text-to-speech, which makes it quite easy to have words or phrases "spoken" by the computer, and XBASIC allows allophones and uses the resident vocabulary of 373 words and phrases in the speech synthesizer.

But if you HAVE used the TEII for text-to-speech, you must have discovered that some words are pronounced incorrectly. Some words in our English language do not sound like they are spelled. Can you imagine the trouble a foreigner must have, trying to learn our language with words like tough and through? Both have the "ough" at the end of the word, but definitely have different sounds.

This is similar to what happens with the speech synthesizer. While a lot of the common sounds (and many of the uncommon ones) are translated correctly, a few words manage to sound like they belong in some other language. I noticed this most when I wrote a simple spelling program for my kids to practice their weekly spelling words on. Where they go to school, they have to learn how to spell 75 words every week. My 6th grader just brought home a list with words I'd never even heard of, and most of the rest were words I didn't know until high school!

They have learned through the years that it helps to sound out a difficult word in order to pronounce it properly (or at least closely!). In the spelling program, I made allowances for words that didn't sound like they were spelled. I used two different variable arrays, one for the correct spelling of the word, and one for the correct pronunciation. If the pronunciation of the word was all right when entered as correctly spelled, it automatically went into the pronouncing array. If not, one could enter the word phonetically, or the way the word sounded. This spelling would go into the pronounce array only.

However, if you write a program which uses a lot of speech or uses words like these, you may decide that even those words that come out sounding ok are still not quite what you had in mind for that particular program. The TEII module has a very nice system for that purpose, in that the text is translated into allophones according to previously programmed speech rules, and these allophone strings are then changed into what the manual calls Linear Predictive Coding (LPC) strings. It is the LPC string that the speech synthesizer "speaks".

Now, I don't know about anyone else, but the default "voice" resident in the synthesizer sounds an awful lot like Marvin the Paranoid Android from the "Hitchhiker's Guide to the Galaxy" series shown on IPT a few years ago. This robot was a terribly depressed individual, and his voice sounded like he was ready to "pull the plug" on himself any minute. The speech rules in the TEII module include sentence intonations, but like Marvin, they make the voice sound like your computer just died and it's in mourning. You can change the pitch and slope, but even then it can sound like either a soprano Marvin (you can do your own jokes here, folks) or like him with a bass voice.

So how can you make the voice more interesting? Well, you can include inflection symbols, which tell the computer that you want a stress on this word or that syllable, but that doesn't always help as much as I'd like. Or, you can vary the pitch and slope slightly within the sentence or phrase itself. I don't recommend the latter, as it makes the speech sound very choppy, like Mel Tillis when he tries to say something and can't get it out right away.

We have a program from the UG library that has a little robot character singing "Old MacDonald". In this program, the pitch and slope are altered according to the notes that the words are supposed to be sung on, and if you've seen and heard it you probably have noticed how choppy it sounds. Good effort, but frustrating to listen to, especially if the kids decide they want to sing along.

Well, where does that leave you? It doesn't sound like the TEII and speech can be programmed to sound very realistically, does it? Ah, but wait a minute! There are also allophones. But what are they, and how do you use them?

Allophones are simply the individual sounds of speech. Remember when you had to learn "long" and "short" vowel sounds? Well, they don't call them that anymore, I guess, but at any rate, each letter in the alphabet has one or more sounds, depending on what letters they are used in combination with.

For example, notice the difference in the combination "in" in the following words: in, insane, think, thing. In the word "insane", the i is spoken for a shorter length of time than the i in "in". The n is pronounced differently in the last two words than in the first two. We speak these words without giving thought to the length of time we hold the i, and "ing" and "ink" are second nature. And we may use a different inflection in our voice to set the mood of what we say. But the speech synthesizer can't do all of that by itself. If you want to say INsane instead of inSANE, you just do it! But you have to tell the speech synthesizer where to put the stress point in the word so it can say it the way you want.

But it looks so time consuming to program a sentence with allophones. Right? The following program is on page 40 of the TEII manual:

```
100 OPEN #1:"SPEECH",OUTPUT
110 OPEN #2:"ALPHON",INTERNAL
120 INPUT "PHRASE ":A$
130 IF A$="" THEN 120
140 PRINT #1:A$
150 INPUT #2:B$
160 Z=LEN(B$)
```

```
170 FOR R=4 TO Z
180 PRINT ASC(SEG$(B$,R,1))
190 NEXT R
200 GOTO 120
```

What this program does is a great time saver. File #1 opens a channel to the speech file, which speaks the words you enter for text-to-speech. File #2 opens the alphon file, which breaks the words down into allophones. When you run the program, you are asked for a phrase. Whatever you enter here is first spoken by the speech file and then input into the alphon file. The standard inflections and stresses make up the first three CHR$'s of the phrase. These include the sentence break code 250 and the number of secondary stress points before and after the primary stress point. Since all we want are the allophones for the words in the phrase, we start at the fourth CHR$ in the phrase and the computer displays the allophones for the words used. Now you can see what allophones are used in each word by comparing the screen display with the chart on page 41.

You can then take your word or phrase as broken down and do a variety of things to it. Add pitch and slope, change the vowel sound or the length of time it is held, alter the consonant sounds, whatever you want to do. When you make up the speech string from allophones to add to your program, the allophone numbers become CHR$ numbers. For example, take the word "the". The allophones used by the TEII for "the" are 95 and 18. CHR$(95) is the hard "th" sound to be used at the beginning of a word and CHR$(18) is a very short "uh" sound. Change CHR$(18) to CHR$(69) to have the "uh" sound held longer and for a more distinct sound. Now you can put "the" into a string variable:

```
A$=CHR$(95)& CHR$(69)
```

Want to add pitch and slope? CHR$(251) is the slope indicator and CHR$(252) sets up the pitch. The default values for pitch and slope are 43 and 128, respectively. We can add them to our string like this:

```
100 THE$=CHR$(95)& CHR$(69)
110 PITCH$=CHR$(252)& CHR$(43)
120 SLOPE$=CHR$(251)& CHR$(128)
130 A$=PITCH$ & SLOPE$& THE$
```

If you don't understand the difference changing slope makes, or the formulae on pages 34-35 of the manual, then you should experiment with it a little. I have found that the lower the number the smoother the slope. If the slope is a high number, you get a rougher sounding voice. That isn't the right word to express it, but you really have to hear it to understand.

As for using the formulae to determine slope, first use the 32 * 10% of the pitch. If the pitch is 35, then 32*3=96, and that is what the manual says should be the best sounding. Now check this in the other two formulae: first, yyy<[xx-1]*16. With our values replacing the variables, the result is this: 96<[35-1]*16 or 96<34*16 or 96<544. Definitely true. In the other, yyy<[63-xx]*16 becomes 96<63-35*16, or 96<28*16 or 96<448. You can see, this gives you a lot of room to work with when you use a low pitch. (The lower the number, the higher the pitch.) Just keep the slope under 255. The format in text-to-speech format is: PRINT #1:"//xx yyy" where file #1 is opened to the speech file, xx=pitch and yyy=slope.

Need stress points in your phrase? Use CHR$(253) for a primary stress in which the sound must rise and CHR$(254) for one in which the sound lowers slightly. CHR$(249) indicates a secondary stress point, which puts less emphasis on a syllable than the primary stress point. CHR$(250) should be used at the beginning of a sentence if stress points are used as this tells the computer how many secondary stresses there are both before and after the primary stress point. Can't understand how? Start your string with CHR$(250). Count the number of stress 2's come before your stress 1 (A). Now count the number of 2's after the 1 (B). Then use this form:

```
CHR$(250) & CHR$(A) & CHR$(B)
```

Continue to build your string from there. If the entire string is supposed to rise or fall, use either CHR$(254) or CHR$(255) after the CHR$(250). Then add a CHR$(A) where A = the number of vowels in the phrase. As an example, enter the following:

```
100 OPEN #1:"ALPHON",INTERNAL
110 A$=CHR$(250)&CHR$(2)&CHR$(1)&CHR$(252)&CHR$(46)&CHR$(249)
120 C$=CHR$(117)&CHR$(34)&CHR$(249)&CHR$(74)&CHR$(67)&CHR$(127)&CHR$(254)
&CHR$(86)&CHR$(50)&CHR$(249)&CHR$(87)
130 C$=C$&CHR$(82)&CHR$(69)&CHR$(127)&CHR$(127)
140 B$=CHR$(250)&CHR$(3)&CHR$(0)&CHR$(252)&CHR$(46)&CHR$(249)
150 D$=CHR$(126)&CHR$(118)&CHR$(51)&CHR$(249)&CHR$(50)&CHR$(126)&CHR$
(249)&CHR$(85)&CHR$(46)&CHR$(127)
160 D$=D$&CHR$(254)&CHR$(73)&CHR$(69)&CHR$(98)
170 E$=A$&C$
180 F$=B$&D$
190 PRINT #1:E$&F$
200 GOTO 110
```

Believe it or not, all this does is say "Hello, Barbara. How are you, love?" You can substitute the allophones for your name where the ones for mine are and change the pitch of the voice as desired. A$ sets up the sentence break, number of secondary stress points and the pitch, then indicates that the phrase will start with a secondary stress point. C$ is the first sentence. B$ does the same as A$ but it does it for the second sentence, D$. The main thing I want you to notice is how the number of secondary stresses are added after the CHR$(250) in A$ and B$. CHR$(249) indicates a secondary stress, and you will see that there are 2 of these before the primary stress in C$ and 1 after it. In lines 120 and 160, CHR$(254) is the primary stress point in each sentence and signifies a falling contour. CHR$(252) in A$ and B$ is setting up for the pitch parameter, CHR$(46). This is not the same as the allophone CHR$(46) in line 150. All other CHR$'s are allophones, also.

Now, how does the computer know that the CHR$(46) in A$ and B$ is not an allophone? Very simply, it is because of the CHR$(252). When the ALPHON file sees a 252 coming through, it knows that the next allophone is going to be the value for the pitch. Similarly, when it sees a 250 coming, it knows that the next TWO CHR$'s are going to tell it how many secondary stresses to expect. If a 254 or a 255 is used as the first parameter after a 250, then the ALPHON file assumes no secondary stresses are coming, the primary stress will be on the first syllable, and the CHR$ following the 254 or 255 will tell it how many vowels (or syllables) there are in the sentence. Here, 254 indicates the sentence will have a rising pitch and the 255 means the pitch will fall.

Completely confused? I was at first. It was a little difficult to understand how CHR$(254) could be both a rising and a falling contour until I realized that, like the allophones, what they indicate depends on the CHR$ they follow. Think about it a little; sooner or later it does sink in!

Now, if a CHR$(255) is used within the phrase itself (not in the sentence break parameters), it indicates a temporary pitch change. The CHR$ that follows it will tell the pitch for the next syllable ONLY.

If you are beginning to get the idea but are still having a little difficulty grasping what I have said here, it may help if you try out the little "programettes" on pages 38-40 in the TEII manual, like the one I included earlier in this article. That's what I had to do to understand better how allophones are used! I don't understand why there aren't more programs with speech, especially when you really CAN do quite a bit with them.

Just for the heck of it, and partly because there are a lot of music programs around that show how the SOUND subprogram can be used, I decided awhile back to see if it was possible to make the computer actually sing. And I wanted a smoother result than good ol' Old McDonald. So I sat at the computer for hours on end, playing with the allophones, finding pitches to correspond with the CALL SOUND tones, and experimenting with slopes until I understood them to my own satisfaction. (When it comes to computers, I'm insatiable!) My first result was a program I called BEATSING, in which the computer sang "I Saw Her Standing There" and displayed an exceptional graphic of Paul McCartney. (No brag, just fact!) The time it took to write this program led me to write a driver program called TI SINGS, which allows anyone to enter a song for the computer to sing. It also allows disk storage of completed songs. This article is also on the disk in TI-Writer form, and with the completion of this article came the completion of the TI SINGS disk. It is now available through Trio+ Software, and if this article piques your interest, it can be had "for a song". (Ba-a-ad pun--sorry!)

I hope this article gives you a start with programming speech with the TEII. My only disappointment with the speech system is that in order to write a program that uses the speech capabilities of the TEII module, one is limited to BASIC. Maybe someone out there will begin to see the possibilities with the speech system and design an extended BASIC with the speech capabilities of the TEII. I would have loved to be able to use some of the X-BASIC features when I wrote the TI SINGS program, but had to settle for BASIC. And it would have been nice to be able to access the 32K memory for longer songs. How about it, hardware developers?

Disk 74. Pop Music Demo

Version: *Author*: Roman Majer
Requires: EA *Language*: AL *Updated*: 12/12/86

Four complete music and graphic demos from Germany with adjustable speed. These are written in 100% assembly language and show the true power of the often under-rated TI sound chip. Load E/A3 DSK1.POP.

```
dskdir. v2.0. 12-dec-96
Disk name               = XBII21
Sectors total           = 360
Sectors used            = 223
Sectors available       = 135
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >010 -README       2 DIS/VAR 80   >0f1 001
002 >002 A            21 PROGRAM      >022 020
003 >003 AM           13 PROGRAM      >036 012
004 >004 ATC          52 PROGRAM      >042 051
005 >005 C            33 PROGRAM      >075 032
006 >006 CHA          12 PROGRAM      >095 011
007 >007 CSC           4 PROGRAM      >0a0 003
008 >008 F            20 PROGRAM      >0a3 019
009 >009 FL           10 PROGRAM      >0b6 009
010 >00a FSCR          4 PROGRAM      >0bf 003
011 >00b M            21 PROGRAM      >0c2 020
012 >00c MO           18 PROGRAM      >0d6 017
013 >00d POP           5 DIS/FIX 80   >0e7 004
014 >00e S             4 PROGRAM      >0eb 003
015 >00f SCRM          4 PROGRAM      >0ee 003
```

Disk 74. Contents of file -README

To start program load the file "POP" from Editor/Assembler option 3.

It will autostart.

All other files on the disk are loaded by the main program.

You can adjust the speed with the + and - keys.

To return to the title screen hit **FCTN 9**.

Disk 75. TI-Forth Demo

Version:
Author:
Requires: EA
Language: Forth
Updated: 12/12/86

This disk looks to have come from inside TI, back when TI-Forth was being developed. It contains two really impressive demos that show off not only the power of TI-Forth, but great graphics and sound capabilities of the 99/4A.

```
dskdir. v2.0. 12-dec-96
Disk name               = TI4THDEMO
Sectors total           = 360
Sectors used            = 45
Sectors available       = 313
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 0
Number of sides         = 0
Density                 = unknown

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 FORTH         6 DIS/FIX 80   >022 005
002 >003 FORTHSAVE    39 PROGRAM      >027 038
```

Disk 76. Enhanced Display Package

Version:
Requires: XB
Author: Curtis Provance
Language: AL
Updated: 03/26/87

The ultimate package for doing 40 columns, windowing, reverse scrolling, clock/alarm, flashing color sets, character redefinition, on screen highlighting, multi-line ACCEPT AT and all sorts of other neat and useful tricks in Extended BASIC. This 40 column package does not interfere with Extended BASIC in any way. Excellent Documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = EDP_DEMO
Sectors total           = 360
Sectors used            = 348
Sectors available       = 10
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00a DOCUMENTS   225 DIS/VAR 80   >08f 170 >13c 044 >006 004 >012 006
002 >002 LOAD         24 PROGRAM      >022 023
003 >00d MERGEBASE1    3 DIS/VAR163   >00b 002
004 >011 MERGEBASE2    4 DIS/VAR163   >00e 003
005 >003 SHOWITOFF    33 PROGRAM      >039 020 >05f 002 >078 007 >139 003
006 >004 SUPPORT1     26 DIS/FIX 40   >04d 018 >07f 005 >08d 002
007 >005 SUPPORT2     33 DIS/FIX 28   >061 023 >084 009
```

Disk 76. Contents of file DOCUMENTS

[This file covers version 1.0 of Paragon Computing's Enhanced Display Package. The version 2.1 documentation can be found in The Cyc, Appendix, *Genial TRAVelER* — Ed.]

Disk 77. RAG Linker

Version:
Author: Art Greene
Requires: EA
Language: AL
Updated: 3/1/88

A powerful system for converting DIS/FIX 80 assembly object code files to PROGRAM image. This allows files to load faster and take up Was space on disk. Extremely flexible with complete documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = RAGLINKER
Sectors total           = 360
Sectors used            = 257
Sectors available       = 101
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >004 LINKER       33 PROGRAM      >053 032
002 >005 LINKES        8 PROGRAM      >073 007
003 >007 LNKDOC        8 DIS/VAR 80   >095 007
004 >008 LNKDOC1      50 DIS/VAR 80   >09c 049
005 >009 LNKDOC2      66 DIS/VAR 80   >0cd 065
006 >00a LNKDOC3      43 DIS/VAR 80   >10e 042
007 >00b LOAD          4 PROGRAM      >138 003
008 >002 RAGLIB       14 DIS/FIX 80   >022 013
009 >003 S/LOADER     31 DIS/VAR 80   >02f 030
```

Disk 77. Contents of file LNKDOC

RAG SOFTWARE
TI-99/4A PROGRAM LINKER

CONTENTS

.IF DSK1.LNKDOC1
.IF DSK1.LNKDOC2
.IF DSK1.LNKDOC3

Disk 77. Contents of file LNKDOC1

INTRODUCTION

The TI-99/4A LINKER is a tool for building assembler language memory image programs from tagged object. It makes this process simple and straight forward. LINKER's main features are:

1. Tagged object modules may be compressed and/or uncompressed,
2. Tagged object modules may be absolute or relocatable,
3. A library search can be done to resolve REFs,
4. A listing can be produced containing object information, memory maps and REF/DEF cross-references,
5. The memory image program can be built to load anywhere in the TI-99/4A's 64K address space.

The LINKER has three modes of operation depending upon the type of input in its control file.

1. Automatic.
   A single tagged object file is processed, a single library file is searched if necessary and a memory image program is produced. The memory image program will load as if the tagged object had been loaded by the E/A loader, excluding the E/A routines in low memory. That is, the object is loaded first in the 24K high memory block, then into the 8K low memory block.

2. Semi-automatic.
   A control file is processed which names one or more tagged object files to be processed and names one or more libraries to be searched. The resulting memory image program will load as if the tagged object had been loaded by the E/A loader, excluding the E/A routines in low memory. That is, the object is loaded first in the 24K high memory block, then into the 8K low memory block.

3. Complete Programmer Control.
   A control file is processed which names one or more tagged object files to be processed, names one or more libraries to be searched and which designates the memory layout for the resulting memory image program.

DISK CONTENTS

The distribution disk is a single-sided single-density disk containing the following files.

LINKER	The linker program in memory image form.
LINKES	The second segment of LINKER.
LNKDOC	The documentation for LINKER to be printed by the TI-Writer formatter.
LNKDOC1	Continuation of the documentation.
LNKDOC2	Continuation of the documentation.
LNKDOC3	Continuation of the documentation.
LOAD	Extended BASIC loader.
RAGLIB	A library of routines similar to those provided by the E/A, MM or XB.

FAIRWARE

This package is being made available via the Fairware concept. If you like the package and are using it, send a donation to:

R. A. Green
1032 Chantenay Drive
Gloucester, Ont. Canada
K1C 2K9

And, at the same time, distribute complete copies of the package to your friends. If you have any suggestions for improvements or have found any bugs please forward them to the above address.

TAILORING THE LINKER

The LINKER has several built-in default values. You can change these using one of the many "disk patchers/editors" that are available.

The first sector of file LINKER contains the default values as shown below. The printer control codes shown below are for the GEMINI 10X. All values below are in hexadecimal notation.

BYTE	*CONTENTS*	*USE*
00	FFFF	Loader flag
02	1FE0	Program segment length
04	2000	Program segment load address
06	0420	BLWP to LINKER initialization
08	A000	
0A	004B	Number of lines per page (75 decimal)
0C	0060	Maximum length of a print line (96 decimal). *Note*: 116 decimal is the absolute maximum for this value,
0E	03	Length of printer reset control sequence
0F	0C	Form feed
10	1B40	GEMINI 10X — printer reset
12	0000...	8 unused bytes for printer control sequence
1A	05	Length of printer setup control sequence
1B	1B4202	GEMINI 10X — elite mode
1E	1B30	GEMINI 10X — 8 lines per inch
20	0000...	22 unused bytes for printer control sequence
36	03	Length of printer device name
37	50494F	Printer name (PIO)
3A	0000...	25 unused bytes for printer name
53	4D4C58	Default LINKER options ("MLX ")
56	202020	
59	00	
5A	0420	Start of LINKER code

The "printer setup sequence" is sent to the printer before the first line is printed. The number of lines per page and the maximum length of the print line must be coordinated with the printer setup values you use. The length of the printer setup sequence should never be set to zero. The "printer reset" sequence is sent to the printer after the last line of the listing is printed. It should have a form feed and should restore the printer to whatever mode you usually use.

RUNNING THE LINKER

The RAG Software LINKER is independent of the module used to run it. It has been run using Option 3 of TI-Writer, Option 5 of Editor/Assembler, Extended BASIC and GRAM KRACKER.

Using Extended BASIC, the LOAD program on the disk automatically loads the LINKER. Using the other modules, the file name of the LINKER is LINKER.

The LINKER is designed to make use of any RAM that you might have in the "module slot". The LINKER work areas, called "frames" are each 1K bytes in size. These frames are used for two purposes, first, for the REF/DEF table and, second, to build 1K sections of the memory image program. With only the memory expansion unit there are 24 1K frames. If you have a cartridge with RAM such as TI's Mini Memory, DataBioTics' Super Space, Millar's Graphics' Gram Kracker or a home built super E/A you can add 4 or 8 additional frames to the LINKER's work area to allow processing of a larger memory image program.

The LINKER will prompt you for the control file name, the library file name, the memory image file name and the options for this run.

CONTROL: Enter the device and file name of your control file or of the single object file that you want processed.

LIBRARY: Enter the device and file name of the final library to be searched for unresolved REFs. A null entry (i.e. pressing **ENTER** only) means no library is to be searched.

OUTPUT: Enter the device and file name into which your memory image program is to be written. If required, more than one file may be written. The names for the second and following files are generated by incrementing the last byte of the given name.

OPTIONS: Enter the one letter option codes. The codes may be entered in any order. The option codes are:

L produce a listing of control statements and other optional data,
F show full information in the listing about each tagged object module processed,
M include in the listing a memory map of the memory image program,
X include in the listing a cross-reference listing of all REFs and DEFs,
P Pause with a message to await changing of disks for the library file and for the memory image program file,
4 Use the 4K of RAM at address >7000 for work area frames,
8 Use the 8K of RAM at address >6000 for work area frames.

If during prompting you wish to restart simply press BACK.

Note: If you have trouble understanding the following sections, do not despair. The LINKER is a powerful tool so that describing its functions sometimes gets complicated. At the same time, LINKER is easy to use if not all of its function is required for the job. If you do get bogged down in the following sections skip them and go directly to the examples at the end. The first three or four examples will show you how to do the easy things easily.

RESTRICTIONS

The LINKER has the following restrictions.

1. The size of the memory image program is limited by the number of frames available for work areas and the number of DEFs in the program. The number of frames available is 24 to 32 depending upon the options specified. At least one frame is always allocated to the REF/DEF table and for every 102 DEFs, after the first 102, another frame is allocated. The resulting memory image program can have code or data loaded in up to 31 of the 1K blocks of memory in the 64K address space of the TI 99/4A. Note that this does not mean that the maximum size of the program is necessarily limited to the size of the work area. For example, if a program has a work area (ie. BSS) that spans a whole 1K block, then that 1K block has no data or code loaded into it and is not counted.

2. Only the following tags in a tagged object can be processed:

>01	compressed object begin
0	uncompressed object begin
1	absolute program entry point
2	relocatable program entry point
3	REF in relocatable code
4	REF in absolute code
5	DEF relocatable
6	DEF absolute
7	checksum
8	checksum ignore
9	load address absolute
A	load address relocatable
B	absolute data
C	relocatable data
F	end of record

Note that the checksum, tag 7, is always ignored. The TI Assembler can produce other tags due to DORG, PSEG, DSEG and CSEG statements, but since these are not documented they cannot be processed.

LINKER INPUT

The LINKER has three types of input file. First, a control file which is the only required input. The control file contains LINKER control statements and/or tagged object modules. The control file may be either VARIABLE or FIXED with a record length of 80. Tagged object modules in the control file begin with the tag "0" or tag >01 record and end with the colon record as is usual for object modules.

Second, tagged object files. A tagged object file is the normal output from an assembly. It ends with a record with a colon in column 1. Two or more tagged object modules can be combined into one file provided that all but the last colon record are deleted. Tagged object files must be FIXED 80.

Third, library files. Library files are searched to resolve REFs. Only the modules required are processed. Each tagged object module in the library begins with one or more special header records and is terminated via the usual colon record. The header records are identified by a period in column 1. Beginning in column 2 of the header record is a list of the names that can be resolved by the module. The list contains 1 to 6 character names separated by commas with no intervening blanks (the first blank stops the scan of the header record). A name can not be started on one header record and continued onto the next. Library files are always FIXED 80.

Building a Library

If the library is going to contain *only* uncompressed object then either the E/A or TI-Writer editors can be used to put header and object records together in the library form. Remember to save the library as FIXED 80. (HINT: In TI-Writer editor use PF then "F DSK1.LIBNAME".)

NOTATION

Throughout the remainder of this document, we will talk as though the LINKER were actually loading the object programs. The LINKER does not actually load the programs, it builds a memory image file that must be loaded by some other means (ie. Option 5 of E/A, Option 3 of TI-Writer). Speaking of the LINKER is this way makes it easier to describe and to understand the functions of the various control statements.

Disk 77. Contents of file LNKDOC2

LINKER EXPRESSIONS

In the descriptions of the LINKER control statements, given below, the term "linker expression" will be used. A linker expression is a series of arithmetic operations on symbols and/or constants. The operations: + (addition), - (subtraction), * (multiplication) and / (division) are performed in a strict left to right sequence. A symbol used in an expression must be the linker origin symbol, "$", or must have been previously defined by appearing as a DEF in an object module or as the second operand of an ORIGIN or EQU statement. The value of a symbol is the address at which the symbol is loaded. Constants may be written as decimal numbers (138) as hexadecimal numbers (>8A) or as character strings ('HZ').

LINKER ORIGIN

The LINKER maintains an "origin" which represents the location in memory just past where the previous relocatable object module was loaded, or represents the value assigned by an ORIGIN statement. The linker origin is always maintained on a word boundary (i.e. the address is always even). The symbol "$" represents this value in linker expressions. Loading absolute object code has no effect on the linker origin.

LINKER CONTROL STATEMENTS

LINKER control statements are used to control the various functions of the LINKER. They are entered into a control file via an editor (either the TI-Writer or E/A editor will do). Control statements are written beginning in column 1 as a control word followed by one or more operands and then optionally followed by comments. They are similar to an assembler language statement except that there is no label field. The operands are separated from the control word by one or more blanks. The operands are separated by commas. The comments are separated from the operand field by one or more blanks. In addition, comment lines which begin with an asterisk in column 1 may be entered. In addition to control statements, the control file may contain tagged object modules.

EXAMPLE

```
*      THIS IS A COMMENT LINE
LOAD DSK1.OBJ1        MAIN PROGRAM
LOAD DSK1.SUB1        SUBROUTINE
LOAD DSK1.SUB2        SUBROUTINE
| |      |              |
| |      |              Comment
| |      Operand
| Control Word
Column 1
```

The LOAD Statement

The LOAD control statement directs the LINKER to load tagged object modules from a named file. The load statement has a single operand, the name of the file to be loaded. The file name is given in the usual way except that the disk number may be specified as an asterisk to indicate the same drive as the control file. A LOAD statement is equivalent to placing the actual tagged object modules into the control file.

Examples

```
LOAD DSK1.OBJECT1
LOAD DSK2.OBJECT2
LOAD DSK*.OBJECT3     SAME DISK AS
CONTROL FILE
LOAD DSK.DISKNAME.OBJECT4
```

The DEFs from the tagged object modules are entered into the LINKER's dictionary and may be used in expressions in following LINKER control statements.

If the "F" option was specified, LINKER will display information about the object modules loaded in the listing. The information consists of a line giving the type of object (compressed, uncompressed, relocatable or absolute) the size and address where a relocatable module is loaded, and the IDT data that was specified at assembly time. A line will be printed for each DEF in the module giving its loaded memory address, and a line will be printed naming each REF in the module. The colon record, which usually identifies the assembler which produced the object module will also be printed.

The LIBRARY Statement

The LIBRARY control statement directs the LINKER to search the named library file for any tagged object modules that contain DEFs for any *currently* unresolved REFs. Note that only REFs that are currently unresolved will be searched for. If other tagged object modules are loaded after the LIBRARY statement has been processed that contain new REFs, these new REFs will remain unresolved unless they are resolved from additional tagged object modules or other LIBRARY searches. Usually then the LIBRARY statement will follow the LOAD statements in a control file. The file name is given in the usual way except that the disk number may be specified as an asterisk to indicate the same drive as the control file.

Examples

```
LIBRARY DSK1.RAGLIB
LIBRARY DSK.DISKNAME.FILE
LIBRARY DSK*.LIB
```

Note that one final library search may be done after the control file has been processed. The library searched is the one specified via the LINKER initial menu.

The ENTRY Statement

The ENTRY control statement is used to specify the entry point of the memory image program. This is the point at which execution begins when the memory image is actually loaded. The ENTRY statement has a single operand, a linker expression that specifies the entry address.

The ENTRY statement is optional. The entry point for the program is determined by the LINKER as follows (in lowest to highest priority order):

1. Entry at the first byte loaded,
2. Entry as specified via the first tag 1 or tag 2 field processed (a tag 1 or tag 2 is generated when a symbol is named on the assembler END statement),
3. Entry as specified via the ENTRY control statement.

Note that memory image programs, by definition, begin execution at the first byte of the first segment. LINKER will insure that this is the case by producing 2 or more segments if necessary.

Examples

```
ENTRY SFIRST
ENTRY BEGIN
ENTRY BEGIN+>100
```

The BLOCK Statement

The BLOCK control statement is used to specify blocks of memory to be used for automatic loading of relocatable tagged object. (Absolute tagged object is, of course, loaded where it must be.) The BLOCK statement has three operands: the block number, the address of the beginning of the block and the size in bytes of the block. All three operands may be linker expressions (although they will usually be constants). The block address is adjusted upwards to a word boundary if necessary. The LINKER can handle four blocks of memory so that the block number (first operand) must be in the range 1 to 4.

The LINKER loads tagged object into the blocks of memory in a way analogous to the way the E/A or Mini-Memory loads tagged object. It searches the memory blocks in order to find space to load the object modules. Ordinarily, the BLOCK statements will precede the LOAD statements in the LINKER control file. Note that if blocks are respecified or if two BLOCK statements specify the same or overlapping memory blocks object modules could be loaded over top of previously loaded modules.

Examples

```
BLOCK 1,>A000,>6000   (HIGH MEMORY)
BLOCK 2,>2000,>2000   (LOW MEMORY)
BLOCK 3,>6000,0       (CARTRIDGE RAM
BLOCK 4,>4000,0       (DSR RAM)
```

The above examples are the block definitions that the LINKER has when it begins executing. Note that blocks 3 and 4 have a length of zero so that nothing will be loaded in these blocks.

The EQU Statement

The EQU statement is used to define the value for a symbol. The EQU statement has two operands. The first is a linker expression that specifies the value to be assigned to the symbol that is the second operand.

EQU statements in the control file could be used to define symbols for REFs that would otherwise remain unresolved. For example, suppose you have a relocatable object program that has REFs to the standard routines VMBR and VMBW. Further suppose you want the program to execute in the Extended BASIC environment using XB's routines in low memory. The control file could be coded as shown below.

```
EQU >2024,VMBW    DEFINE XB'S VMBW
EQU >202C,VMBR    DEFINE XB'S VMBR
LOAD DSK1.OBJECT      THE PROGRAM
```

The ORIGIN Statement

The ORIGIN control statement is used to specify the location in memory for the next relocatable tagged object module to be loaded. The ORIGIN statement cancels LINKER's automatic or semi-automatic mode. Once an ORIGIN statement has been used, the LINKER no longer searches the memory blocks for areas to load the object modules. All modules are simply loaded sequentially from the specified origin unless another ORIGIN statement is encountered.

The ORIGIN statement has two operands. The first is a linker expression that specifies the memory location. This location is adjusted upwards to a word boundary if necessary. The second operand is optional, if specified it must be a 1 to 6 character name. This name is placed in the LINKER's dictionary just as DEFs from object modules are. The value assigned to the symbol is the value of the first operand. The symbol can be used to resolve REFs.

Examples

```
ORIGIN >A000
ORIGIN >2000,LOW    DEFINE SYMBOL "LOW"
ORIGIN LOW+>1000    ORIGIN WILL BE
>3000
```

It is important to remember that once an ORIGIN statement is used, relocatable object modules are loaded sequentially in the TI 99/4A 64K address space (wrapping from >FFFF to >0000 if necessary). As with the BLOCK statement, it is possible with the ORIGIN statement to cause LINKER to load one object module over the top of another. You are in complete control and must exercise that control. LINKER assumes you know what you are doing. An ORIGIN statement could precede each LOAD statement thus giving you complete control of where each module is loaded.

Absolute object modules or sections of modules which are absolute are not affected by ORIGIN statements. They are always loaded at the location specified in the object module.

The PATCH Statement

The PATCH control statement can be used to make patches to the loaded program. The PATCH statement has two operands. The first is the location to be patched specified as a linker expression. The second operand is the patch data. The patch data may be specified in one of three ways: as a ">" followed by a string of hexadecimal digits; as a character string in single quotes; or as a linker expression. All patching is done a word (or 2 bytes) at a time, if the specified patch data is less than a word the data is extended by adding zeros to a hexadecimal string and a byte of >00 to character strings. The linker expression is taken as a word value.

Examples

```
PATCH PGM1+>A00,>FF037AFF   HEXADECIMAL
PATCH
PATCH >A010,'TEXT'          CHARACTER
PATCH
PATCH 16+PGM2,PGM1+50       EXPR -
DATA PGM1+50
*
*      PATCH A BRANCH AT LOCATION >A000
TO
*      ROUTINE "MAIN"
PATCH >A000,>0460           BRANCH
INSTRUCTION
PATCH >A002,MAIN            ADDRESS OF
ROUTINE "MAIN"
```

If the "F" option has been selected, LINKER will display the existing data in the listing before patching. It is possible to patch data into areas of memory that do not have any object code loaded previously.

The VERIFY Statement

The VERIFY control statement can be used to verify the contents of some loaded part of the program. It is often useful to verify data before patching it so that you are sure you are patching the correct location.

The VERIFY statement has two operands. The first is the location to be verified specified as a linker expression. The second operand is the verify data. The verify data may be specified in one of three ways: as a ">" followed by a string of hexadecimal digits; as a character string in single quotes; or as a linker expression. All verification is done a word (or 2 bytes) at a time, if the specified verify data is less than a word the data is extended by adding zeros to a hexadecimal string and a byte of >00 to character strings. The linker expression is taken as a word value.

If the data in memory is not the same as that specified, an error is caused. If the "F" option has been selected, LINKER will display the data in memory.

Examples

```
VERIFY PGM+>100,>0A1B2C3D0000
VERIFY >20A8,'TEXT'
VERIFY PGM1+50,PGM2
```

MEMORY IMAGE PROGRAMS

Memory image programs are just that. They consist of the exact contents of memory written to a disk. A memory image program may be split into two or more segments depending upon the length of the program. Each of the segments is written to a separate disk file. Option 3 of TI-Writer will load a segment with a maximum size of >2000 while Option 5 of Editor/Assembler will load a segment with maximum size of >2400 bytes. LINKER produces segments with a maximum length of >1FFA. The name of the first segment is specified and the names for all following segments is derived by adding 1 to the last byte of the name of the previous segment. Each memory image file has a 3 word (or 6 bytes) header that indicates to a loader where in memory to load the program. The header is:

WORD 1 Flag word which is either >FFFF to indicate that the program consists of at least one more segment after this one; or is >0000 which indicates that this is the last segment of the program.

WORD 2 Length of code in this segment in bytes. Note that the length of the segment on disk is 6 bytes more to include the 6 bytes of header information.

WORD 3 Address at which this segment is to be loaded.

NOTE that the length of the memory image program may not be the same as the length of the tagged object program. This can be caused by a work area that has no code or data loaded into it occurring at the end of the program. LINKER will not waste disk space writing out an area that you have not filled with code or data. In fact, if your program contains areas inside the program that are unused and which exceed 518 bytes in size LINKER will break the program at that point and create two separate segments. The 518 bytes is the overhead for creating another file — 1 sector for the disk catalog, 1 sector minimum for a file and 6 bytes of header information per file.

@SUBROUTINE LIBRARY

On the LINKER distribution disk is a LINKER library with name "RAGLIB" containing the following routines:

DSRLNK	Device Service Routine Link
GPLLNK	GPL Subroutine Link
KSCAN	Keyboard Scan
VMBR	VDP Multi Byte Read
VMBW	VDP Multi Byte Write
VSBR	VDP Single Byte Read
VSBW	VDP Single Byte Write
VWTR	VDP Write To Register

Which perform the same functions as the routines described in the Editor/Assembler or Mini-Memory manuals. The routines are all separate so that when the library is searched by LINKER only those used will be loaded. These routines (in particular GPLLNK) will work no matter which cartridge is used to load the memory image program. They are all relocatable object modules, and are not necessarily loaded into low memory as they are by E/A.

Note also, that the information about the DSR routine left in memory by the E/A and MM DSRLNK routines is *not* left by the library DSRLNK.

Disk 77. Contents of file LNKDOC3

EXAMPLES

The following examples show some typical uses of the LINKER. In each case, a situation is set up then the LINKER control file and the LINKER prompts are shown. In the design and coding of LINKER every effort was made (considering the memory available) to keep everything as general as possible. Because of this the LINKER can do a lot of useful things (and probably a lot of things that are not useful). The examples show some "normal" uses. With a little thought and practice you can soon develop some tricks that seem useful to you.

While experimenting with LINKER you should select all the options so that a listing is produced showing exactly what the LINKER did with your program.

EXAMPLE 1

This is the simplest case. Suppose you have a single assembler language program that is complete and requires no subroutines. The program is assembled (I prefer the RAG SOFTWARE Macro Assembler) into a relocatable tagged object module into file "DSK1.EXAMPLE1/O". You want to make a memory image program that loads into the high memory expansion (i.e. at >A000). The memory image program is to be placed in the file "DSK1.PROGRAM1".

LINKER Control File

The object file is the control file in this case.

LINKER Prompts

```
CONTROL: DSK1.EXAMPLE1/O
LIBRARY:
OUTPUT : DSK1.PROGRAM1
OPTIONS: ML
LINKER COMPLETED
```

The requested memory image program is created and written to file DSK1.PROGRAM1. A memory map of the program is printed along with a message giving the header information of the memory image file.

EXAMPLE 2

Suppose you have a single assembler language program that has REFs to the routines VMBR and VMBW (from the library supplied on the LINKER disk). The program is assembled as a relocatable tagged object module into file "DSK1.EXAMPLE2/O". You want to make a memory image program that loads into the high memory expansion (i.e. at >A000). The memory image program is to be placed in the file "DSK1.PROGRAM2". You want a listing with all the optional data LINKER can provide. Suppose also that you have a single disk system.

LINKER Control File

The object file is the control file in this case.

LINKER Prompts

```
CONTROL: DSK1.EXAMPLE2/O
LIBRARY: DSK1.RAGLIB
OUTPUT : DSK1.PROGRAM2
OPTIONS: MLFXP
INSERT LIBRARY DISK
PRESS ANY KEY TO CONTINUE
(You insert the LINKER disk and press a key)
INSERT OUTPUT DISK
PRESS ANY KEY TO CONTINUE
(You insert the disk on which you want the memory image program and press
a key)
LINKER COMPLETED
```

The requested memory image program is created and written to file DSK1.PROGRAM2. The printed listing contains full information about the object modules processed and the memory image program produced.

EXAMPLE 3

Suppose your program consists of a main program and two subroutines all three of which have been separately assembled into files DSK2.MAIN3/O, DSK2.SUBA3/O and DSK2.SUBB3/O. This program also has REFs to the routines VMBR and VMBW (from the library supplied on the LINKER disk). You want to make a memory image program that loads into the high memory expansion (i.e. at >A000). The memory image program is to be placed in the file "DSK2.PROGRAM3". You want a listing with all the optional data LINKER can provide.

LINKER Control File (EXAMPLE3/L)

```
*
* CREATE PROGRAM 3
*
LOAD DSK2.MAIN3/O
LOAD DSK2.SUBA3/O
LOAD DSK2.SUBB3/O
```

LINKER Prompts

```
CONTROL: DSK2.EXAMPLE3/L
LIBRARY: DSK1.RAGLIB
OUTPUT : DSK2.PROGRAM3
OPTIONS: MLFX
LINKER COMPLETED
```

The requested memory image program is created and written to file DSK1.PROGRAM3. A listing is printed showing everything about the linked program.

EXAMPLE 4

Suppose you have a single assembler language program that has REFs to the routines VMBR and VMBW (from the library supplied on the LINKER disk). The program is assembled into a relocatable tagged object module into file "DSK1.EXAMPLE4/O". You want to make a memory image program that loads into the low memory expansion (i.e. at >2000). The memory image program is to be placed in the file "DSK1.PROGRAM4". You want a listing with all the optional data LINKER can provide. Suppose also that you have a single disk system.

In this case you need linker control statements because you want your program loaded in low memory. Suppose you have created the control file with an editor on the same disk as the object file named "DSK1.EXAMPLE4/L".

LINKER Control File

```
* INTERCHANGE BLOCK 1 AND 2
* TO MAKE PROGRAM LOAD IN LOW MEMORY
FIRST
BLOCK 1,>2000,>2000    LOW MEMORY
BLOCK 2,>A000,>6000    HIGH MEMORY
LOAD DSK*.EXAMPLE4/O
```

LINKER Prompts

```
CONTROL: DSK1.EXAMPLE4/L
LIBRARY: DSK1.RAGLIB
OUTPUT : DSK1.PROGRAM4
OPTIONS: MLFXP
INSERT LIBRARY DISK
PRESS ANY KEY TO CONTINUE
(You insert the LINKER disk and press a key)
INSERT OUTPUT DISK
PRESS ANY KEY TO CONTINUE
(You insert the disk on which you want the memory image program and press
a key)
LINKER COMPLETED
```

The requested memory image program is created and written to file DSK1.PROGRAM4.

EXAMPLE 5

Suppose you have an assembler program in object form in file DSK2.EXAMPLE5/O. This program begins execution at START which is defined by a DEF in the program, but which is not the first byte of the program. You want to make a memory image program that loads into the high memory expansion (i.e. at >A000). The memory image program is to be placed in the file "DSK1.PROGRAM5". You want a listing with all the optional data LINKER can provide.

LINKER Control File (EXAMPLE5/L)

```
*
*    CREATE PROGRAM 5
*
LOAD DSK1.EXAMPLE5/O
ENTRY START           TELL LINKER WHERE THE ENTRY IS
```

LINKER Prompts

```
      CONTROL: DSK2.EXAMPLE5/L
      LIBRARY:
      OUTPUT : DSK1.PROGRAM5
      OPTIONS: MLFX
      LINKER COMPLETED
```

Because the program begins execution at other than the first byte, LINKER creates two segments for the program. The first segment begins at START and ends at the end of the program; it is written to file DSK1.PROGRAM5. The second segment begins at the first byte of the program and ends just before START; it is written to file DSK1.PROGRAM6.

EXAMPLE 6

Suppose, the same situation as example 5, that is, you have an assembler program in object form in file DSK2.EXAMPLE6/O. This program begins execution at START which is defined by a DEF in the program, but which is not the first byte of the program. You want to make a memory image program that loads into the high memory expansion (i.e. at >A000). In this case you want to make the memory image program be a single segment (or file). The memory image program is to be placed in the file "DSK1.PROGRAM6". You want a listing with all the optional data LINKER can provide.

LINKER Control File (EXAMPLE6/L)

```
*    CREATE PROGRAM 6
*
*  IN ORDER TO CREATE A SINGLE SEGMENT,
*  WE PATCH A BRANCH AT >A000 TO THE
REAL
*  ENTRY POINT OF THE PROGRAM, "START".
*
BLOCK 1,>A004,>5FFC   LEAVE 4 BYTES FOR
*                     THE PATCH
LOAD DSK2.EXAMPLE6/O
PATCH >A000,>0460     BRANCH INSTRUCTION
PATCH >A002,START     ADDRESS FOR BRANCH
ENTRY >A000           TELL LINKER WHERE THE ENTRY IS
```

LINKER Prompts

```
      CONTROL: DSK1.EXAMPLE6/L
      LIBRARY:
      OUTPUT : DSK1.PROGRAM6
      OPTIONS: MLFX
      LINKER COMPLETED
```

Note that the PATCH statements must come after the object module is loaded so that we can refer to the symbol "START".

EXAMPLE 7

It is sometimes convenient when programming a large program to divide it into smaller pieces. Often when this is done you need all the separate pieces to refer to the same work/data area. This can be done by making the work/data area absolute code so that all pieces know the address of the data. A problem can arise when you have a mixture of absolute and relocatable code like this.

This problem occurs because the loaders for the TI-99/4A do not keep track of absolute code and may load a relocatable module over top of your common work/data area.

Making use of the LINKER's BLOCK structure you can overcome this problem. At the same time, LINKER keeps track of all code or data loaded and will automatically produce whatever segments are required for the memory image program.

Suppose you have this situation. You have a main program (MAIN7/O), two subroutines (SUBA7/O and SUBB7/O) and an absolute work/data area (WORK7/O). The work/data area was assembled at absolute address >F000 to >FFFF.

LINKER Control File (EXAMPLE7/L)

```
*    CREATE PROGRAM 7
*
*  CHANGE THE LINKER MEMORY BLOCKS TO
*  PREVENT OVERLAY OF ABSOLUTE WORK
AREA
*
BLOCK 1,>A000,>5000  LEAVE OUT >F000 UP
LOAD DSK2.MAIN7/O
LOAD DSK2.SUBA7/O
LOAD DSK2.SUBB7/O
```

LINKER Prompts

```
      CONTROL: DSK1.EXAMPLE7/L
      LIBRARY:
      OUTPUT : DSK1.PROGRAM7
      OPTIONS: MLFX
      LINKER COMPLETED
```

EXAMPLE 8

Suppose you have an object program in file DSK1.EXAMPLE8/O. You want to make a memory image program that loads into the RAM in the cartridge slot (i.e. at >6000). The memory image program is to be placed in the file "DSK1.PROGRAM8". No listing is required.

LINKER Control File (EXAMPLE8/L)

```
*
*     CREATE PROGRAM8
*
*  MAKE PROGRAM LOAD IN CARTRIDGE RAM
*
BLOCK 1,>6000,>2000   CARTRIDGE SLOT
RAM
LOAD DSK*.EXAMPLE8/O
```

LINKER Prompts

```
      CONTROL: DSK1.EXAMPLE8/L
      LIBRARY:
      OUTPUT : DSK1.PROGRAM8
      OPTIONS: (type a blank for no options)
      LINKER COMPLETED
```

The requested memory image program is created and written to file DSK1.PROGRAM8. The program will load in the RAM at address >6000 to >7FFF.

Disk 77. Contents of file S/LOADER

```
  TITL  'LIBRARIAN XB LOAD'
  IDT   'RA GREEN'
*
*TITLE:   LIBRIAIAN loader for use with XBASIC
*
*AUTHOR:  R. A. Green
*         October 1987
*
* This loader assumes it is called with the GPL workspace,
* and that interrupts are off.
* It also assumes that no segment of the module being loaded
* will not overlay the loader.
*
* This loader is in the form of an XBASIC program.
* To build the XB program do to following.
* 1. Assemble in uncompressed object, O/LOADER.
* 2. Select XB.
* 3. CALL INIT
* 4. CALL LOAD("DSKx.O/LOADER")
* 5. SAVE DSKx.LOAD
*
*VDP Usage:
*
VPAB   EQU   >0A00         Input PAB
VBUF   EQU   >0D00         Input buffer, length >2400 + >0200
VDSKN  EQU   >3EEB         Number of last disk accessed
*
*
* XBASIC program header
*
       AORG  >8330         Setup XB line # table ptrs
       DATA  N1            Start of line # table
       DATA  NLAST         End of line # table
*
* XBASIC line number table
*
       AORG  >FE00-20
N1     DATA  5,S5
       DATA  4,S4
       DATA  3,S3
       DATA  2             Statement number
       DATA  S2            ->Statement 2
       DATA  1             Statement number
       DATA  S1            ->Statement 1
NLAST  EQU   $-1           End of line # table
 PAGE
*
* LOADER code (should be at location >FE00)
```

```
*
LOADER CLR   @>83C4             Clear user intrpt addr
       BL    @VMBRA             Get disk number from VDP
       DATA  VDSKN,PABNO,1
ONE    EQU   $-1                BYTE OF 1
       SOCB  @X30,@PABNO        Make disk number a character
*
NXTSEG BL    @VMBWA             Write PAB to VDP
       DATA  VPAB,PAB,PABL
       CLR   @>837C             Zero GPL status byte
       LI    R0,VPAB+14
       MOV   R0,@>8356          Set pointer to name in PAB for DSR
       LI    R0,4               length of "DSKn"
       MOV   R0,@>8354          Set length of device name for DSR
       LI    R12,>1100          CRU address of disk ROM
       MOV   R12,@>83D0
       SBO   0                  Turn on disk ROM
       MOV   @>4008,R3          Start of device search list
DSR1   MOV   R3,R2              Get next search list address
       JEQ   ERROR              Error if end of search chain
       MOV   *R2+,R3            Get next entry from search list
       MOV   *R2+,R9            Get DSR routine address from the list
       CB    @>8355,*R2+        Are the name lengths the same
       JNE   DSR1               JUMP no, try next device
       MOV   R0,R5              Get device name length for comparing
       LI    R6,PABNL+2         R6->the device name
DSR3   CB    *R6+,*R2+          Compare device names
       JNE   DSR1               JMP if not correct device name
       DEC   R5
       JNE   DSR3               Loop for device name
*                               FOUND THE DEVICE
       BL    *R9                Go to the DSR
       JMP   ERROR              JMP if device error
       SBZ   0                  Turn off device rom
*
       MOV   @PABBUF,R0         Get address of buffer in VDP
       BL    @VMBRB             FLAG to R3, LNGT to R4,
       DATA  >83E6,6            ADDR to R5
       MOV   @START,R6          START will be zero for 1st segment
       JNE   NFIRST             JMP if not 1st segment
       MOV   R5,@START          Save starting address of module
NFIRST MOV   R4,R2              Segment length to R2
       MOV   R5,R1              Segment address to R1
       AI    R0,6               Bump VDP addr past header
       BL    @VMBR              Load the segment into CPU memory
       AB    @ONE,@LASTB-1      Increment segment name in PAB
       MOV   R3,R3              Test segment FLAG
       JNE   NXTSEG             Go do next segment if any
       BL    @VMBRA
LAST   DATA  >0800,>8300,32
BLANK  EQU   $-1           A BLANK
```

```
       BL    @VMBWA
       DATA  >0380,>830C,32
COPY   BL    @VMBRA         Copy char defns
ADDIN  DATA  >0400,>8300,32
       BL    @VMBWA
ADDOUT DATA  >0900,>8300,32
N32    EQU   $-2            constant of 32
       A     @N32,@ADDIN    Bump VDP addresses
       A     @N32,@ADDOUT
       C     @ADDIN,@LAST   Finished?
       JL    COPY          JMP no.
       LI    R1,VREGS      Point to values for VDP registers
       LI    R2,4
SVR    MOVB  *R1+,*R15     Set VDP registers
       DEC   R2
       JNE   SVR
* Clear some of PAD as some pgms like this
RUN    LI    R0,>8300      Start of PAD
CLEAR  CLR   *R0+
       CI    R0,>8360
       JL    CLEAR
       MOV   @2,R11        Set return to power up routine
       B     @0            Go to loaded program
START  EQU   $-2           Start address for module
*
VMBWA  MOV   *R11+,R0      Get VDP address
       MOV   *R11+,R1      Get CPU address
       ORI   R0,>4000
       SWPB  R0
       MOVB  R0,*R15
       SWPB  R0
       MOVB  R0,*R15
       MOV   *R11+,R2      Get length
VMBW2  MOVB  *R1+,@>8C00
       DEC   R2
       JNE   VMBW2
       RT
VMBRA  MOV   *R11+,R0      Get VDP address
VMBRB  MOV   *R11+,R1      Get CPU address
       MOV   *R11+,R2      Get length
VMBR   SWPB  R0
       MOVB  R0,*R15
       SWPB  R0
       MOVB  R0,*R15
       NOP
VMBR2  MOVB  @>8800,*R1+
       DEC   R2
       JNE   VMBR2
       RT
ERROR  BLWP  @0
```

```
*
VREGS  DATA  >0E83       VDP R3=>0E
       DATA  >0184       VDP R4=>01
*
X30    BYTE  >30
PAB    DATA  >050C             LOAD OP code, flags
PABBUF DATA  VBUF              VDP buffer address
       DATA  0
       DATA  >2400             Maximum segment length
PABNL  BYTE  0,14              Screen offset, name length
       TEXT  'DSK'             Device name
PABNO  BSS   1                 Disk number
       TEXT  '.LIBRARIAN'
LASTB  TEXT  '    '            Byte previous is incremented
PABL   EQU   $-PAB             Length of the PAB
*
* XBASIC program statements
*
       AORG >FF83
*
       BYTE 21
S3     BYTE >C8,18
       TEXT '  Linker Librarian'
       BYTE 0
       BYTE 14
S4     BYTE >C8,11
       TEXT '         by'
       BYTE 0
       BYTE 18
S5     BYTE >C8,15
       TEXT '    Tom Bentley'
       BYTE 0
       BYTE >05     Length of statement
S2     BYTE >86     GOTO token
       BYTE >C9     Statement number token
       BYTE >00,>02 Statement number
       BYTE >00     End of statement
*
       BYTE >26     Length of statement
S1     BYTE >9D     CALL token
       BYTE >C8     Unquoted string token
       BYTE >04     Length of string
       TEXT 'INIT'  String
       BYTE >82     :: token
       BYTE >9D     CALL token
       BYTE >C8     Unquoted string token
       BYTE >04     Length of string
       TEXT 'LOAD'
       BYTE >B7     ( token
       BYTE >C2     Minus token
       BYTE >C8     Unquoted string token
```

```
BYTE >05     Length of string
TEXT '31804'
BYTE >B3     Comma token
BYTE >C8     Unquoted string token
BYTE >03     Length of string
TEXT '254'
BYTE >B3     Comma token
BYTE >C8     Unquoted string token
BYTE >03     Length of string
TEXT '000'
BYTE >B6     ) token
BYTE >00     End of statement
END
```

Disk 78. Clint Pulley Disk

Version: *Author*: Clint Pulley
Requires: *Language*: *Updated*: 03/28/87

A collection of previously unreleased programs by Clint Pulley. Some in c99, others in BASIC, and assembly. Some source code included. Prepared for the Boston Fayuh.

```
dskdir. v2.0. 12-dec-96
Disk name               = PULLEY
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01b -README       7 DIS/VAR 80   >0e7 002 >112 003 >00d 001
002 >00c BREAKFORTH   18 DIS/VAR 80   >0d6 017
003 >003 BT1          23 DIS/VAR 80   >037 022
004 >004 BT2          19 DIS/VAR 80   >04d 018
005 >005 BT3          21 DIS/VAR 80   >05f 020
006 >006 BT4          12 DIS/VAR 80   >073 011
007 >010 BTHU1         8 PROGRAM      >108 007
008 >008 BTHU;S        6 DIS/VAR 80   >085 005
009 >009 BTHUDOC      18 DIS/VAR 80   >07e 007 >08a 010
010 >00b FORTHSWAP     9 PROGRAM      >0cb 008
011 >007 GOTHIC;C     10 DIS/VAR 80   >094 009
012 >012 GOTTAB1      40 DIS/VAR 80   >115 039
013 >013 GOTTAB2      51 DIS/VAR 80   >13c 044 >014 006
014 >00a GOTTAB3      52 DIS/VAR 80   >09d 046 >0d3 003 >110 002
015 >002 LEFT/RIGHT   24 PROGRAM      >020 023
016 >00f PFL;S        29 DIS/VAR 80   >0ec 028
017 >011 PMENU         2 DIS/VAR 80   >10f 001
018 >01a SD            5 PROGRAM      >01c 004
019 >00e UTIL1         4 PROGRAM      >0e9 003
```

Disk 78. Contents of file -README

Fayuh Disk

This disk contains some of the programs which I mentioned in my talk. I've tried to select files which have not been widely distributed.

The programs are :

BREAKFORTH
A breakout game in FORTH. Use the FORTHSWAP program to move to a FORTH disk.

BTHU etc
My 9900 Breakthru game. Files BTHU;S and BT1/2/3 are the source code, BTHU1 is the program file and BTHUDOC contains instructions.

FORTHSWAP
ExBas program for converting FORTH screens to D/V80 files and back.

GOTHIC etc
Source code for a c99 program which prints banners in large Gothic letters. You may have to set your printer externally (I get best results in condensed mode at 10 lines/ inch). The use of a pointer array may be of interest.

LEFT/RIGHT
My first "substantial" program for the 99/4A. It makes use of the color registers to speed up the game. TI Basic.

PFL;S
Source code for my program file loader which I use on my E/A program disks as UTIL1. Reads a file, PMENU, which has a descriptor in columns 1-12 and a filename in columns 13. . . A menu is displayed on the screen and the selected program is loaded along with the E/A utilities.

SD
My show directory utility.

I hope that this disk will provide useful examples of some of the programming methods which I have used with my 99/4A over the years.

Clint Pulley

Disk 78. Contents of file BREAKFORTH

```
( BREAKFORTH INTRODUCTION)  : BFSTART ;

CLS 14 1 GOTOXY ." BREAKFORTH" 14 2 GOTOXY ." **********" CR CR
."   Adapted to TI FORTH by Clint Pulley"  CR
CR ." This game demonstrates the the speed of"
CR ." TI FORTH.   It is written for text mode"
CR ." and uses only the RESIDENT and SYNONYMS"
CR ." vocabularies."  CR
CR ." To play, type   BREAKFORTH  then select"
CR ." speed and number of balls.  Press enter"
CR ." after each selection." CR
CR ." To move your paddle,  use the right and"
CR ." left arrow keys." CR
14 20 GOTOXY ." GOOD LUCK!" 0 22 GOTOXY -->

( BREAKFORTH adapted from A. Schaefer, BYTE Aug 80)
: ARRAY  <BUILDS 2 * ALLOT  DOES> SWAP 2 * + ;
90 ARRAY BLOCKMAP 0 VARIABLE #BLOCKS
0 VARIABLE SPEED  0 VARIABLE SPVAR  0 VARIABLE SCORE
0 VARIABLE XPOS   0 VARIABLE YPOS   0 VARIABLE PPOS
1 VARIABLE XDIR   1 VARIABLE YDIR   0 VARIABLE BEST
: LINE 0 SWAP GOTOXY ;
: BALLSET GOTOXY 42 EMIT ;  : BALLCLR GOTOXY 32 EMIT ;
: PDLSET PPOS @ 22 GOTOXY 61 DUP DUP EMIT EMIT EMIT ;
: PDLCLR PPOS @ 22 GOTOXY 32 DUP DUP EMIT EMIT EMIT ;
: MOVEPDL PDLCLR 83 = IF -1 PPOS @ + 2 MAX PPOS !
  ELSE 1 PPOS @ + 35 MIN PPOS ! ENDIF PDLSET ;
: PADDLE ?TERMINAL IF QUIT ENDIF ?KEY -DUP IF MOVEPDL ENDIF ;
: BEEP 52 GPLLNK ;  : BOP 50 0 DO BEEP LOOP ;
: BLOCKMAPINDEX -2 + 18 * SWAP 2 / + 1- ;
-->

( BREAKFORTH SCREEN 2)
: BLKTST YPOS @ 7 < IF XPOS @ YPOS @ BLOCKMAPINDEX BLOCKMAP @
  ELSE 0 ENDIF ;
: BLKCLR OVER OVER SWAP 126 AND SWAP GOTOXY 32 EMIT 32 EMIT
  BLOCKMAPINDEX BLOCKMAP 0 SWAP ! ;
: BLKSET OVER OVER GOTOXY 91 EMIT 93 EMIT BLOCKMAPINDEX
  BLOCKMAP 1 SWAP ! ;
: WALLSET 37 2 DO I 1 AND 0= IF I 2 BLKSET I 3 BLKSET
  I 4 BLKSET I 5 BLKSET I 6 BLKSET ENDIF LOOP 90 #BLOCKS ! ;
: INIT CLS 0 LINE ." Speed (1-9, 1 is fastest) : " KEY DUP EMIT
  KEY DROP 48 - 1 MAX 9 MIN 5 * SPEED !
  0 LINE ." Number of balls desired (1-9) : " KEY DUP EMIT
  KEY DROP 48 - 1 MAX 9 MIN  CLS
  40 0 DO I 1 GOTOXY 61 EMIT LOOP
  22 1 DO 0 I GOTOXY 124 EMIT 124 EMIT
         38 I GOTOXY 124 EMIT 124 EMIT LOOP        -->
```

```
( BREAKFORTH SCREEN 3)
  WALLSET 0 SCORE ! 0 LINE 0 0 GOTOXY ." Score: 0 "
  15 0 GOTOXY ." Best: " BEST ?  30 0 GOTOXY ." Ball: " ;
: XCHK
  XPOS @ 2 < IF XDIR @ MINUS XDIR ! 2 XPOS ! BEEP ENDIF
  XPOS @ 37 > IF XDIR @ MINUS XDIR ! 37 XPOS ! BEEP ENDIF ;
: YCHK
  YPOS @ 7 < IF SPVAR @ 8 MIN SPVAR ! ENDIF
  YPOS @ 6 < IF SPVAR @ 7 MIN SPVAR ! ENDIF
  YPOS @ 5 < IF SPVAR @ 6 MIN SPVAR ! ENDIF
  YPOS @ 4 < IF SPVAR @ 5 MIN SPVAR ! ENDIF
  YPOS @ 3 < IF SPVAR @ 4 MIN SPVAR ! ENDIF
  YPOS @ 2 < IF 1 YDIR ! 2 YPOS ! 2 SPVAR ! BEEP ENDIF ;
5 ARRAY PDLVEC
-2 0 PDLVEC !  -1 1 PDLVEC !  0 2 PDLVEC !
 1 3 PDLVEC !   2 4 PDLVEC !   -->

( BREAKFORTH SCREEN 4)
: PCHK  0 YPOS @ 21 >
  IF 21 YPOS ! XPOS @ PPOS @ - 1+ DUP 0< 0= OVER 5 < AND
  IF -1 YDIR ! BEEP PDLVEC @ DUP 0= IF DROP ELSE XDIR ! ENDIF
  ELSE DROP 1+  ENDIF  ENDIF  ;
: CLR  XPOS @ YPOS @ BLKCLR
  YPOS @ 27 - ABS SCORE +! 7 0 GOTOXY SCORE ? BOP
  YDIR @ MINUS YDIR ! #BLOCKS @ 1- #BLOCKS ! ;
: BALLCHK YDIR @ YPOS +! XDIR @ XPOS +! XCHK YCHK PCHK
  BLKTST IF CLR ENDIF ;
: BALL XPOS @ YPOS @ BALLCLR
  BALLCHK DUP 0= IF XPOS @ YPOS @ BALLSET ENDIF ;
: LAUNCH 2 RND 1 = IF 1 ELSE -1 ENDIF XDIR ! 1 YDIR !
  34 RND 3 + XPOS ! 8 YPOS ! 10 SPVAR ! ;
: GAMECHK #BLOCKS @ 0=
  IF WALLSET SPEED @ 5 - 5 MAX SPEED ! LAUNCH ENDIF ;  -->

( BREAKFORTH SCREEN 5 - MAIN LOOP)
: DELAY SPEED @ SPVAR @ * 0 DO LOOP ;
: BREAKFORTH        RANDOMIZE
  BEGIN 18 PPOS ! INIT PDLSET
  0 DO 2000 SPEED @ / 0 DO DELAY PADDLE LOOP
  36 0 GOTOXY I 1+ . LAUNCH
  BEGIN 10 0 DO PADDLE DELAY LOOP
  BALL GAMECHK UNTIL 100 0 DO 54 GPLLNK LOOP
  LOOP SCORE @ BEST @ MAX BEST !
  200 0 DO DELAY LOOP 12 8 GOTOXY ." Play again? " KEY DUP EMIT
  KEY DROP 89 = 0= UNTIL  CLS 0 0 GOTOXY ;
;S
```

Disk 78. Contents of file BT1

```
       TITL 'BREAKTHRU MAIN PROGRAM'
       PAGE
*
*  REV 84/06/16 09:20
*
START  LWPI >8300        PAD WSP
       LI   R9,VSBW
       LI   R10,VMBW
       LI   R12,VSBR
       LI   R13,NGB      NEGATE MS BYTE
       LI   R14,VF       VRAM FILL
       BL   @INI         INITIALIZE VDP
RS1    BL   @TTL         DISPLAY TITLE SCREEN
RS2    BL   @DHS         POST HIGH SCORE FOR LEVEL
       BL   @NGM         NEW GAME
NBX    BL   @NBD         NEW BOARD
NLX    BL   @NBL         NEW BALL
*
ML     LIMI 2            *** MAIN LOOP ***
       LIMI 0
       SETO @>83D6       INHIBIT SCREEN BLANKOUT
       CLR  R8           INDEX FOR SOUND LISTS
       LI   R0,>305      GET PADDLE X POSITION
       BLWP *R12
       SRL  R1,8         SHIFT TO LS BYTE
       MOV  R1,R6        SAVE X POS
       CLR  R1
       BL   @RD          GET X JOYSTICK VALUE
       SLA  R1,3         (*8)
       JEQ  M0A          IF NO JOYSTICK
       MOVB @>8375,R3    GET KEY VALUE
       JLT  M0A          IF NO KEY DOWN
       SLA  R1,1         DOUBLE PADDLE SPEED
M0A    LI   R0,>785      PADDLE SPRITE X VELOCITY
       BLWP *R9
       CI   R6,16        TEST IF PADDLE AT LEFT WALL
       JH   M1           IF NOT LHS
       MOV  R1,R1
       JGT  M2           IF PADDLE VEL > 0
M0     CLR  R1           SET PADDLE VELOCITY TO 0
       BLWP *R9
       JMP  M2
M1     CI   R6,223       TEST IF PADDLE AT RIGHT WALL
       JLE  M2           IF NOT RHS
       MOV  R1,R1
       JGT  M0           IF PADDLE VEL > 0
M2     MOV  R1,R7        SAVE PADDLE VELOCITY
       LI   R0,>300      GET BALL Y POSITION
```

```
        BLWP *R12
        SRL  R1,8          TO LS BYTE
        MOV  R1,R2         SAVE Y POS
        INC  R0            GET BALL X POSITION
        BLWP *R12
        SRL  R1,8
        MOV  R1,R3         SAVE X POSITION
        LI   R4,4          COMPUTE CHARACTER POSITION
        MOV  R4,R0         OF THE CENTRE OF THE BALL
        A    R2,R4         Y+4
        SRL  R4,3          INT()/8
        SLA  R4,5          *32
        A    R3,R0         X+4
        SRL  R0,3
        A    R4,R0         R0 CONTAINS CHAR POSITION
        CI   R0,768
        JHE  M2A           IF AT BOTTOM OF SCREEN
        BLWP *R12          GET CHARACTER VALUE
        CI   R1,>7F00
        JLE  M2A           IF NOT A BRICK
        BL   @HIT          CLEAR BRICK AND UP SCORE
        INCT R8            (=2)
        BL   @SND
        DEC  @BKL          BRICKS LEFT ON SCREEN
        JNE  ML            IF MORE BRICKS
        CLR  R1            SPRITE MOTION OFF
        MOVB R1,@>837A
        INC  @BCT          AWARD EXTRA BALL
        AB   @INBVL,@BVL   INCREASE BALL VELOCITY
        JMP  NBX           SETUP NEW BOARD
M2A     LI   R0,>780       SPRITE VELOCITY TABLE
        MOVB @>837B,R1     VDP STATUS
        ANDI R1,>2000      COINCIDENCE BIT
        JEQ  M3            IF NO COINC
        BLWP *R12          GET BALL Y VEL
        MOV  R1,R1
        JLT  M5            IF BALL MOTION ^
        BL   *R13          REVERSE Y VEL
        BLWP *R9
        INC  R0
        BLWP *R12          GET BALL X VEL
        MOV  R7,R7         CHECK PADDLE MOTION
        JEQ  MP2           IF NO MOTION
        JGT  MP1           IF MOTION >
        AI   R1,>400       SLICE BY CHANGING X VEL
        JMP  MP2
MP1     AI   R1,>FC00
MP2     S    R3,R6         PADDLE X POS - BALL X POS
        CI   R6,4
        JLT  M2D           IF BALL NOT HITTING LEFT END
```

```
        MOV  R1,R1
        JLT  M2C          IF X VEL -
        JMP  M2B
M2D     CI   R6,-12
        JGT  M2C          IF BALL NOT HITTING RIGHT END
        MOV  R1,R1
        JGT  M2C          IF X VEL +
M2B     BL   *R13         REVERSE X VELOCITY
M2C     BLWP *R9
        INC  R8           (=1)
        BL   @SND         PADDLE HIT SOUND
ML1     B    @ML
M3      CI   R2,192       CHECK FOR BOTTOM OF SCREEN
        JLE  M4           IF BALL NOT MISSED
        CLR  R1           BALL LOST !
        MOVB R1,@>837A    TURN OFF SPRITE MOTION
        LI   R2,>10       CLEAR VELOCITY TABLE
        BL   *R14
        LI   R8,3
        BL   @SND         LOST BALL SOUND
        LIMI 2            WAIT FOR SOUND TIMEOUT
LBW     MOVB @>83CE,R0
        JNE  LBW          IF SOUND STILL ACTIVE
        LIMI 0
        DEC  @BCT         - 1 BALL
        JEQ  M3A          IF ALL BALLS GONE
        A    @INBVL,@BVL  INCREASE BALL VELOCITY
        B    @NLX         LAUNCH NEW BALL
M3A     MOV  @CS,R4       ** GAME OVER **
        BL   @UHS         UPDATE HIGH SCORE
        BL   @END         ENDGAME ROUTINE
        CI   R3,6         TEST KEY VALUE
        JEQ  M3C          IF 'REDO'
        B    @RS1         RESET GAME IF 'BACK'
M3C     B    @RS2         NEXT GAME
*
M4      CI   R2,7         TEST Y POSITION
        JGT  M5           IF BALL NOT AT TOP
        BLWP *R12         GET Y VEL
        MOV  R1,R1
        JLT  REVEL        IF MOTION ^
M5      INC  R0           ADDR OF X VEL
        BLWP *R12         GET X VEL
        CI   R3,16        TEST X POSITION
        JGT  M6           IF NOT LHS
        MOV  R1,R1        CHECK X VEL
        JGT  ML1          IF MOTION >
        JMP  REVEL
M6      CI   R3,231       TEST X POSITION
        JLE  ML1          IF NOT RHS
        MOV  R1,R1
```

```
       JLT  ML1          IF MOTION <
REVEL  BL   *R13         REVERSE VEL
       BLWP *R9          UPDATE VELOCITY
       BL   @SND         WALL HIT SOUND
       JMP  ML1
*
```

Disk 78. Contents of file BT2

```
       TITL ' BREAKTHRU DISPLAY SETUP CODE'
       PAGE
*
*  REV 84/06/16  08:30
*
BRD    DATA >F00F,>F00F  BORDER CHAR PATTERN
       DATA >F00F,>F00F
       DATA >3C7E,>FFFF  BALL CHAR PATTERN
       DATA >FFFF,>7E3C
SPA    DATA >D010,>8004  SPRITE ATTRIB TABLE
       DATA >BC78,>8401
       DATA >D000        END OF SPRITES
CTB    DATA >B140,>88EE  COLOR TABLE
       DATA >22BB,>FFCC,>DD66
BPT    DATA >7700,>0077  BORDER CHAR LAYOUT
*
PFM    TEXT 'PRESS FIRE TO PLAY !'
*
INI    MOV  R11,R15      INITIALIZE VDP
       LI   R0,>01E2     SPRITE SIZE 2
       BLWP @VWTR        SET VDP REG 1
       SWPB R0
       MOVB R0,@>83D4    REG 1 RELOAD LOC
       LI   R0,>0707     CYAN BACKGROUND
       BLWP @VWTR        SET VDP REG 7
       LI   R0,>400
       CLR  R1           CLEAR SPRITE CHAR PATTERNS
       LI   R2,64
       BL   *R14
       LI   R0,>380      SET CHAR COLOR TO
       LI   R1,>1700     BLACK/CYAN
       SRL  R2,2         (=16)
       BL   *R14
       LI   R0,>038E     SET BRICK COLORS
       LI   R1,CTB
       LI   R2,10
       BLWP *R10 VMBW
       LI   R0,>0BB8     CHAR PAT 119
       LI   R1,BRD       BORDER CHAR
       DECT R2           (=8)
       BLWP *R10
       A    R2,R0        CHAR PAT 120
       A    R2,R1
       BLWP *R10         BALL CHAR
       LI   R0,>400
       BLWP *R10         BALL SPRITE PATTERN
       LI   R0,>420
       SETO R1           PADDLE SPRITE PATTERN
```

```
       BL   *R14
       LI   R0,>430
       BL   *R14
       B    *R15
*
NGM    MOV  R11,R15       START NEW GAME
       MOV  @STBVL,@BVL   SET BALL VELOCITY
       BL   @BTX          PUT TEXT ON BOARD
       CLR  @CS           CLEAR SCORE
       LI   R2,6          SET BALL COUNT TO 6
       MOV  R2,@BCT
       LI   R0,1          DRAW BORDER
       LI   R1,>7700
       LI   R2,30         DO TOP
       BL   *R14
       MOV  R2,R0         DO SIDES
       LI   R1,BPT        EDGE LAYOUT
       LI   R2,4
L4     BLWP *R10
       AI   R0,32         NEXT ROW
       CI   R0,766
       JLT  L4            IF NOT ROW 24
       SRL  R2,2          (R2)=1
       BLWP *R10
       B    *R15
*
NBD    MOV  R11,R15       SETUP NEW BOARD
       LI   R0,224        NO OF BRICKS
       MOV  R0,@BKL       BRICKS LEFT COUNTER
       LI   R0,386        BRICK ROW 1
       LI   R1,>8000      CHAR 128
       LI   R2,28
L5     BL   *R14          DRAW A ROW OF BRICKS
       AI   R0,-32        NEXT ROW
       AI   R1,>800       NEXT CHAR VALUE
       CI   R0,289        GAP?
       JGT  L5            IF IN 1ST BLOCK
       AI   R0,-128       LEAVE 4 EMPTY ROWS
L6     BL   *R14
       AI   R0,-32
       AI   R1,>800
       CI   R0,31
       JGT  L6            IF IN 2ND BLK
       LI   R3,5          REPEAT BRICK HIT SOUND
SD1    LI   R8,4
       LIMI 2
SDW    MOVB @>83CE,R0     WAIT FOR SOUND TIMEOUT
       JNE  SDW WAIT      IF SOUND STILL ACTIVE
       LIMI 0
       DEC  R3
```

```
        JLT   SD2           IF SOUND PLAYED 5 TIMES
        BL    @SND
        JMP   SD1
SD2     B     *R15
*
NBL     MOV   R11,R15       LAUNCH NEW BALL
        MOV   @BCT,R2       GET BALL COUNT
        CI    R2,6
        JLE   NL2           IF BCT < 7
        LI    R2,6          MAX OF 5 ON BOARD
NL2     LI    R0,248        SCREEN LOC OF BALLS
        LI    R1,'  '
        BL    *R14          BLANK OUT BALLS
        DEC   R2
        JEQ   L7            IF ONLY 1 BALL LEFT
        LI    R1,>7800      CHAR 120
        BL    *R14          DISPLAY BALLS LEFT
L7      LI    R0,>300       SPRITE ATTRIB TBL
        LI    R1,SPA
        LI    R2,9
        BLWP  *R10
        LI    R0,>0200      2 SPRITES IN MOTION
        MOV   R0,@>837A
        LI    R0,549
        LI    R1,PFM
        LI    R2,20
        BLWP  *R10          POST START MSG
L8      INC   R8            FOR RANDOM START LOC
        BL    @RD
        JEQ   L8            WAIT FOR KEY PRESS
        LI    R1,'  '
        BL    *R14          CLEAR MSG
        LI    R0,>300       SET STARTING Y POS
        LI    R1,>6400
        ANDI  R8,>1F        RANDOM VALUE IN (0,31)
        SLA   R8,8          TO MS BYTE
        A     R8,R1         RANDOMIZE STARTING POS
        BLWP  *R9
        LI    R0,>780       VEL TBL LOC
        LI    R1,BVL        SET INITIAL VELS
        LI    R2,2
        BLWP  *R10          BEGIN MOTION
        INCT  R0
        CLR   R1            CLEAR Y VEL REMAINDER
        BLWP  *R9
        B     *R15
*
```

Disk 78. Contents of file BT3

```
       TITL 'BREAKTHRU TITLES AND SOUND'
       PAGE
*
*  REV 84/06/16  08:35
*
* BREAKTHRU TITLE SCREEN TEXT
*
TL1    DATA 16,200
       TEXT '9900 BREAK-THRU!'
TL2    DATA 17,424
       TEXT '1984 CLINT PULLEY'
TL3    DATA 14,713
       TEXT '-PRESS ANY KEY'
TL4    DATA 28,546
       TEXT 'GAME OVER-PRESS REDO OR BACK'
GS0    DATA 13,490
       TEXT 'SELECT LEVEL-'
GS1    DATA 8,554
       TEXT '1 = LAZY'
GS2    DATA 10,586
       TEXT '2 = NORMAL'
GS3    DATA 11,618
       TEXT '3 = FRANTIC'
GS4    DATA 14,650
       TEXT '4 = IMPOSSIBLE'
GS5    DATA 12,714
       TEXT 'YOUR CHOICE?'
*
* VELOCITY TABLES
*
TVS    DATA >0A09        STARTING
       DATA >1411        VELOCITY
       DATA >201B        TABLE
       DATA >2D29
TVI    DATA >0202        VELOCITY
       DATA >0202        INCREMENT
       DATA >0303        TABLE
       DATA >0404
*
TTL    MOV  R11,@RET     DISPLAY TITLE SCREEN
       BL   @CLS         CLR SCREEN
       BL   @NBD         DRAW BARS
       LI   R8,DSP
       LI   R1,TL1       GAME TITLE
       BL   *R8
       LI   R0,423
       LI   R1,>0A00
       BLWP *R9          COPYRIGHT SYMBOL
```

```
       LI   R1,TL2       COPYRIGHT TITLE
       BL   *R8
       LI   R1,GS0       SPEED SELECTION
       BL   *R8
       LI   R1,GS1
       BL   *R8
       LI   R1,GS2
       BL   *R8
       LI   R1,GS3
       BL   *R8
       LI   R1,GS4
       BL   *R8
       LI   R1,GS5
       BL   *R8
       CLR  R4
N1     BL   @RDK         SELECT SPEED
       JEQ  N1           IF NO KEYPRESS
       MOVB @>8375,R1    CHAR VALUE
       SRL  R1,8         TO LS BYTE
       AI   R1,-49       CONVERT TO BINARY
       JLT  N1           IF < '1'
       CI   R1,3
       JH   N1           IF > '4'
       MOV  R1,@LV       SAVE SPEED LEVEL
       SLA  R1,1         CONVERT TO WORD OFFSET
       MOV  @TVS(R1),@STBVL
       MOV  @TVI(R1),@INBVL
       SLA  R1,4         (*16)
       AI   R1,552       PLACE BALL BY LEVEL MSG
       MOV  R1,R0
       LI   R1,>7800
       BLWP *R9
N2     BL   @RDK         DEBOUNCE
       JNE  N2           IF KEY DOWN
       LI   R1,TL3       PRESS ANY KEY MSG
       BL   *R8
N2A    BL   @RDK
       JEQ  N2A          WAIT FOR KEYPRESS
       BL   @CLS
       MOV  @RET,R11
       B    *R11
*
END    MOV  R11,@RET     END GAME SUBROUTINE
       LI   R0,>300      TURN OFF SPRITES
       LI   R1,>D000
       BLWP *R9
N2B    LI   R1,TL4       GAME OVER MESSAGE
       BL   @DSP
       CLR  R2
N3     BL   @RDK         SELECT ACTION
       JEQ  N3
```

```
       BL   @CLS
       MOVB @>8375,R3    CHAR VALUE
       SRL  R3,8
       CI   R3,5
       JEQ  EXIT         IF QUIT (FCTN =)
       CI   R3,6
       JEQ  N4           IF REDO (FCTN 8)
       CI   R3,15
       JNE  N2B          IF NOT BACK (FCTN 9)
N4     MOV  @RET,R11
       B    *R11
*
EXIT   LIMI 2            RETURN TO MASTER SCREEN
       LWPI >83E0        GPLWS
       BLWP @0
*
* BREAKTHRU SOUND ROUTINE AND TABLES
*
SND    SLA  R8,1         (*2) LIST TABLE OFFSET
       MOV  @LPT(R8),R1  GET SOUND LIST POINTER
       MOV  *R1+,R2      LENGTH
       LI   R0,>1000     VDP RAM BUFFER ADDR
       BLWP *R10         (R1) = ADDR OF LIST
       MOV  R0,@>83CC    POINTER TO LIST IN VRAM
       SRL  R0,4         (=>0100)
       SOCB R0,@>83FD    SET VRAM FLAG
       MOVB R0,@>83CE    START SOUND GENERATOR
       B    *R11
*
* SOUND LIST POINTER TABLE
*
LPT    DATA SL0          WALL
       DATA SL1          PADDLE
       DATA SL2          BRICK
       DATA SL3          LOST BALL
       DATA SL4          NEW BALL
*
* SOUND LISTS  (LENGTH IN FIRST WORD)
*
SL0    DATA 11
       BYTE 1,>9F,1,3,>8F,>07,>90,3
       BYTE 1,>9F,0
SL1    DATA 16
       BYTE 1,>9F,1,3,>83,>15,>90,2
       BYTE 3,>8A,>0A,>90,2,1,>9F,0
SL2    DATA 26
       BYTE 1,>9F,1,3,>83,>15,>90,2
       BYTE 3,>8A,>0A,>90,2,3,>8F,>08,>90,2
       BYTE 3,>8B,>06,>90,3,1,>9F,0
SL3    DATA 36
```

```
       BYTE 1,>9F,1,3,>8A,>2F,>90,8
       BYTE 3,>87,>32,>90,8,3,>87,>35,>91,8
       BYTE 3,>8A,>38,>92,8,3,>80,>3C,>94,8
       BYTE 3,>89,>3F,>97,12,1,>9F,0
SL4    DATA 26
       BYTE 1,>9F,1,3,>8B,>06,>90,2
       BYTE 3,>8F,>08,>90,2,3,>8A,>0A,>90,2
       BYTE 3,>86,>0D,>90,3,1,>9F,0
       DATA 0
*
* PLAYING SCREEN TITLES
*
BL1    DATA 25,196
       TEXT 'LEVEL  SCORE 00000  TO GO'
BL2    DATA 4,235
       TEXT 'BEST'
*
BTX    MOV  R11,@RET     PUT TEXT ON SCREEN
       LI   R1,BL1
       BL   @DSP
       LI   R1,BL2
       BL   @DSP
       MOV  @LV,R1       GET SPEED LEVEL
       AI   R1,>31       CONVERT TO ASCII
       SWPB R1
       LI   R0,230
       BLWP *R9
       MOV  @RET,R11
       B    *R11
*
```

Disk 78. Contents of file BTHU;S

```
       TITL '9900 BREAKTHRU! V2.0 ( EDASM/MM)'
       IDT  'BRKTHRU'
*
*  9900 BREAKTHRU FOR EDITOR/ASSEMBLER AND MINIMEMORY
*
*  REV 84/06/16  08:20
*
*  PLACED IN THE PUBLIC DOMAIN, APRIL 4, 1987
*
*
       DEF  BTHU,SLOAD,SFIRST,SLAST
       REF  KSCAN,VSBW,VMBW,VSBR,VMBR,VWTR
*
*  PROGRAM FILE ENTRY
*
SLOAD
SFIRST JMP  BTHU
*
*  WORKING STORAGE
*
HS     DATA 0,0,0,0      HIGH SCORES FOR EACH LEVEL
*
RET    BSS  2            RETURN ADDRESS STORAGE
*
LV     BSS  2            SELECTED LEVEL
CS     BSS  2            CURRENT SCORE
BCT    BSS  2            BALL COUNT
BKL    BSS  2            BRICKS LEFT ON BOARD
BVL    BSS  2            BALL X,Y VELOCITY
STBVL  BSS  2            STARTING BALL VELOCITY
INBVL  BSS  2            BALL VELOCITY INCREMENT
*
*  ENTRY POINT
*
BTHU   JMP  START
*
       COPY "DSK1.BT1"   MAIN PROGRAM
       COPY "DSK1.BT2"   DISPLAY SETUP
       COPY "DSK1.BT3"   TITLES AND SOUND TABLES
       COPY "DSK1.BT4"   SUPPORT SUBROUTINES
*
SLAST  END
```

Disk 78. Contents of file BTHUDOC

9900 BREAK-THRU!

A Classic Arcade Game for the TI 99/4(A) Home Computer.

Hardware needed : Joystick #1, Disk Drive, and either Mini-Memory or Editor/Assembler + 32K.

You must move your paddle so that the ball touches it and bounces back to the rows of colored bricks. Each brick that is hit disappears and your score increases. If you get the ball's angle and speed just right, the ball may BREAK THRU several rows of bricks and bounce back and forth in the gap, causing your score to increase rapidly. If you miss the ball with your paddle, you lose it. The next ball is faster, so you must be alert. The game starts with six balls and ends when all balls have been lost. If you clear the board, you get an extra ball and a new board appears.

LOADING : For Mini-Memory or Editor/Assembler, select the Load and Run option. Enter the file name of DSKn.BTHU.

STARTUP : Select the Load Program option and enter the program name of BTHU. When the title screen appears, press any key to continue. When the game title screen appears, select the Difficulty Level by typing a digit between 1 and 4. A ball will appear against the level you have chosen, and the program will wait for a key to be pressed.

GAME BOARD : The screen consists of a wall on three sides, a paddle, eight rows of bricks in two groups, and displays of difficulty level, scores, and balls remaining. Although only 5 balls are displayed, the correct count is maintained internally. During play, the ball will move around the board, bouncing off the walls, paddle, or bricks. The ball is not affected by the displays in the gap between brick groups.

PLAY : Play begins when the fire button is pressed to launch the first ball from a random location by the left wall. The object of the game is to keep the ball in flight by bouncing it off the paddle so that all the bricks are eliminated. When the ball hits the paddle at the bottom of the screen it is reflected upward unless :

a) the paddle is moving. In this case, the ball is "sliced" and its direction is altered;

b) the ball hits the nearer end of the paddle. In this case the ball bounces back from whence it came.

Joystick 1 is used to move the paddle from side to side. Pressing the fire button while moving the paddle doubles the paddle speed. This feature is essential at higher levels. If the ball misses the paddle and drops off the bottom of the screen, it is lost. When the ball hits a wall, it is reflected. When it hits a brick, the brick disappears and the ball is also reflected.

SCORING : The bricks have values of 1 to 8 depending on the row in which they occur. The total value of a board is 1008 points. When the board is cleared of bricks it is renewed, another ball is awarded, and play resumes at a higher speed. Best scores are maintained for each level.

STRATEGY : Since each lost ball results in an increase in playing speed, you should try to avoid losing them for as long as possible. By controlling the angle of the ball's motion with the paddle, try to break through a narrow opening in the lower group of bricks so that the ball will bounce in the gap as long as possible. At levels 3 and 4, great skill is required just to avoid losing the ball!!

GAME END : When all balls have been lost, the game is over. If your score is the new best score for that level, it is saved. The following keys are recognized at this time :

REDO Play another game at this level.

BACK Select another level.

QUIT Return to the Master Title Screen.

This program, in source and object form, was contributed to the Public Domain at the Boston TI Faire, April 4, 1987.

Disk 78. Contents of file GOTHIC;C

```
/* Gothic Banner Printer
**
** Adapted to c99 by Clint Pulley
**
** These characters are displayed:
**
**  #.(@!$);-,?:'" A-Z a-z 0-9
**
** All others become blanks.
**
*/
#include "dsk1.stdio"
#include "dsk1.gottab1"
#include "dsk1.gottab2"
#include "dsk1.gottab3"
char outbuf[133];
char line[133];
char pch[]="0 ";
char fn[25];
int  out;
int  half; /* Set to halve display */
/**/
main ()
{ puts("Gothic Banner Printer v1.0\n\n");
  out=getfn("Gothic output","w132");
  half=0;
  puts("Half size? (y/n) [n] ");
  gets(line);
  if((*line|32)=='y') ++half;
  puts("\nInput text for banner:\n");
  while (gets(line))
    dotext(line);
  puts("\f");
}
/*
** Convert text to gothic letters and print
*/
dotext(text) char *text;
{
  char  *tp;
  int  c;
  for (tp = text; (c = *tp++) ;)
  { gothic(c);
    poll(1);
  }
}
/*
** Process the character
```

```
*/
gothic(character) int character;
{ char *gp;
  char *op;
  int  i;
  int  byte;
  int  index;
  int  lhalf; /* Line half flag */
  int  chalf; /* Column half flag */

  index = 0;
  lhalf = chalf = 0;
  gp = gottab[character & 127];
  op = outbuf;
  for (;;) {
    if ((i = *gp++ ) >= 254)
    { *op = 0;
      lhalf = ~lhalf;
      chalf = 0;
      if (half)
      { if (lhalf)
        { outbuf[66] = 0;
          fputs(outbuf,out);
        }
      }
      else
        fputs(outbuf,out);

      op = outbuf;
      if (i == 255)
        break;
    }
    else
    { byte = pch[- (index = (-1 - index))];
      while (--i >= 0)
      { if (half)
        { chalf = ~chalf;
          if (chalf)
            *op++ = byte;
        }
        else
          *op++ = byte;
      }
      if (op >= &outbuf[132])
      { puts("\nLine too long");
        exit(1);
      }
    }
  }
}
```

```
/* getfn returns unit # or 0 if null name entered
*/
getfn(text,m) char *text,*m;
{ int unit;
  unit=0;
  while(1)
  { puts(text);
    puts(" filename?");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
```

Disk 78. Contents of file GOTTAB1

```
/*
* "#"
*/
char c01[] = {
254,
254,
47,21,254,
46,23,254,
45,25,254,
44,27,254,
43,29,254,
42,8,21,2,254,
41,8,23,2,254,
40,8,25,2,254,
39,8,27,2,254,
38,8,29,2,254,
37,8,29,4,254,
25,19,29,16,254,
25,18,29,17,254,
36,8,27,8,254,
37,8,25,8,254,
38,8,23,8,254,
39,8,21,8,254,
40,8,20,7,254,
41,5,24,4,254,
254,
255 };
/*
* "."
*/
char c02[] = {
254,
254,
30,1,254,
30,3,254,
29,6,254,
27,10,254,
25,13,254,
25,13,254,
26,10,254,
28,6,254,
30,3,254,
254,
255 };
/*
* "("
*/
char c03[] = {
```

```
254,
254,
40,1,48,1,254,
39,52,254,
38,54,254,
37,56,254,
36,58,254,
35,60,254,
34,2,58,2,254,
33,2,60,2,254,
32,2,62,2,254,
31,2,64,2,254,
30,2,66,2,254,
29,2,68,2,254,
28,2,70,2,254,
27,2,72,2,254,
26,2,74,2,254,
254,
255 };
/*
* "@"
*/
char c04[] = {
254,
254,
73,1,16,1,254,
38,21,14,19,254,
37,23,13,20,254,
36,25,11,22,254,
35,27,9,24,254,
34,29,8,25,254,
33,31,6,27,254,
32,10,21,2,4,9,18,2,254,
31,10,23,2,2,9,17,5,254,
30,10,25,11,16,8,254,
29,10,25,11,16,10,254,
28,10,24,12,15,13,254,
27,10,22,14,15,15,254,
26,10,19,17,15,17,254,
25,10,16,19,17,18,254,
26,8,12,22,18,14,254,
27,6,9,23,21,12,254,
28,4,6,25,23,10,254,
29,2,3,23,28,9,254,
30,23,32,8,254,
29,20,36,7,254,
27,17,41,7,254,
25,15,46,5,254,
25,11,50,3,254,
25,9,1,19,33,2,254,
25,8,3,19,33,1,254,
```

```
25,7,5,19,254,
25,7,6,19,254,
27,5,7,19,254,
254,
255 };
/*
* "!"
*/
char c05[] = {
254,
254,
30,1,63,4,254,
30,3,62,6,254,
29,6,60,8,254,
27,10,40,27,254,
25,13,4,63,254,
25,13,4,63,254,
26,10,14,54,254,
28,6,31,39,254,
30,3,49,21,254,
254,
255 };
/*
* "$"
*/
char c06[] = {
254,
254,
35,3,20,9,254,
37,4,15,14,254,
38,5,11,19,254,
39,5,10,9,10,2,254,
39,6,9,8,10,5,254,
39,7,7,8,10,7,254,
39,7,7,8,9,10,254,
25,64,254,
25,64,254,
39,8,6,8,7,9,254,
38,9,6,8,7,8,254,
25,64,254,
25,64,254,
35,10,8,8,8,7,254,
37,7,9,8,8,7,254,
38,5,9,8,10,6,254,
40,2,9,9,11,5,254,
42,18,12,5,254,
45,14,15,4,254,
254,
255 };
/*
```

```
* ")"
*/
char c07[] = {
254,
254,
25,2,76,2,254,
26,2,74,2,254,
27,2,72,2,254,
28,2,70,2,254,
29,2,68,2,254,
30,2,66,2,254,
31,2,64,2,254,
32,2,62,2,254,
33,2,60,2,254,
34,2,58,2,254,
35,60,254,
36,58,254,
37,56,254,
38,54,254,
39,52,254,
254,
255 };
/*
* ";"
*/
char c08[] = {
254,
254,
50,1,254,
50,2,22,1,254,
51,2,21,3,254,
38,1,12,3,19,6,254,
38,2,10,5,16,10,254,
39,3,7,7,13,13,254,
40,17,12,13,254,
41,15,14,10,254,
42,13,17,6,254,
44,10,20,3,254,
254,
255 };
/*
* "-"
*/
char c09[] = {
254,
254,
49,5,254,
51,6,254,
52,7,254,
52,8,254,
52,8,254,
```

```
52,8,254,
52,8,254,
52,8,254,
52,8,254,
52,8,254,
52,8,254,
52,8,254,
52,8,254,
52,8,254,
53,7,254,
55,6,254,
254,
255 };
/*
* ","
*/
char c10[] = {
254,
254,
50,1,254,
50,2,254,
51,2,254,
38,1,12,3,254,
38,2,10,5,254,
39,3,7,7,254,
40,17,254,
41,15,254,
42,13,254,
44,10,254,
254,
255 };
/*
* "?"
*/
char c11[] = {
254,
254,
92,1,254,
76,18,254,
75,20,254,
74,22,254,
73,24,254,
72,26,254,
73,26,254,
30,1,22,7,14,7,17,2,254,
30,3,16,15,11,5,19,2,254,
29,6,11,21,9,3,19,4,254,
27,10,7,24,9,1,19,6,254,
25,13,4,28,26,8,254,
25,13,3,3,12,15,24,10,254,
```

```
26,10,4,2,16,14,22,10,254,
28,6,26,12,21,10,254,
30,3,29,11,19,10,254,
32,1,30,11,17,10,254,
64,10,16,10,254,
65,34,254,
66,32,254,
66,31,254,
67,29,254,
67,28,254,
254,
255 };
/*
* ":"
*/
char c12[] = {
254,
254,
30,1,31,1,254,
30,3,29,3,254,
29,6,26,6,254,
27,10,22,10,254,
25,13,19,13,254,
25,13,19,13,254,
26,10,22,10,254,
28,6,26,6,254,
30,3,29,3,254,
254,
255 };
/*
* "'"
*/
char c13[] = {
254,
254,
99,1,254,
99,2,254,
86,1,12,3,254,
86,2,10,5,254,
87,3,7,7,254,
88,17,254,
89,15,254,
90,13,254,
92,10,254,
254,
255 };
/*
* """
*/
char c14[] = {
254,
```

```
254,
94,6,254,
92,10,254,
90,13,254,
89,15,254,
88,17,254,
87,3,7,7,254,
86,2,10,5,254,
86,1,12,3,254,
99,2,254,
99,1,254,
254,
94,6,254,
92,10,254,
90,13,254,
89,15,254,
88,17,254,
87,3,7,7,254,
86,2,10,5,254,
86,1,12,3,254,
99,2,254,
254,
255 };
/*
* "a"
*/
char c15[] = {
254,
254,
32,16,12,2,254,
31,17,13,2,254,
30,19,13,2,254,
29,20,14,2,254,
28,22,14,2,254,
27,8,14,1,15,2,254,
26,8,15,2,13,4,254,
25,8,17,1,12,6,254,
26,6,18,2,10,8,254,
27,4,20,1,9,8,254,
28,3,20,2,7,8,254,
29,3,20,1,6,8,254,
30,36,254,
29,36,254,
28,36,254,
27,36,254,
26,36,254,
25,8,254,
27,5,254,
254,
255 };
```

```
/*
* "b"
*/
char c16[] = {
254,
254,
90,1,254,
89,3,254,
32,61,254,
31,63,254,
30,65,254,
29,66,254,
28,66,254,
27,8,28,2,24,3,254,
26,8,30,2,24,1,254,
25,8,32,2,254,
26,6,32,4,254,
27,4,32,6,254,
28,2,32,8,254,
29,2,30,8,254,
30,2,28,8,254,
31,36,254,
32,34,254,
33,32,254,
34,30,254,
254,
255 };
/*
* "c"
*/
char c17[] = {
254,
254,
37,21,254,
36,23,254,
35,25,254,
34,27,254,
33,29,254,
32,8,21,2,254,
31,8,23,2,254,
30,8,25,2,254,
29,8,27,2,254,
28,8,29,2,254,
27,8,29,4,254,
26,8,29,6,254,
25,8,29,8,254,
26,8,27,8,254,
27,8,25,8,254,
28,8,23,8,254,
29,8,21,8,254,
30,8,20,7,254,
```

```
31,5,24,4,254,
254,
255 };
/*
* "d"
*/
char c18[] = {
254,
254,
32,28,254,
31,30,28,1,254,
30,32,25,5,254,
29,34,22,9,254,
28,36,19,12,254,
27,8,28,2,16,13,254,
26,8,30,2,13,13,254,
25,8,32,2,10,13,254,
26,6,34,2,7,13,254,
27,4,36,2,4,13,254,
28,2,38,2,1,13,254,
29,2,38,13,254,
30,2,35,13,254,
29,49,254,
28,48,254,
27,47,254,
26,46,254,
25,45,254,
27,4,254,
254,
255 };
/*
* "e"
*/
char c19[] = {
254,
254,
37,21,254,
36,23,254,
35,25,254,
34,27,254,
33,29,254,
32,8,7,2,12,2,254,
31,8,9,2,12,2,254,
30,8,11,2,12,2,254,
29,8,13,2,12,2,254,
28,8,15,2,12,2,254,
27,8,17,2,10,4,254,
26,8,19,2,8,6,254,
25,8,21,2,6,8,254,
26,8,21,2,4,8,254,
```

```
27,8,21,2,2,8,254,
28,8,21,10,254,
29,8,21,8,254,
30,8,21,6,254,
31,5,24,4,254,
254,
255 };
/*
* "f"
*/
char c20[] = {
254,
254,
60,1,254,
60,3,254,
60,5,254,
60,7,254,
28,1,31,8,254,
27,59,254,
26,61,254,
25,63,254,
26,63,254,
27,63,254,
28,1,31,8,21,2,254,
60,8,22,2,254,
61,7,21,4,254,
63,5,20,6,254,
65,3,19,8,254,
67,1,13,13,254,
81,12,254,
81,11,254,
81,10,254,
254,
255 };
/*
* "g"
*/
char c21[] = {
254,
254,
9,6,17,28,254,
8,7,16,30,254,
7,8,15,32,254,
6,9,14,34,254,
5,10,13,36,254,
4,7,16,8,28,2,254,
3,7,16,8,30,2,254,
2,7,16,8,32,2,254,
1,7,18,6,32,4,254,
0,7,20,4,32,6,254,
1,5,22,2,32,8,254,
```

```
2,3,24,2,30,8,254,
3,2,25,2,28,8,254,
4,63,254,
5,61,254,
6,59,254,
7,57,254,
254,
255 };
/*
* "h"
*/
char c22[] = {
254,
254,
90,1,254,
28,1,60,3,254,
27,66,254,
26,68,254,
25,70,254,
26,68,254,
27,66,254,
28,1,33,2,25,3,254,
64,2,24,1,254,
65,2,254,
64,4,254,
63,6,254,
62,8,254,
28,1,32,8,254,
27,41,254,
26,41,254,
25,41,254,
26,39,254,
27,37,254,
254,
255 };
/*
* "i"
*/
char c23[] = {
254,
254,
66,1,9,1,254,
29,39,7,3,254,
28,41,5,5,254,
27,43,3,7,254,
26,43,5,5,254,
25,43,7,3,254,
26,7,33,1,9,1,254,
27,5,254,
28,3,254,
```

```
29,2,254,
254,
255 };
/*
* "j"
*/
char c24[] = {
254,
254,
8,7,254,
7,8,254,
6,9,254,
5,10,254,
4,11,254,
3,7,254,
2,7,254,
1,7,254,
0,7,254,
1,5,254,
2,3,254,
3,2,254,
4,2,60,1,9,1,254,
5,63,7,3,254,
6,63,5,5,254,
7,63,3,7,254,
8,61,5,5,254,
9,59,7,3,254,
254,
255 };
/*
* "k"
*/
char c25[] = {
254,
254,
90,1,254,
28,1,60,3,254,
27,66,254,
26,68,254,
25,70,254,
26,68,254,
27,66,254,
28,1,18,2,14,2,24,3,254,
48,1,15,2,24,1,254,
47,3,15,2,254,
46,4,14,4,254,
45,6,12,6,254,
44,7,11,8,254,
28,1,14,9,9,8,254,
27,23,1,1,8,8,254,
26,23,2,16,254,
```

```
25,23,4,14,254,
26,21,5,13,254,
27,19,7,11,254,
254,
255 };
/*
* "l"
*/
char c26[] = {
254,
254,
28,1,62,1,254,
27,66,254,
26,68,254,
25,70,254,
26,68,254,
27,66,254,
254,
255 };
/*
* "m"
*/
char c27[] = {
254,
254,
63,1,254,
28,1,34,3,254,
27,41,254,
26,44,254,
25,45,254,
26,42,254,
27,39,254,
28,1,33,2,254,
63,2,254,
64,2,254,
65,2,254,
64,4,254,
63,6,254,
28,1,33,8,254,
27,42,254,
26,42,254,
25,42,254,
26,40,254,
27,38,254,
28,1,35,2,254,
65,2,254,
64,4,254,
63,6,254,
28,1,33,8,254,
27,42,254,
```

```
26,42,254,
25,42,254,
26,40,254,
27,38,254,
254,
255 };
/*
* "n"
*/
char c28[] = {
254,
254,
63,1,254,
28,1,34,3,254,
27,41,254,
26,44,254,
25,45,254,
26,42,254,
27,39,254,
28,1,33,2,254,
63,2,254,
64,2,254,
65,2,254,
64,4,254,
63,6,254,
28,1,33,8,254,
27,42,254,
26,42,254,
25,42,254,
26,40,254,
27,38,254,
254,
255 };
/*
* "o"
*/
char c29[] = {
254,
254,
34,26,254,
33,28,254,
32,30,254,
31,32,254,
30,34,254,
29,8,26,2,254,
28,8,28,2,254,
27,8,30,2,254,
26,8,30,4,254,
25,8,30,6,254,
26,6,30,8,254,
27,4,30,8,254,
```

```
28,2,30,8,254,
29,2,28,8,254,
30,2,26,8,254,
31,34,254,
32,32,254,
33,30,254,
34,28,254,
254,
255 };
/*
* "p"
*/
char c30[] = {
254,
254,
4,1,60,1,254,
3,3,58,3,254,
2,66,254,
1,68,254,
0,70,254,
1,67,254,
2,64,254,
3,3,21,8,28,2,254,
4,1,21,8,30,2,254,
25,8,32,2,254,
26,6,32,4,254,
27,4,32,6,254,
28,2,32,8,254,
29,2,30,8,254,
30,2,28,8,254,
31,36,254,
32,34,254,
33,32,254,
34,30,254,
254,
255 };
/*
* "q"
*/
char c31[] = {
254,
254,
32,28,254,
31,30,254,
30,32,254,
29,34,254,
28,36,254,
27,8,28,2,254,
26,8,30,2,254,
25,8,32,2,254,
```

```
26,6,32,4,254,
27,4,32,6,254,
28,2,32,8,254,
4,1,24,2,30,8,254,
3,3,24,2,28,8,254,
2,65,254,
1,65,254,
0,65,254,
1,63,254,
2,61,254,
3,3,254,
254,
255 };
/*
* "r"
*/
char c32[] = {
254,
254,
63,1,254,
28,1,34,3,254,
27,41,254,
26,44,254,
25,45,254,
26,42,254,
27,39,254,
28,1,33,2,254,
63,2,254,
64,2,254,
65,2,254,
64,4,254,
63,6,254,
62,8,254,
61,8,254,
57,11,254,
57,10,254,
57,9,254,
57,8,254,
254,
255 };
/*
* "s"
*/
char c33[] = {
254,
254,
25,3,20,9,254,
27,4,15,14,254,
28,5,11,19,254,
29,5,10,9,10,2,254,
29,6,9,8,10,5,254,
```

```
29,7,7,8,10,7,254,
29,7,7,8,9,10,254,
29,8,6,8,8,10,254,
29,8,6,8,8,9,254,
29,8,6,8,7,9,254,
28,9,6,8,7,8,254,
27,9,7,8,7,8,254,
26,10,7,8,7,8,254,
25,10,8,8,8,7,254,
27,7,9,8,8,7,254,
28,5,9,8,10,6,254,
30,2,9,9,11,5,254,
32,18,12,5,254,
35,14,15,4,254,
254,
255 };
/*
* "t"
*/
char c34[] = {
254,
254,
68,1,254,
68,2,254,
68,3,254,
68,4,254,
68,5,254,
68,6,254,
68,7,254,
29,54,254,
28,56,254,
27,58,254,
26,60,254,
25,62,254,
26,6,36,7,254,
27,4,38,6,254,
28,2,40,5,254,
29,2,40,4,254,
30,2,40,3,254,
73,2,254,
254,
255 };
```

Disk 78. Contents of file GOTTAB2

```
/*
* "u"
*/
char c35[] = {
254,
254,
65,1,254,
64,4,254,
29,41,254,
28,41,254,
27,41,254,
26,41,254,
25,41,254,
26,6,254,
27,4,254,
28,2,254,
29,2,254,
30,2,33,1,254,
30,3,31,4,254,
29,41,254,
27,42,254,
25,43,254,
25,42,254,
27,39,254,
29,3,254,
254,
255 };
/*
* "v"
*/
char c36[] = {
254,
254,
65,1,254,
64,4,254,
33,37,254,
32,37,254,
31,37,254,
30,37,254,
29,37,254,
28,8,30,1,254,
27,8,32,1,254,
26,8,33,2,254,
25,8,33,4,254,
26,6,33,6,254,
27,4,33,8,254,
29,2,32,10,254,
30,4,27,13,254,
```

```
32,4,22,15,254,
34,6,14,18,254,
36,34,254,
39,29,254,
42,23,254,
254,
255 };
/*
* "w"
*/
char c37[] = {
254,
254,
66,1,254,
30,38,254,
29,40,254,
28,42,254,
27,42,254,
26,42,254,
25,8,33,1,254,
26,6,254,
27,4,254,
28,2,254,
29,2,35,1,254,
30,38,254,
29,40,254,
28,42,254,
27,42,254,
26,42,254,
25,8,33,1,254,
27,4,254,
28,2,254,
29,2,35,1,254,
30,38,254,
31,38,254,
32,38,254,
33,36,254,
34,34,254,
254,
255 };
/*
* "x"
*/
char c38[] = {
254,
254,
25,1,41,1,254,
26,2,39,2,254,
27,3,37,3,254,
26,5,35,4,254,
```

```
25,6,34,5,254,
28,6,29,6,254,
33,5,22,8,254,
36,5,10,16,254,
40,26,254,
34,30,254,
31,30,254,
29,27,254,
28,17,9,5,254,
27,9,21,5,254,
26,6,29,6,254,
25,5,34,6,254,
25,4,35,5,254,
25,3,37,3,254,
26,2,39,2,254,
254,
255 };
/*
* "y"
*/
char c39[] = {
254,
254,
65,1,254,
64,4,254,
7,8,14,41,254,
6,9,13,41,254,
5,10,12,41,254,
4,11,11,41,254,
3,12,10,41,254,
2,7,17,4,254,
1,7,19,2,254,
0,7,21,2,254,
1,5,23,2,254,
2,3,25,2,33,1,254,
3,2,26,2,31,4,254,
4,66,254,
5,64,254,
6,62,254,
7,60,254,
254,
255 };
/*
* "z"
*/
char c40[] = {
254,
254,
25,3,34,1,254,
25,5,32,3,254,
26,6,30,5,254,
```

```
26,5,1,2,27,8,254,
27,5,3,2,24,9,254,
27,5,5,2,22,8,254,
27,6,6,2,20,7,254,
27,6,9,2,17,7,254,
27,7,10,2,15,7,254,
27,7,12,2,13,7,254,
27,7,15,2,10,7,254,
27,7,17,2,8,7,254,
27,7,20,2,6,6,254,
27,7,22,2,4,6,254,
26,8,24,2,3,5,254,
25,9,26,2,1,5,254,
26,8,28,7,254,
28,5,32,4,254,
30,3,34,3,254,
254,
255 };
/*
* "A"
*/
char c41[] = {
254,
254,
25,4,254,
27,4,254,
28,5,254,
29,5,254,
29,6,254,
29,7,254,
29,7,32,2,254,
28,9,32,3,254,
27,15,28,4,254,
27,8,5,7,24,5,254,
26,8,11,8,18,7,254,
26,8,17,8,12,8,254,
25,8,24,8,6,9,254,
25,8,27,21,16,1,254,
25,8,27,22,16,2,254,
25,8,27,8,4,10,17,3,254,
26,7,27,8,10,10,11,4,254,
28,6,26,8,16,9,6,5,254,
31,4,25,8,22,15,254,
35,2,23,8,27,10,254,
60,8,23,14,254,
60,8,16,20,254,
60,8,9,26,254,
60,8,2,31,254,
60,38,254,
56,36,254,
```

```
49,36,254,
42,36,254,
35,36,254,
31,34,254,
29,28,254,
27,23,254,
26,17,254,
25,12,254,
25,8,254,
25,7,254,
25,7,254,
26,6,254,
28,5,254,
254,
255 };
/*
* "B"
*/
char c42[] = {
254,
254,
25,1,60,4,254,
26,3,59,6,254,
27,3,61,6,254,
27,5,26,1,33,7,254,
28,5,26,3,31,7,254,
28,6,26,5,29,7,254,
28,7,26,6,27,8,254,
29,7,25,7,26,8,254,
29,7,25,8,25,8,254,
29,8,24,9,23,8,254,
29,8,24,10,22,8,254,
29,9,22,11,22,8,254,
29,64,1,8,254,
29,9,1,55,1,7,254,
29,10,1,55,1,7,254,
29,10,3,54,2,5,254,
29,11,4,54,3,3,254,
29,11,7,54,2,2,254,
29,11,33,2,18,12,254,
29,12,32,2,18,12,254,
29,12,32,2,18,12,254,
29,12,32,2,18,11,254,
29,12,32,2,18,11,254,
29,12,31,3,17,12,254,
29,11,31,4,17,11,254,
29,11,30,5,17,10,254,
29,11,29,6,16,11,254,
28,12,28,7,16,10,254,
28,11,28,8,15,11,254,
27,12,27,9,15,10,254,
```

```
27,12,25,8,1,2,14,10,254,
26,12,25,9,1,2,14,9,254,
25,13,23,10,3,2,12,9,254,
26,12,21,11,4,3,9,10,254,
28,9,20,13,4,5,6,10,254,
30,7,18,14,6,19,254,
31,5,17,15,8,17,254,
33,3,14,17,10,14,254,
36,3,5,21,13,11,254,
38,25,18,5,254,
41,20,254,
254,
255 };
/*
* "C"
*/
char c43[] = {
254,
254,
58,9,254,
52,21,254,
47,31,254,
44,37,254,
41,43,254,
39,47,254,
37,23,5,2,10,11,254,
35,20,11,4,12,8,254,
33,18,16,5,14,5,254,
32,16,19,7,13,1,1,3,254,
31,15,21,8,13,1,2,3,254,
30,14,22,10,13,1,3,2,254,
29,62,3,2,254,
28,13,1,50,3,2,254,
27,12,4,50,4,1,254,
26,12,6,50,254,
26,11,8,50,254,
26,10,10,50,254,
25,10,59,4,254,
25,9,59,7,254,
25,8,59,10,254,
25,8,58,14,254,
25,7,58,14,254,
25,7,57,14,254,
25,6,57,14,254,
25,6,56,14,254,
25,5,56,14,254,
25,5,55,14,254,
25,5,54,14,254,
26,4,54,13,254,
26,4,54,12,254,
```

```
27,3,54,12,254,
27,4,53,11,254,
28,3,54,9,254,
29,3,54,8,254,
30,2,55,7,254,
31,2,55,6,254,
32,2,56,4,254,
254,
255 };
/*
* "D"
*/
char c44[] = {
254,
254,
25,1,60,4,254,
26,3,59,6,254,
27,3,61,6,254,
27,5,26,1,33,7,254,
28,5,26,3,31,7,254,
28,6,26,5,29,7,254,
28,7,26,6,27,8,254,
29,7,25,7,26,8,254,
29,7,25,8,25,8,254,
29,8,24,9,23,8,254,
29,8,24,10,22,8,254,
29,9,22,11,22,8,254,
29,64,1,8,254,
29,9,1,55,1,7,254,
29,10,1,55,1,7,254,
29,10,3,54,2,5,254,
29,11,4,54,3,3,254,
29,11,7,54,2,2,254,
29,11,53,12,254,
29,12,52,12,254,
29,12,51,12,254,
29,12,50,13,254,
29,12,50,12,254,
29,12,49,13,254,
29,11,49,14,254,
29,11,48,14,254,
29,11,47,14,254,
28,12,46,14,254,
28,11,45,15,254,
27,12,44,15,254,
27,12,42,16,254,
26,12,42,15,254,
25,13,40,16,254,
26,12,38,16,254,
28,9,37,17,254,
30,7,35,17,254,
```

```
31,5,33,18,254,
33,3,30,19,254,
36,3,23,20,254,
39,5,12,21,254,
44,28,254,
254,
255 };
/*
* "E"
*/
char c45[] = {
254,
254,
25,1,60,4,254,
25,3,60,6,254,
26,4,61,6,254,
27,4,27,1,33,7,254,
28,4,27,3,31,7,254,
29,5,26,5,29,7,254,
29,6,26,6,27,8,254,
29,7,25,7,26,8,254,
29,8,24,8,25,8,254,
29,9,23,9,23,8,254,
30,8,23,10,22,8,254,
30,9,21,11,22,8,254,
30,63,1,8,254,
29,65,1,7,254,
29,11,1,54,1,7,254,
29,11,3,53,2,5,254,
28,12,5,53,3,3,254,
27,12,9,53,2,2,254,
27,12,24,3,27,12,254,
26,13,25,4,24,12,254,
26,12,25,7,22,11,254,
26,12,24,10,20,10,254,
25,13,23,10,21,10,254,
25,12,23,10,22,10,254,
25,12,22,10,24,8,254,
25,12,21,10,25,8,254,
25,11,24,8,26,7,254,
25,11,26,6,26,7,254,
25,11,27,4,28,6,254,
25,11,29,2,28,6,254,
25,11,60,6,254,
25,11,61,5,254,
27,9,62,4,254,
29,7,63,4,254,
31,6,64,3,254,
33,5,65,2,254,
35,3,254,
```

```
37,2,254,
254,
255 };
/*
* "F"
*/
char c46[] = {
254,
254,
25,1,60,4,254,
25,3,60,6,254,
26,4,61,6,254,
27,4,27,1,33,7,254,
28,4,27,3,31,7,254,
29,5,26,5,29,7,254,
29,6,26,6,27,8,254,
29,7,25,7,26,8,254,
29,8,24,8,25,8,254,
29,9,23,9,23,8,254,
29,10,22,10,22,8,254,
30,9,21,11,22,8,254,
30,63,1,8,254,
31,63,1,7,254,
32,63,1,7,254,
33,63,2,5,254,
35,63,3,3,254,
37,64,2,2,254,
63,3,27,12,254,
64,4,24,12,254,
63,7,22,11,254,
62,10,20,10,254,
61,10,21,10,254,
60,10,22,10,254,
59,10,24,8,254,
58,10,25,8,254,
60,8,26,7,254,
62,6,26,7,254,
63,4,28,6,254,
64,2,29,6,254,
96,6,254,
97,5,254,
98,4,254,
99,4,254,
101,3,254,
254,
255 };
/*
* "G"
*/
char c47[] = {
254,
```

```
254,
58,9,254,
52,21,254,
47,31,254,
44,37,254,
41,43,254,
39,47,254,
37,23,5,2,10,11,254,
35,20,11,4,12,8,254,
33,18,16,5,14,5,254,
32,16,19,7,13,1,1,3,254,
31,15,21,8,13,1,2,3,254,
30,14,22,10,13,1,3,2,254,
29,62,3,2,254,
28,13,1,50,3,2,254,
27,12,4,50,4,1,254,
26,12,6,50,254,
26,11,8,50,254,
26,10,10,50,25436,7,3,3,7,14,254,
25,5,36,8,3,3,5,14,254,
26,4,36,8,4,3,3,14,254,
26,4,36,8,4,3,3,13,254,
27,3,36,8,4,3,3,12,254,
27,4,34,8,5,3,3,11,254,
28,3,34,8,6,3,2,10,254,
29,2,33,8,7,3,3,9,254,
30,2,31,8,8,3,4,8,254,
31,2,28,9,10,3,4,7,254,
33,1,25,10,12,3,4,6,254,
34,2,21,10,16,3,4,4,254,
36,2,16,11,20,3,4,2,254,
38,3,7,16,24,3,2,2,254,
41,19,31,2,254,
254,
255 };
/*
* "H"
*/
char c48[] = {
254,
254,
25,3,58,4,254,
27,4,57,6,254,
28,5,58,6,254,
28,7,23,1,33,7,254,
28,8,23,3,31,7,254,
29,8,23,5,29,7,254,
29,8,24,6,27,8,254,
29,9,23,7,26,8,254,
29,9,23,8,25,8,254,
```

```
28,9,24,9,23,8,254,
28,9,24,10,22,8,254,
28,8,24,11,22,8,254,
28,65,1,8,254,
27,8,1,58,1,7,254,
26,9,2,58,1,7,254,
26,8,5,57,2,5,254,
0,1,24,8,8,57,3,3,254,
1,2,22,8,12,35,13,12,254,
2,3,20,8,42,2,15,12,254,
3,3,19,8,43,2,14,11,254,
3,5,17,8,43,3,13,10,254,
4,5,16,8,42,5,12,10,254,
5,5,16,8,40,7,11,10,254,
6,6,15,7,39,9,11,8,254,
7,7,15,6,38,10,10,8,254,
8,8,16,4,36,12,10,7,254,
9,9,17,3,34,11,11,7,254,
10,11,51,10,13,6,254,
12,12,47,11,13,6,254,
13,68,15,6,254,
15,66,16,5,254,
18,62,18,4,254,
20,60,19,4,254,
22,58,21,3,254,
26,54,23,2,254,
75,6,254,
77,5,254,
80,3,254,
254,
255 };
/*
* "I"
*/
char c49[] = {
254,
254,
25,1,60,4,254,
25,3,60,6,254,
26,4,61,6,254,
27,4,27,1,33,7,254,
28,4,27,3,31,7,254,
29,5,26,5,29,7,254,
29,6,26,6,27,8,254,
29,7,25,7,26,8,254,
29,8,24,8,25,8,254,
29,9,23,9,23,8,254,
28,11,22,10,22,8,254,
28,11,21,11,22,8,254,
27,66,1,8,254,
26,68,1,7,254,
```

```
26,69,1,7,254,
25,71,2,5,254,
27,71,3,3,254,
254,
255 };
/*
* "J"
*/
char c50[] = {
254,
254,
34,13,254,
30,20,254,
28,24,254,
27,27,32,4,254,
27,15,3,10,33,6,254,
26,12,12,6,35,6,254,
26,9,18,5,8,1,25,7,254,
25,8,22,5,4,2,27,7,254,
25,7,25,8,29,7,254,
25,6,28,4,31,8,254,
26,4,64,8,254,
26,4,64,8,254,
27,4,46,8,8,8,254,
28,4,36,21,4,8,254,
29,4,30,29,1,8,254,
30,5,20,38,1,8,254,
32,62,1,7,254,
34,61,1,7,254,
36,36,16,8,2,5,254,
39,26,26,7,3,3,254,
44,14,36,6,3,2,254,
97,5,254,
254,
255 };
/*
* "K"
*/
char c51[] = {
254,
254,
25,3,58,4,254,
27,4,57,6,254,
28,5,58,6,254,
28,7,23,1,33,7,254,
28,8,23,3,31,7,254,
29,8,23,5,29,7,254,
29,8,24,6,27,8,254,
29,9,23,7,26,8,254,
29,9,23,8,25,8,254,
```

```
28,9,24,9,23,8,254,
28,9,24,10,22,8,254,
28,8,24,11,22,8,254,
28,65,1,8,254,
27,8,1,58,1,7,254,
26,9,2,58,1,7,254,
26,8,5,57,2,5,254,
25,8,8,57,3,3,254,
25,8,12,56,2,2,254,
25,8,25,2,16,2,15,12,254,
25,8,26,2,16,2,13,12,254,
25,8,26,3,16,2,12,11,254,
25,8,26,4,14,4,11,10,254,
25,8,26,5,11,7,10,10,254,
26,8,25,6,8,10,9,10,254,
27,7,24,8,5,13,9,8,254,
29,6,22,10,2,16,8,8,254,
32,4,19,10,1,17,11,7,254,
35,3,14,12,3,14,13,7,254,
44,20,4,11,16,6,254,
35,28,6,8,18,6,254,
31,31,8,5,21,6,254,
29,32,10,2,24,5,254,
28,30,40,4,254,
27,23,49,4,254,
26,16,59,3,254,
26,11,66,2,254,
25,9,254,
25,7,254,
25,6,254,
26,4,254,
27,3,254,
254,
255 };
/*
* "L"
*/
char c52[] = {
254,
254,
25,1,60,4,254,
25,3,60,6,254,
26,4,61,6,254,
27,4,27,1,33,7,254,
28,4,27,3,31,7,254,
28,6,26,5,29,7,254,
29,6,26,6,27,8,254,
30,6,25,7,26,8,254,
30,7,24,8,25,8,254,
31,7,23,9,23,8,254,
31,8,22,10,22,8,254,
```

```
31,8,21,11,22,8,254,
31,62,1,8,254,
30,9,1,54,1,7,254,
30,10,1,54,1,7,254,
29,11,3,53,2,5,254,
29,11,5,53,3,3,254,
28,11,9,53,2,2,254,
28,11,54,12,254,
27,12,53,10,254,
27,11,54,9,254,
26,12,55,7,254,
26,11,57,5,254,
26,10,59,4,254,
25,10,61,3,254,
25,10,62,3,254,
25,9,65,2,254,
25,9,254,
26,8,254,
26,8,254,
27,7,254,
27,7,254,
28,6,254,
29,5,254,
30,4,254,
32,3,254,
34,2,254,
36,2,254,
254,
255 };
/*
* "M"
*/
char c53[] = {
254,
254,
25,2,59,4,254,
27,3,58,6,254,
28,4,59,6,254,
29,4,25,1,33,7,254,
29,6,24,3,31,7,254,
29,7,24,5,29,7,254,
29,8,24,6,27,8,254,
29,9,23,7,26,8,254,
29,9,23,8,25,8,254,
29,10,22,9,23,8,254,
29,10,22,10,22,8,254,
28,11,21,11,22,8,254,
27,66,1,8,254,
25,69,1,7,254,
26,69,1,7,254,
```

```
28,68,2,5,254,
30,68,3,3,254,
34,67,2,2,254,
93,12,254,
92,12,254,
58,1,32,12,254,
59,3,28,12,254,
60,5,24,12,254,
61,6,21,12,254,
61,7,19,12,254,
61,8,17,12,254,
30,1,30,9,15,12,254,
29,3,29,10,13,12,254,
28,5,27,11,12,12,254,
27,67,254,
26,67,254,
25,67,254,
26,65,254,
27,63,254,
28,61,254,
29,5,25,10,19,3,254,
30,3,26,9,22,3,254,
31,1,28,8,24,3,254,
61,7,26,3,254,
62,6,27,4,254,
63,5,26,7,254,
64,5,25,9,254,
67,3,23,12,254,
30,1,39,1,22,11,254,
29,3,60,11,254,
28,5,59,10,254,
27,74,254,
26,75,254,
25,76,254,
26,75,254,
27,75,254,
28,74,254,
29,5,64,5,254,
30,3,66,4,254,
31,1,69,3,254,
254,
255 };
/*
* "N"
*/
char c54[] = {
254,
254,
25,2,34,3,31,2,254,
26,3,34,4,29,3,254,
27,4,33,5,28,4,254,
```

```
28,5,32,6,26,5,254,
28,6,31,7,25,6,254,
29,6,30,8,23,8,254,
29,7,28,10,21,10,254,
29,76,254,
29,76,254,
29,75,254,
29,10,47,17,254,
28,11,43,19,254,
28,12,39,20,254,
27,13,36,20,254,
25,15,33,20,254,
27,13,30,20,254,
29,11,27,20,254,
31,9,24,20,254,
32,8,21,20,254,
35,4,19,20,254,
37,1,17,20,17,1,254,
52,20,19,4,254,
49,20,21,7,254,
46,20,24,9,254,
43,20,27,11,254,
40,20,30,13,254,
37,20,33,15,254,
34,20,36,13,254,
31,20,39,12,254,
28,20,43,11,254,
27,18,46,10,254,
26,75,254,
25,76,254,
25,76,254,
25,11,58,7,254,
25,10,60,6,254,
26,8,62,6,254,
28,6,63,5,254,
30,4,65,4,254,
32,3,66,3,254,
254,
255 };
/*
* "O"
*/
char c55[] = {
254,
254,
58,9,254,
53,21,254,
47,31,254,
44,38,254,
41,43,254,
```

```
39,47,254,
37,23,6,2,9,11,254,
35,20,12,3,10,2,1,7,254,
33,18,16,5,9,2,4,5,254,
32,16,19,6,9,2,5,4,254,
31,15,21,7,9,2,6,3,254,
30,14,22,9,9,2,7,2,254,
29,58,254,
28,12,3,45,254,
27,12,5,45,254,
27,11,7,46,254,
26,11,10,45,254,
26,10,15,43,254,
25,10,58,3,254,
26,8,58,6,254,
28,5,58,10,254,
29,3,58,15,254,
30,2,58,14,254,
31,2,56,14,254,
33,2,53,15,254,
34,2,51,15,254,
36,2,47,16,254,
37,3,44,16,254,
39,4,40,16,254,
40,6,35,17,254,
42,7,29,18,254,
44,8,23,19,254,
46,11,14,21,254,
48,41,254,
51,36,254,
53,31,254,
56,25,254,
59,19,254,
254,
255 };
```

Disk 78. Contents of file GOTTAB3

```
/*
* "P"
*/
char c56[] = {
254,
254,
25,3,58,4,254,
27,4,57,6,254,
28,5,58,6,254,
28,7,23,1,33,7,254,
28,8,23,3,31,7,254,
29,8,23,5,29,7,254,
29,8,24,6,27,8,254,
29,9,23,7,26,8,254,
29,9,23,8,25,8,254,
28,9,24,9,23,8,254,
28,9,24,10,22,8,254,
28,8,24,11,22,8,254,
28,65,1,8,254,
27,8,1,58,1,7,254,
26,9,2,58,1,7,254,
26,8,5,57,2,5,254,
25,8,8,57,3,3,254,
25,8,12,56,2,2,254,
25,8,31,4,25,12,254,
25,8,30,7,23,11,254,
25,8,29,10,20,11,254,
25,8,28,13,18,11,254,
25,8,28,11,19,11,254,
26,8,26,10,21,11,254,
27,7,25,9,22,11,254,
29,6,24,8,23,11,254,
32,4,23,7,23,11,254,
35,3,20,7,24,10,254,
58,6,24,10,254,
58,5,24,10,254,
58,5,23,10,254,
59,3,23,10,254,
59,3,22,9,254,
59,3,20,10,254,
60,2,18,10,254,
61,2,15,10,254,
62,2,12,9,254,
64,18,254,
254,
255 };
/*
* "Q"
```

```
*/
char c57[] = {
254,
254,
58,9,254,
53,21,254,
47,31,254,
44,38,254,
41,43,254,
39,47,254,
37,23,6,2,9,11,254,
35,20,12,3,10,2,1,7,254,
33,18,16,5,9,2,4,5,254,
32,16,19,6,9,2,5,4,254,
31,15,21,7,9,2,6,3,254,
30,14,22,9,9,2,7,2,254,
29,58,254,
28,12,1,47,254,
27,12,3,47,254,
27,11,4,49,254,
26,11,5,4,1,45,254,
26,10,6,5,4,43,254,
25,10,7,6,45,3,254,
26,8,7,7,44,6,254,
28,5,7,9,42,10,254,
29,3,6,11,41,15,254,
30,2,4,12,42,14,254,
31,16,42,14,254,
30,15,43,15,254,
28,16,43,15,254,
27,15,43,16,254,
26,14,44,16,254,
25,13,1,4,40,16,254,
25,11,4,6,35,17,254,
25,9,8,7,29,18,254,
25,8,11,8,23,19,254,
25,8,13,11,14,21,254,
25,8,15,41,254,
26,7,18,36,254,
27,6,20,31,254,
28,5,23,25,254,
29,4,26,19,254,
30,4,30,9,254,
31,3,254,
33,2,254,
254,
255 };
/*
* "R"
*/
char c58[] = {
```

```
254,
254,
25,3,58,4,254,
27,4,57,6,254,
28,5,58,6,254,
28,7,23,1,33,7,254,
28,8,23,3,31,7,254,
29,8,23,5,29,7,254,
29,8,24,6,27,8,254,
29,9,23,7,26,8,254,
29,9,23,8,25,8,254,
28,9,24,9,23,8,254,
28,9,24,10,22,8,254,
28,8,24,11,22,8,254,
28,65,1,8,254,
27,8,1,58,1,7,254,
26,9,2,58,1,7,254,
26,8,5,57,2,5,254,
25,8,8,57,3,3,254,
25,8,12,56,2,2,254,
25,8,29,2,29,12,254,
25,8,29,2,29,11,254,
25,8,29,2,29,11,254,
25,8,30,2,28,10,254,
25,8,30,2,28,10,254,
26,8,29,2,28,9,254,
27,7,29,3,26,10,254,
29,6,28,4,25,10,254,
32,4,26,6,24,9,254,
35,3,23,8,22,10,254,
59,11,21,10,254,
57,14,19,10,254,
29,43,18,10,254,
27,43,1,2,16,10,254,
26,44,2,2,14,11,254,
25,43,5,3,10,12,254,
25,41,9,4,5,14,254,
25,37,14,20,254,
25,5,48,16,254,
25,4,51,12,254,
25,4,54,6,254,
26,3,254,
27,3,254,
254,
255 };
/*
* "S"
*/
char c59[] = {
254,
```

```
254,
25,3,254,
27,4,254,
29,4,254,
31,4,254,
32,5,254,
33,5,43,2,254,
34,5,40,5,254,
34,6,37,8,254,
35,6,34,11,254,
35,7,32,13,254,
36,7,30,15,254,
36,7,29,17,254,
36,8,27,19,254,
36,8,26,17,2,3,254,
36,9,11,2,11,16,6,3,254,
35,10,10,1,12,16,9,3,254,
35,10,9,1,12,16,10,5,254,
34,11,9,1,12,15,10,9,254,
33,12,10,1,10,15,10,14,254,
32,12,12,2,7,15,10,13,254,
31,13,15,2,4,14,11,11,254,
30,13,19,16,11,11,254,
28,14,22,13,12,9,254,
25,17,21,13,12,10,254,
26,14,23,12,12,11,254,
26,13,23,12,1,2,10,11,254,
27,11,24,12,4,2,6,12,254,
28,8,25,12,8,2,3,11,254,
29,6,26,12,11,13,254,
29,5,26,12,14,11,254,
30,2,28,12,14,11,254,
31,2,26,12,16,10,254,
32,2,24,12,17,10,254,
33,2,22,13,18,9,1,2,254,
34,3,19,13,20,8,4,2,254,
35,3,16,13,23,7,6,1,254,
36,4,11,14,27,6,6,1,254,
38,25,31,5,5,1,254,
40,20,36,4,3,1,254,
254,
255 };
/*
* "T"
*/
char c60[] = {
254,
254,
58,9,23,1,254,
52,21,18,2,254,
47,31,15,2,254,
```

```
44,37,13,3,254,
41,43,11,4,254,
39,47,10,4,254,
37,23,17,11,9,4,254,
35,20,27,8,8,4,254,
33,18,14,2,19,5,7,5,254,
32,16,18,3,20,3,6,6,254,
31,15,21,4,20,3,4,6,254,
30,14,23,5,21,2,3,7,254,
29,13,25,6,21,2,2,7,254,
28,13,26,7,21,10,254,
27,12,27,8,22,9,254,
26,60,9,10,254,
26,11,1,49,7,10,254,
26,10,3,49,5,10,254,
25,10,5,49,3,11,254,
25,9,7,49,1,11,254,
25,8,9,60,254,
25,8,57,11,254,
25,7,58,11,254,
25,7,57,11,254,
25,6,58,11,254,
25,6,58,11,254,
25,5,60,9,254,
25,5,60,9,254,
25,5,60,9,254,
26,4,61,8,254,
26,4,61,8,254,
27,3,62,8,254,
27,4,62,7,254,
28,3,62,7,254,
29,3,62,6,254,
30,2,63,6,254,
31,2,63,5,254,
32,2,64,4,254,
254,
255 };
/*
* "U"
*/
char c61[] = {
254,
254,
25,2,36,3,29,2,254,
26,4,34,5,28,3,254,
27,5,33,6,27,4,254,
28,5,33,6,26,5,254,
28,6,32,7,25,6,254,
28,7,31,8,24,7,254,
29,7,29,10,22,8,254,
```

```
29,76,254,
29,76,254,
29,75,254,
29,74,254,
28,73,254,
27,71,254,
27,12,254,
26,13,254,
25,14,254,
26,12,254,
27,11,254,
28,9,254,
29,7,254,
30,6,61,2,254,
31,4,63,4,254,
32,2,65,4,254,
33,2,64,5,254,
34,2,62,7,254,
35,70,254,
33,72,254,
31,74,254,
29,76,254,
27,77,254,
25,78,254,
26,13,57,7,254,
27,10,60,6,254,
28,7,63,5,254,
29,4,67,4,254,
30,2,71,2,254,
31,2,254,
254,
255 };
/*
* "V"
*/
char c62[] = {
254,
254,
25,2,36,3,31,2,254,
26,4,34,5,29,4,254,
27,5,33,6,28,4,254,
28,5,32,7,27,5,254,
28,6,30,9,26,6,254,
28,77,254,
29,76,254,
29,76,254,
29,76,254,
29,75,254,
29,74,254,
28,10,58,7,254,
27,11,59,6,254,
```

```
27,12,59,5,254,
26,13,61,4,254,
25,14,64,2,254,
26,12,254,
27,11,254,
28,9,254,
29,7,254,
30,6,254,
31,4,254,
32,2,254,
33,2,254,
34,2,51,9,254,
35,3,33,28,254,
36,65,254,
38,64,254,
40,63,254,
42,62,254,
45,33,18,8,254,
48,14,36,7,254,
100,5,254,
101,3,254,
102,2,254,
254,
255 };
/*
* "W"
*/
char c63[] = {
254,
254,
25,2,36,3,29,2,254,
26,4,34,5,28,3,254,
27,5,33,6,27,4,254,
28,5,33,6,26,5,254,
28,6,32,7,25,6,254,
28,7,31,8,24,7,254,
29,7,29,10,22,8,254,
29,76,254,
29,76,254,
29,75,254,
29,74,254,
28,73,254,
27,71,254,
27,13,254,
26,14,254,
25,15,254,
26,12,25,3,254,
27,11,26,5,254,
28,9,28,6,26,2,254,
29,7,30,6,26,4,254,
```

```
30,6,30,7,26,4,254,
31,4,31,8,25,5,254,
32,3,30,10,23,7,254,
33,72,254,
31,74,254,
29,76,254,
27,78,254,
25,80,254,
25,79,254,
26,8,28,10,23,8,254,
27,5,31,8,25,7,254,
28,2,34,7,26,6,254,
29,2,34,6,27,5,254,
30,2,34,6,28,4,254,
31,2,35,5,30,2,254,
32,3,36,3,254,
33,4,49,6,254,
34,7,41,14,254,
35,11,22,31,254,
36,65,254,
38,64,254,
40,63,254,
42,62,254,
45,33,18,8,254,
48,14,36,7,254,
52,6,42,5,254,
101,3,254,
102,2,254,
254,
255 };
/*
* "X"
*/
char c64[] = {
254,
254,
25,3,64,4,254,
26,5,63,5,254,
27,6,63,6,254,
27,7,62,7,254,
28,7,18,3,40,8,254,
28,8,18,5,37,9,254,
27,9,19,6,34,10,254,
27,9,20,7,31,11,254,
27,10,20,8,27,13,254,
26,11,21,9,22,15,254,
26,11,21,10,18,17,254,
25,12,21,12,12,20,254,
31,6,22,12,7,22,254,
36,6,17,12,3,24,254,
42,6,11,36,254,
```

```
48,6,4,33,254,
54,33,254,
58,24,254,
55,23,254,
51,23,254,
47,29,254,
43,29,4,6,254,
39,32,11,6,254,
35,36,17,6,254,
30,24,6,12,21,6,254,
28,21,13,10,21,12,254,
27,17,19,9,21,11,254,
26,15,24,8,20,11,254,
25,13,29,7,19,10,254,
25,11,33,6,19,9,254,
25,10,36,5,18,9,254,
25,9,40,3,17,8,254,
26,8,61,7,254,
27,7,62,7,254,
28,6,63,6,254,
31,5,63,5,254,
254,
255 };
/*
* "Y"
*/
char c65[] = {
254,
254,
25,1,33,3,33,2,254,
25,3,33,4,31,4,254,
26,4,32,5,30,5,254,
27,4,31,7,28,6,254,
0,2,25,6,29,8,27,7,254,
2,3,22,7,28,9,25,9,254,
3,4,21,7,26,11,23,10,254,
4,5,19,77,254,
5,5,18,77,254,
5,6,17,77,254,
6,6,16,76,254,
6,6,16,75,254,
6,7,14,73,254,
6,7,14,11,52,3,254,
6,8,12,12,54,3,254,
6,8,11,12,57,3,254,
6,9,12,10,58,4,254,
5,10,14,8,57,7,254,
5,10,16,5,58,9,254,
5,10,18,3,57,12,254,
4,11,20,3,55,11,254,
```

```
4,11,22,3,53,10,254,
3,11,25,3,50,11,254,
3,11,27,3,48,10,254,
2,12,29,3,46,10,254,
1,12,20,69,254,
0,13,16,72,254,
1,11,15,74,254,
2,9,14,76,254,
3,7,13,78,254,
4,5,12,80,254,
5,3,11,16,60,7,254,
6,3,8,16,63,6,254,
8,23,66,6,254,
10,18,71,5,254,
14,11,77,3,254,
254,
255 };
/*
* "Z"
*/
char c66[] = {
254,
254,
25,3,52,5,254,
27,3,46,14,254,
28,4,42,19,254,
29,5,39,5,6,11,254,
29,6,37,3,11,11,254,
30,7,35,2,13,12,254,
30,10,32,1,15,13,254,
30,11,47,15,254,
31,8,2,2,45,17,254,
31,9,3,2,43,16,254,
31,9,5,2,40,16,254,
31,10,6,2,11,5,22,15,254,
31,10,8,2,11,7,17,15,254,
31,11,9,2,10,8,15,14,254,
31,11,11,2,8,9,13,14,254,
31,11,13,2,6,9,13,14,254,
31,11,15,2,4,9,13,14,254,
31,12,16,2,2,9,13,13,254,
31,12,18,11,12,14,254,
31,12,20,9,12,14,254,
31,12,20,9,12,14,254,
31,12,20,9,12,14,254,
30,13,20,9,13,13,254,
30,13,20,10,12,13,254,
30,13,20,9,1,2,10,13,254,
29,14,20,9,3,2,9,12,254,
28,15,20,9,5,2,7,12,254,
28,15,20,9,7,2,5,12,254,
```

```
27,15,22,8,9,2,4,11,254,
26,16,24,7,10,2,2,11,254,
25,17,28,5,10,2,1,10,254,
26,15,46,12,254,
27,14,12,2,34,10,254,
29,12,13,2,35,9,254,
31,11,11,4,36,7,254,
33,10,9,5,38,6,254,
35,10,5,6,41,4,254,
37,18,44,3,254,
39,14,48,2,254,
254,
255 };
/*
* "0"
*/
char c67[] = {
254,
254,
35,44,254,
34,46,254,
33,48,254,
32,50,254,
31,52,254,
30,8,44,2,254,
29,8,46,2,254,
28,8,48,2,254,
27,8,48,4,254,
26,8,48,6,254,
25,8,48,8,254,
26,6,48,8,254,
27,4,48,8,254,
28,2,48,8,254,
29,2,46,8,254,
30,54,254,
31,52,254,
32,50,254,
33,48,254,
254,
255 };
/*
* "1"
*/
char c68[] = {
254,
254,
25,1,58,1,254,
26,2,57,1,254,
27,2,56,2,254,
28,60,254,
```

```
29,60,254,
30,58,254,
31,56,254,
32,54,254,
33,2,254,
34,2,254,
254,
255 };
/*
* "2"
*/
char c69[] = {
254,
254,
25,7,34,15,254,
26,6,1,4,30,15,254,
26,7,4,4,27,15,254,
27,7,7,3,25,15,254,
27,7,10,3,23,15,254,
28,7,12,2,33,4,254,
28,7,14,2,32,4,254,
27,7,17,2,29,6,254,
27,7,18,2,27,8,254,
26,7,21,2,24,8,254,
26,7,22,2,22,8,254,
25,7,24,2,20,8,254,
25,7,25,2,18,8,254,
25,6,27,2,16,8,254,
25,6,28,2,14,8,254,
25,6,29,22,254,
26,6,29,20,254,
27,6,29,18,254,
28,6,29,16,254,
254,
255 };
/*
* "3"
*/
char c70[] = {
254,
254,
35,1,45,1,254,
34,4,43,2,254,
33,7,41,3,254,
32,8,41,4,254,
31,8,19,1,22,5,254,
30,8,19,3,21,6,254,
29,8,19,5,20,7,254,
28,8,19,7,19,8,254,
27,8,19,9,18,8,254,
26,8,19,8,1,2,17,8,254,
```

```
25,8,19,8,3,2,16,8,254,
26,6,19,8,5,2,15,8,254,
27,4,19,8,7,2,14,8,254,
28,2,19,8,9,2,13,8,254,
29,2,17,8,11,2,11,9,254,
30,25,13,21,254,
31,23,15,19,254,
32,21,17,17,254,
33,19,19,15,254,
254,
255 };
/*
* "4"
*/
char c71[] = {
254,
254,
54,1,254,
54,3,254,
54,5,254,
55,6,254,
55,8,254,
55,10,254,
56,7,1,3,254,
56,7,3,3,254,
56,7,5,3,254,
57,7,6,3,254,
57,7,8,3,254,
57,7,10,3,254,
57,7,12,3,254,
56,7,15,3,254,
29,1,25,8,17,3,254,
28,3,24,7,20,3,254,
27,60,254,
26,63,254,
25,64,254,
26,63,254,
27,62,254,
28,3,24,7,254,
29,1,27,5,254,
254,
255 };
/*
* "5"
*/
char c72[] = {
254,
254,
35,1,15,38,254,
34,4,14,36,254,
```

```
33,7,13,8,20,6,254,
32,8,14,8,18,7,254,
31,8,16,8,16,7,254,
30,8,18,8,15,7,254,
29,8,20,8,14,6,254,
28,8,22,8,12,7,254,
27,8,24,8,11,7,254,
26,8,26,8,10,7,254,
25,8,28,8,9,7,254,
26,6,28,8,10,7,254,
27,4,28,8,11,7,254,
28,2,28,8,13,6,254,
29,2,26,8,14,6,254,
30,34,16,6,254,
31,32,17,6,254,
32,30,21,4,254,
33,28,24,3,254,
254,
255 };
/*
* "6"
*/
char c73[] = {
254,
254,
35,44,254,
34,46,254,
33,48,254,
32,50,254,
31,52,254,
30,8,25,2,17,2,254,
29,8,27,2,17,2,254,
28,8,29,2,17,2,254,
27,8,31,2,15,4,254,
26,8,31,4,13,6,254,
25,8,31,6,11,8,254,
26,6,31,8,9,8,254,
27,4,31,9,8,8,254,
28,2,31,8,9,8,254,
29,2,29,8,9,8,254,
30,37,9,8,254,
31,35,9,8,254,
32,33,10,7,254,
33,31,13,4,254,
254,
255 };
/*
* "7"
*/
char c74[] = {
254,
```

```
254,
25,2,54,1,254,
26,2,53,3,254,
27,2,52,5,254,
28,5,48,7,254,
29,12,40,8,254,
30,19,32,7,254,
31,23,27,7,254,
32,29,20,6,254,
33,31,17,6,254,
34,2,10,21,15,5,254,
35,2,16,17,12,5,254,
36,2,21,14,9,5,254,
64,11,8,4,254,
69,9,5,4,254,
73,7,4,4,254,
76,6,2,4,254,
79,9,254,
83,5,254,
86,3,254,
254,
255 };
/*
* "8"
*/
char c75[] = {
254,
254,
34,16,13,16,254,
33,19,10,18,254,
32,22,7,20,254,
31,24,5,22,254,
30,26,3,24,254,
29,8,17,3,1,8,16,2,254,
28,8,20,9,18,2,254,
27,8,21,8,20,2,254,
26,8,21,8,20,4,254,
25,8,21,8,20,6,254,
26,6,21,8,20,8,254,
27,4,21,8,20,8,254,
28,2,21,8,20,8,254,
29,2,19,9,19,8,254,
30,2,17,8,1,3,16,8,254,
31,25,3,25,254,
32,23,5,23,254,
33,21,7,21,254,
34,19,10,18,254,
254,
255 };
/*
```

```
* "9"
*/
char c76[] = {
254,
254,
34,1,20,25,254,
33,4,17,27,254,
32,7,14,29,254,
31,8,13,31,254,
30,8,13,33,254,
29,8,13,8,25,2,254,
28,8,13,8,27,2,254,
27,8,13,8,27,4,254,
26,8,13,8,27,6,254,
25,8,15,6,27,8,254,
26,6,17,4,27,8,254,
27,4,19,2,27,8,254,
28,2,21,2,25,8,254,
29,2,21,2,23,8,254,
30,2,21,2,21,8,254,
31,52,254,
32,50,254,
33,48,254,
34,46,254,
35,44,254,
254,
255 };
/*
* ""
*/
char c77[] = {
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
254,
```

```
254,
254,
255 };
 int gottab[1];
/* fake a pointer array! */
#asm
 RORG $-2
 DATA C77,C77,C77,C77,C77,C77,C77,C77
 DATA C77,C77,C77,C77,C77,C77,C77,C77
 DATA C77,C77,C77,C77,C77,C77,C77,C77
 DATA C77,C77,C77,C77,C77,C77,C77,C77
 DATA C77,C05,C14,C01,C06,C77,C77,C13
 DATA C03,C07,C77,C77,C10,C09,C02,C77
 DATA C67,C68,C69,C70,C71,C72,C73,C74
 DATA C75,C76,C12,C08,C77,C77,C77,C11
 DATA C04,C41,C42,C43,C44,C45,C46,C47
 DATA C48,C49,C50,C51,C52,C53,C54,C55
 DATA C56,C57,C58,C59,C60,C61,C62,C63
 DATA C64,C65,C66,C77,C77,C77,C77,C77
 DATA C77,C15,C16,C17,C18,C19,C20,C21
 DATA C22,C23,C24,C25,C26,C27,C28,C29
 DATA C30,C31,C32,C33,C34,C35,C36,C37
 DATA C38,C39,C40,C77,C77,C77,C77,C77
#endasm
```

Disk 78. Contents of file PFL;S

```
       TITL 'PROGRAM FILE BUILDER/LOADER'
*
*  MENU DRIVEN VERSION
*
* BY CLINT PULLEY  LAST EDIT 85/07/16 0720
*
       IDT  'PFLOAD'
*
       DEF  START,BUILD
       REF  VDPWD,GRMRA,GRMWA,GRMRD
*
MDRV   EQU  1            DISK DRIVE FOR PMENU FILE
BDRV   EQU  1            DISK DRIVE FOR BUILD OUTPUT
*
KSCAN  EQU  >2108
VSBW   EQU  >210C
VMBW   EQU  >2110
VSBR   EQU  >2114
VMBR   EQU  >2118
DSRLNK EQU  >2120
*
       AORG >2900        USE MINIMEM FOR BUILD
*
START  B    @PGMSEL
*
*  PFLOAD - PROGRAM FILE LOADER
*           ENTER WITH STRING ADDR FOR FILENAME
*           IN R3 (STRING PRECEDED BY COUNT BYTE)
*
PFLOAD CLR  R4           START ADDR FLAG
PFL1   LI   R0,>F00      MOVE PAB TO VRAM
       LI   R1,PABI
       LI   R2,10
       BLWP @VMBW
       MOVB *R3,R2       NAME LENGTH
       SRL  R2,8
       MOV  R2,R5        SAVE LENGTH
       INC  R2           FOR COUNT BYTE
       MOV  R3,R1
       LI   R0,>F09
       BLWP @VMBW        COUNT & FN TO PAB
       MOV  R0,@>8356
       BLWP @DSRLNK      LOAD FILE
       DATA 8
       JEQ  DSKERR
       LI   R0,>1000
       LI   R1,HDR
       LI   R2,6
```

```
       BLWP @VMBR         READ HEADER
       MOV  @HDR+4,R1     FWA FOR THIS FILE
       MOV  R4,R4         IS START ADDR SET?
       JNE  PFL2          YES
       MOV  R1,@RUNWS     NO, SAVE FOR START
       SETO R4
PFL2   MOV  @HDR+2,R2     LENGTH
       LI   R0,>1006      START OF IMAGE
       BLWP @VMBR
       ABS  @HDR          MORE FILES?
       JEQ  PFL3          NO, RUN PGM
       A    R3,R5         ADDR OF LAST CHAR IN NAME
       AB   @ONE,*R5      INCREMENT FILENAME
       JMP  PFL1          LOAD NEXT FILE
*
PFL3   BL   @CLS
       LI   R0,>2104      BUILD EXBASIC UTILITY
       LI   R1,>2018      VECTORS
       LI   R2,14
PFL4   MOV  *R0+,*R1+
       DEC  R2
       JNE  PFL4
       LWPI RUNWS
       B    *R0           RUN LOADED PGM
*
CLS    CLR  R0            CLEAR SCREEN
       LI   R1,'  '
       BLWP @VSBW
       LI   R0,767
CLS1   MOVB R1,@VDPWD
       DEC  R0
       JNE  CLS1
       RT
*
DSKERR BL   @CLS          DISK ERROR - REPORT & WAIT
       LI   R0,358
       LI   R1,ERRMSG
       LI   R2,20
       BLWP @VMBW
       LIMI 2
WAIT   JMP  WAIT          WAIT FOR QUIT
*
GRMSAV BSS  2             GROM ADDR SAVE
RUNWS  EQU  >20BA         WS FOR LOADED PGM
PABI   DATA >500,>1000,0,>2400,0
ONE    DATA >0100
HDR    BSS  6             FILE HEADER
ERRMSG TEXT '*DISK ACCESS ERROR!*'
*
SELC   TEXT 'A '          SELECTION CHAR
```

```
ATXT   TEXT 'PROGRAMS AVAILABLE :'
BTXT   TEXT 'YOUR CHOICE? '
NULL   BYTE >FF          NO KEY VALUE
AAA    TEXT 'A'
*
*  SELECT PROGRAM FILE TO LOAD
*
PGMSEL MOV  R11,@RUNWS+22
       LWPI >8300
       LI   R0,>70E8     MOVE UTILITY CODE
       LI   R1,>20FA     FROM ED/ASM GROM
       LI   R2,>2C2      TO LOW RAM SO COLD
       MOVB @GRMRA,@GRMSAV SAVE GROM ADDR
       NOP
       MOVB @GRMRA,@GRMSAV+1
       DEC  @GRMSAV
       MOVB R0,@GRMWA    STARTS WILL WORK
       SWPB R0
       MOVB R0,@GRMWA    THIS ASSUMES ALL
       SWPB R0           ED/ASM CARTRIDGES
PSL0   MOVB @GRMRD,*R1+  ARE THE SAME
       DEC  R2
       JNE  PSL0
       MOVB @GRMSAV,@GRMWA RESTORE GROM ADDR
       NOP
       MOVB @GRMSAV+1,@GRMWA
       BL   @CLS
       LI   R0,6
       LI   R1,ATXT
       LI   R2,20
       BLWP @VMBW
       BL   @DSRACC      OPEN MENU FILE
       JEQ  DSKERR       IF NO FILE
       MOVB @RDOP,@MPAB  SET PAB FOR READ
       LI   R12,73       1ST PGM LINE ON SCREEN
       LI   R13,FNB      FILENAME BUFFER
       CLR  R14          CTR FOR # PGMS
PSL1   BL   @DSRACC      READ RECORD
       JEQ  PSL2         IF EOF
       LI   R0,>1000
       LI   R1,MRBUF
       LI   R2,12
       BLWP @VMBR        GET TEXT
       MOV  R12,R0
       INCT R0
       BLWP @VMBW        TO SCREEN
       LI   R0,>F05
       BLWP @VSBR        GET REC LEN
       SRL  R1,8
       AI   R1,-12       FN LENGTH
       MOV  R1,R2
```

```
        MOV  R13,R1       FNB LOC
        INC  R1           FOR LENGTH BYTE
        LI   R0,>100C     FN FIELD
        BLWP @VMBR        READ FILENAME
        SWPB R2
        MOVB R2,*R13      FN LEN TO BUFFER
        AI   R13,16       NEXT FN
        MOV  R12,R0       SCREEN LOC
        LI   R1,SELC
        LI   R2,2
        BLWP @VMBW        SELECTION CHAR
        A    @ONE,*R1     INC CHAR
        AI   R12,32       NEXT LINE
        INC  R14
        JMP  PSL1         GET NEXT RECORD
PSL2    MOVB @CLSOP,@MPAB
        BL   @DSRACC      CLOSE FILE
        LI   R0,745
        LI   R1,BTXT
        LI   R2,13
        BLWP @VMBW        DISPLAY CHOICE?
        A    R2,R0
        CLR  @>8374
PSL3    LIMI 2            ALLOW QUIT
        LIMI 0
        BLWP @KSCAN
        MOVB @>8375,R1
        CB   R1,@NULL     KEY PRESSED?
        JEQ  PSL3         NO
        CB   R1,@AAA      IS IT ALPHA
        JLT  PSL3         NO
        ANDI R1,>5F00     MAKE UPPER CASE
        BLWP @VSBW        DISPLAY CHAR
        SWPB R1
        AI   R1,-65       MAKE INDEX
        C    R1,R14
        JHE  PSL3         IF OUT OF RANGE
        SLA  R1,1         MAKE WORD INDEX
        MOV  @ADDTAB(R1),R3 GET FILENAME PTR
        B    @PFLOAD
*
* DSRACC - DSR ACCESS SUBROUTINE FOR PMENU
*
DSRACC  LI   R0,>F00      VRAM PAB ADDR
        LI   R1,MPAB
        LI   R2,20
        BLWP @VMBW        STORE PAB
        AI   R0,9
        MOV  R0,@>8356
        BLWP @DSRLNK
```

```
       DATA 8
       RT
*
MPAB   DATA >0014,>1000,>5000,0,10 MENU FILE PAB
       TEXT 'DSK'
       BYTE 48+MDRV
       TEXT '.PMENU'
CLSOP  BYTE 1            I/O OPCODES
RDOP   BYTE 2
*
*  ADDRESS TABLE (20 SLOTS)
*
ADDTAB DATA FNB,FNB+16,FNB+32,FNB+48,FNB+64,FNB+80
       DATA FNB+96,FNB+112,FNB+128,FNB+144,FNB+160,FNB+176
       DATA FNB+192,FNB+208,FNB+224,FNB+240,FNB+256,FNB+272
       DATA FNB+288,FNB+304
*
MRBUF  EQU  $            MENU READ BUFFER
FNB    EQU  $+16         FILENAME BUFFER
*
SLAST  EQU  $
*
*  BUILD PROGRAM FILE (UTIL1)
*
BUILD  LWPI >8300
*      BL   @CLS
       LI   R0,322
       LI   R1,CTXT
       LI   R2,30
       BLWP @>6028       VMBW
       CLR  @>8374
       CLR  R1
BLD1   BLWP @>6020       WAIT FOR SPACEBAR
       MOVB @>8375,R1
       CI   R1,>2000
       JNE  BLD1         IF NOT SPACE
       LI   R1,SLAST     COMPUTE LENGTH
       AI   R1,-START
       MOV  R1,@BHDR+2   PUT IN HEADER
       AI   R1,6
       MOV  R1,@FLEN     PUT IN PAB
       LI   R0,>1000
       LI   R1,BHDR
       LI   R2,6
       BLWP @>6028       STORE HEADER
       A    R2,R0
       LI   R1,START
       MOV  @BHDR+2,R2
       BLWP @>6028       STORE PGM
       LI   R0,>F00
       LI   R1,BPAB
```

```
       LI   R2,20
       BLWP @>6028       STORE PAB
       AI   R0,9
       MOV  R0,@>8356
       BLWP @>6038       DSRLNK
       DATA 8
       LIMI 2
       BLWP @0           RETURN TO SYSTEM
*
BHDR   DATA 0,0,START    UTIL1 PF HDR
CTXT   TEXT '<INSERT DISK AND PRESS SPACE>'
BPAB   DATA >0600,>1000,0
FLEN   DATA 0,10
       TEXT 'DSK'
       BYTE 48+BDRV
       TEXT '.UTIL1'
       END
```

Disk 78. Contents of file PMENU

```
9900 Bthu   DSK1.BTHU1
Disk Direct DSK1.SD
This programDSK1.THIS
That programDSK1.THAT
```

Disk 79. Checkbook Writer

Version: *Author*: Melvin Nomina
Requires: *Language*: *Updated*:

Records checks you've written and can reconcile your account vs your bank statement. Has a calculator window callable from inside the program; also has disk management capabilities.

```
dskdir. v2.0. 12-dec-96
Disk name               = CHECKWRITE
Sectors total           = 360
Sectors used            = 255
Sectors available       = 103
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >007 -READ-ME-     5 DIS/VAR 80   >0dc 004
002 >00a BKBBOOK1      2 DIS/VAR 80   >108 001
003 >00b BKBBOOK2      2 DIS/VAR 80   >109 001
004 >00c BKBBOOK3      2 DIS/VAR 80   >10a 001
005 >00d BKBBOOK4      2 DIS/VAR 80   >10b 001
006 >00e BKNBOOK1      2 DIS/VAR 80   >10c 001
007 >00f BKNBOOK2      2 DIS/VAR 80   >10d 001
008 >010 BKNBOOK3      2 DIS/VAR 80   >10e 001
009 >011 BKNBOOK4      2 DIS/VAR 80   >10f 001
010 >003 CBW4/41DOC   26 DIS/VAR 80   >06a 024 >0e0 001
011 >005 DIM          33 PROGRAM      >0a5 032
012 >006 DIM/HELP     24 DIS/VAR 80   >0c5 023
013 >008 DIM/REVIEW   39 DIS/VAR 80   >0e1 038
014 >002 LOAD         73 INT/VAR254   >022 072
015 >009 PRI-SET       2 DIS/VAR 80   >110 001
016 >004 RECONCILE    37 PROGRAM      >082 035 >107 001
```

Disk 79. Contents of file -READ-ME-

This disk is a gift from the LIMA (OH) AREA USER GROUP. It contains the checkbook and checkwriting program "Checkbook Writer v4.41" as LOAD and the companion program RECONCILE. These are in the public domain and are described in the May 87 issue of our newsletter Bits Bytes & Pixels. The text of this description is included here as CBW4/41DOC.

This disk also contains Disk Information Manager (DIM) and its documentation (DIM/HELP). This program is reviewed in our June 87 newsletter. The review is on this disk as file DIM/REVIEW. DIM has been around awhile and is in the public domain. It has some unique features not found in other software packages. DIM loads using the E/A module option #5. DIM cannot be loaded from Funnelweb or from a Gram Kracker E/A that does not reside in GRAM 1 (>6000).

Disk 79. Contents of file BKBBOOK1

0

Disk 79. Contents of file BKBBOOK2

0

Disk 79. Contents of file BKBBOOK3

0

Disk 79. Contents of file BKBBOOK4

0

Disk 79. Contents of file BKNBOOK1

0

Disk 79. Contents of file BKNBOOK2

0

Disk 79. Contents of file BKNBOOK3

0

Disk 79. Contents of file BKNBOOK4

0

Disk 79. Contents of file CBW4/41DOC

CHECKBOOK WRITER ver 4.41

by MELVIN NOMINA

Lima Ohio User Group

Here is the latest version of my Checkwriter program with many enhancements over previously published versions. In case you haven't used this program before, it may be the only checkbook program that actually SAVES YOU TIME. The program prints checks and stores check data on a disk file for later retrieval and/or editing. All you do is type in the data for each check (date, payee, dollar amount, etc) and the computer then prints a check stub and prompts you to insert a check in your Star-Epson compatible printer. You can use ordinary sized bank checks and slip them behind the print head and over the paper already in the printer. Tuck the bottom of the check under the bar from which the paper emerges, align the top of the check with the printed dashed line, and align the left of the check with the left paper perforation. Lowering the paper bail bar over the check holds the check in position. Pressing **ENTER** then prints the data directly onto the check. This program will also support large 8 × 11 checks with sprocket holes that feed through the printer like fan fold paper. These large checks are available from most banks on special order.

This new version of Checkbook Writer comes in two parts, the Checkbook Writer program itself, and a linked program called RECONCILE which is used to reconcile bank statements. Both programs should be placed on the same disk and named LOAD and RECONCILE respectively and used in DSK1. To change to another drive number see line 25. The two programs will keep track of 4 different accounts, all from the same two programs and all on the same SS/SD disk. A D/V 80 file named CHECKBOOKx is created for each of the four accounts. These files show the following information for every check or deposit written on the account: check number, date, written for, deposit amount, payee, check amount, has check cleared bank, and account balance after the particular check or deposit. The CHECKBOOKx files can be read and if necessary edited with TI-Writer.

Checkwriter V4.41 has the following features not present on the earlier version published in the Feb 87 issue of BITS BYTES and PIXELS:

1. The program looks to see if the necessary data files are present. If not present, these data files are automatically generated on the disk as needed. No user intervention is needed for the automatic creation of these files.

2. A window calculator has been added so +-/* calculations can be done manually from within the program. The program, automatically keeps continuous track of account balances.

3. A "Memo" of up to 28 characters can now be printed directly onto the lower left part of the check, such as "for VISA account 3333333333". The memo size is larger than the "For" data (only up to 8 characters) which is retained in the CHECKBOOKx file for every check transaction.

4. Several minor bugs have been fixed, and the display of the checkwriting menu (the third menu) has been improved.

When your bank statement arrives, load the RECONCILE program. This can be loaded by name, or from the "Reconcile statement" option of the first Checkbook Writer menu. The RECONCILE program searches the CHECKBOOKx file of written checks for any that have not yet been checked off. These are listed on the screen and you type in the record number (same as the TI-Writer line number) of those checks and deposits listed on the bank statement. These are then read back into CHECKBOOKx file flagged as having been cleared by the bank. From RECONCILE you can also generate on a printer a complete listing of CHECKBOOKx and a summary of your outstanding checks and bank balance. The RECONCILE program provides a very easy way to reconcile and balance your checking accounts. After reconciliation, you have the option of loading Checkbook Writer or exiting to Extended Basic.

If you have used an earlier version of Checkbook Writer and your data files seem to be incompatible, the problem may be a null string (blank line) at the beginning of each file. Load your Checkwriter data files (CHECKBOOKx, BKBBOOKx, BKNBOOKx, and PRI-SET) with TI-Writer and delete the first blank line, if it is present. All BKBBOOKx BKNBOOKx and the PRI-SET file used in this version 4.41 have a null string (blank line) as the second record of the file. This empty second record makes Checkbook Writer fully compatible with Horizon Ramdisks. The operating system of these ramdisks has trouble with single record VARIABLE files that are overwritten again and again. If you are using BKBBOOKx BKNBOOKx and PRI-SET files from a previous version of Checkbook Writer, you may have to add this empty second record (line) with TI-Writer in order to make them compatible with this new version. You could, of course, let ver 4.41 automatically create its own data files from scratch.

These two programs are fairly easy to use. Each is menu driven, and each has an instruction subprogram that can be read at any time. I hope you like them. I think they really do save time compared to hand writing checks and using a calculator to figure out balances. These programs are for poor people. The cost is low (free) since I am placing them in the public domain. However, $9999.99 is the largest amount that you can write a check for or deposit at one time. If you need to write larger checks, than you can probably afford to pay for the required software.

JUNE 2, 1987 — Bugs noted in the June 87 Lima User Group newsletter have been fixed in the Checkbook Writer LOAD and RECONCILE programs on this disk.

Disk 79. Contents of file DIM/HELP

Disk Information Manager
Operating Instructions

Once the drive number (1 , 2 or 3) is selected , the five options available are listed below.

CTRL B	Back to drive select.
CTRL I	Initialize disk in drive selected.
CTRL L	Look at sectors in drive selected.
CTRL S	Save the disk name at NAME=.
ENTER	Catalogue the disk in drive selected.

Look Options

CTRL 0	Looks at the first 128 bytes in the sector selected.
CTRL 8	Looks at the last 128 bytes in the sector selected.
CTRL S	Sector selection. Note that sector number is a hex value. **FCTN E** and **FCTN X** may be used to increase or decrease the sector number.
CTRL B	Back to drive select.
CTRL O	Send sector information to a printer or disk file. **CTRL Append** in this option allows you to add the information to the bottom of the file name on the cursor line. e.g. DSK2.SECTOR
CTRL H	Modify HEX values in the sector. Use **CTRL W** option to save the new values.
CTRL A	Modify ASCII values in the sector.
CTRL R	Sets **ENTER** option to sector read.
CTRL W	Sets **ENTER** option to sector write.
ENTER	Reads or writes disk sector.

Catalogue Options

FCTN E	Scrolls file names up.
FCTN X	Scrolls file names down.
CTRL A	Sets or resets all files less than 46 sectors long for copying or recording. A small "c" to the left of the file name indicates which files have been selected.
CTRL B	Back to drive select.
CTRL C	Copy file (max. 45 sectors) to a second drive if the **CTRL A** or **CTRL 0** option has been used previously ; or a selected file if those options haven't been used. The copy disk must be inserted in the drive selected before you enter the copy file name. If the **CTRL A** or **CTRL 0** options were used it will copy from drive 1 to drive 2 or vise-versa.
CTRL D	Deletes file on cursor line. Actual file deletion takes place with **CTRL S** option.
CTRL L	A listing of the disk sectors used by the file is displayed. **FCTN E** or **FCTN X** may be used to change the sector. **ENTER** will read and display sector values. Use 0-8 to change the range displayed. CTRL-Out option lists the file sectors used by ALL files on a printer or disk file. CTRL-Modify allows you to change sector values (refer to Look Options).
CTRL N	Catalogues new disk in drive selected.
CTRL O	Outputs catalogue information to a disk or printer. The Save option in this mode stores the catalogue in a file named *CATALOGUE. This information may be retrieved using the Load option.
CTRL P	If a file is write protected , this option unprotects it or vise-versa.
CTRL R	Records files selected by **CTRL 0** or
CTRL A	options on CS1 and/or CS2.
CTRL S	Saves information from the **CTRL P** or **CTRL D** or **ENTER** options on the disk. A ">" symbol to the left of the file name indicates there is information to be saved on the disk.
CTRL T	Tabs over to the 1.INFO , 2.USE or 3.DATE section to allow modification of that data. Use **CTRL T** to tab back to the file name.
CTRL 0	Selects or de-selects file on cursor for the Cpy or Rec options.
CTRL 1	Selects the 1.INFO data for display. This information may be any 8 character file extension information.
CTRL 2	Selects the 2.USE data for display. One of 6 file types may be selected.
CTRL 3	Selects the 3.DATE data for display. The date must be between JAN01/80 and DEC31/95 and entered in that format to be accepted.
CTRL 4	Resets to the top of the catalogue.
CTRL 5	Moves cursor line half way up the file list.
ENTER	A new file name may be typed in and entered. Once entered , the file sequence is resorted. To save the new file name use the CTRL-Save option.

Disk 79. Contents of file DIM/REVIEW

A REVIEW OF "DISK INFORMATION MANAGER"

by Charles Good

There are several disk managers available to TI users including DM1000 and TI's Disk Manager II module. These managers do housekeeping functions such as disk initialization, file and whole disk copying, changing names of files, etc. There are several good disk sector editors available which can examine and modify the contents of disk sectors. Such programs include DISKO, the DISK FIXER, and DISK+AID. There is, however, only one TI program I know of that works as both a disk manager and a disk sector editor. This program is the public domain DISK INFORMATION MANAGER, or DIM.

DIM loads from option 5 of the E/A module. It will not load from Funnelweb or from a moved E/A file in a Gram Kracker. The operation of DIM requires pressing the CTRL key and one other key to access commands. The FCTN key is only used with the arrow keys to move the cursor. Documentation is rather sparse, but there is always an on screen display of the options immediately available at various points in the program. With this on screen display you can usually figure out what to do next. DIM accesses only drive numbers 1-3. This is a disadvantage for those with more than a total of 3 drives and ramdisks.

Using DIM as a disk manager you can initialize disks in single or double sided and single or double density format. If you don't specify a disk name, the disk is initialized without any disk name. As the disk is automatically verified after initialization you see a graphic shape that is erased from the screen pixel by pixel as each sector is verified. You can rename disks, delete files from a disk, and copy single files or an entire disk file by file. Unfortunately, files larger than 45 sectors cannot be copied. This is a significant limitation. Copying from DSK1. to DSK2. or visa versa is easy. Copying using only one drive is harder since there are no prompts that say when to insert the master and copy disks. To copy with one drive do not use the **CTRL A** or **CTRL 0** commands. Move the cursor next to the file to be copied (**FCTN E** and **X**). Then press **CTRL C**(opy) and **ENTER**. Next indicate the copy disk will be in DSK1 and with the master disk in DSK1. Press **ENTER**. The drive whirrs. Replace the master disk with the copy disk and press **ENTER** again. The drive whirrs again and the file is copied.

One interesting feature of DIM, not found on other disk managers, is the ability to store right on the disk extra information about each file. You can display a disk directory by pressing **ENTER** three times after loading DIM and inserting a disk to be cataloged. You can add three fields of extra information to each file of this catalog. **CTRL 1**,**2** or **3** displays one of these fields next to the file name and **CTRL T**(ab) moves the cursor to the field. After data is typed in the field, you need to press **ENTER**. Field #1 is a file description of up to 8 characters. The second information field shows the type of file. Choices are Xbasc, Basic, Assmbly, Data, Source, and Object. You enter the first letter (X,B,A,D,S,O) to show the file type. The 3rd information field contains a date. After entering this extra file information the catalog may be output to a printer (**CTRL O**, then PIO, then **ENTER**) or a D/V80 disk file [**CTRL O**(utput), then DSK1.xxxxx, then **ENTER**]. This disk file (a good name would be CATALOG) can than be read any time with TI-Writer. You can also output the catalog and information fields to a special disk file called *CATALOG [**CTRL O**(utput) then, **CTRL S**(ave)].

*CATALOG is a 2 or 3 sector program image file that is loaded and displayed the next time you use DIM to catalog the disk. Data in *CATALOG can only be read with DIM. A sample DIM catalog output to a printer is shown below.

A really interesting feature of DIM when used as a disk manager, is DIM's ability to dump a while disk or selected files to tape in one smooth operation. As far as I know, no other software has the ability to dump a whole disk to tape. To dump a whole disk press **CTRL A**(ll) to mark all files less than 46 sectors long for copying. A small "c" appears next to each file. You can then scroll up and down (FCTN E and X) and press **CTRL 0**) to unmark those few files (such as the disk LOAD program) that you don't want to copy. When you are ready to copy press **CTRL R**(ecorder), press cassette record, press **ENTER**, and go have a cup of coffee. There is no need to mess with the cassette or the keyboard any more until all the files are copied one after the other in alphabetical order to tape. The recorder then stops and you are instructed to press CS1 stop and then press **ENTER**. It takes DIM about 15 minutes to dump a SS/SD disk to tape. DIM does not verify the tape, but as long as I use good quality tape I have had no problem with this lack of verification. As a user group librarian I do a lot of disk to tape transfers for our members who don't have disk systems. DIM makes it sooo easy! The successive files are spaced very closely on the tape, with only about one tape count number between files.

You can use DIM to dump E/A #5 program image files to tape. These files can then be loaded back using only a console, the E/A module, and 32K. This works great for those who have built 32K into their consoles, or those without a disk system and with a 32K side car as their only peripheral. To load these E/A #5 files press **2** for E/A then press **5** for RUN PROGRAM FILE. When asked for FILE NAMEtype "CS1.X" and follow the tape loading instructions. If you just type CS1 you can only load one E/A #5 file. If the software you want to load consists of a series of files (such as ADVENTURE1 followed by ADVENTURE2) then you need to use CS1.X for the file name.

DIM will dump any kind of file to tape, including file structures that aren't supposed to be compatible with tape such as D/V 80 or I/F 200. These files dump to tape just fine, but you can't get them back off the tape into the computer. To get a relocatable D/F 80 assembly language program onto tape and then back into the computer you first have to convert the D/F 80 files to PROGRAM image format using the SAVE utility that comes with TI's E/A package, or the FWSAVE utility that is part of Funnelweb.

DIM does everything other sector editor programs do, and sometimes a little bit more. You can select a sector for examination and editing immediately after selecting a drive number. Or from the DIM catalog you can get a list of sectors occupied by each program (**CTRL L**(ook)). You can then display these sectors on the screen and modify them if you choose in either HEX or ASCII. The screen display shows BOTH HEX and ASCII at the same time. You don't have to switch back and forth between the two, but you do have to switch back and forth between the first 128 bytes of a sector and the last 128 bytes of the selected sector (**CTRL 0** and **8**). DIM is the only sector editor program I know that shows both ASCII and HEX at the same time. You can dump the entire sector (no split between the first and second 128 bytes) to a printer in both ASCII and HEX side by side with **CTRL O**(utput). If you have anything in the information fields of the catalog you can optionally add this to the bottom of the sector printout.

In summary, I don't use DIM for my routine disk management because DIM is limited to manipulating only files smaller than 46 sectors and only drives 1-3. I LOVE the ability to dump all or most of a disk to tape. The ability to put E/A #5 assembly programs on tape helps some of our group's tape only members. User Group librarians and those in charge of making tapes for members take note! DIM is my favorite disk editor. It is better than DISKO (even the enhanced disko that comes with Funnelweb) in that it displays both ASCII and HEX side by side and has a built in printer dump.

DISK INFORMATION MANAGER can be obtained from the Boston Computer Society for $3 plus $1 postage and handling. Ask for public domain disk #43 which contains DIM and other goodies. Send your check to:

Boston Computer society
TI User Group
One Center Plaza
Boston MA 02108

DIM can also be obtained from the FREE ACCESS LIBRARY for a small copying fee. Phone Guy Romano at 415-753-5581 Mon-Sat 9-4 Pacific time and ask about obtaining program H171.

Disk 79. Contents of file PRI-SET

PIO

Disk 80. Omega and Archiver II

Version: Archiver v2.4 *Author*: Travis Watford, Barry Boone
Requires: EA *Language*: AL *Updated*: 3/1/88

A telecommunications disk featuring an excellent new terminal program featuring on-line graphics and macros, and a very fast, full-featured archiver utility (version 2.4).

```
dskdir. v2.0. 12-dec-96
Disk name               = OMEGA/ARC2
Sectors total           = 360
Sectors used            = 147
Sectors available       = 211
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00a -README       5 DIS/VAR 80   >0a8 004
002 >009 ARCHIVER     22 PROGRAM      >093 021
003 >002 DF128/DV80    6 PROGRAM    Y >022 005
004 >003 OMEGA/REF    29 DIS/VAR 80   >027 028
005 >004 OMEGA1       33 PROGRAM      >043 032
006 >005 OMEGA2       33 PROGRAM      >063 032
007 >006 OMEGA3       11 PROGRAM      >083 010
008 >007 PRINTDOC      5 PROGRAM      >08d 004
009 >008 READ*ME       3 DIS/VAR 80   >091 002
```

Disk 80. Contents of file -README

Boston Computer Society
TI-99/4A User Group
Public Domain Software Library
OMEGA and ARCHIVER II

Some notes

Omega is a terminal emulator program by Travis Watford, creator of the MAX-RLE program for the 99/4A which allowed us to use graphics pictures from other computers. It has many unique features which are tersely described in the OMEGA/REF file. Travis is interested in what you have to say about the program so that he can work to improve it, so please contact him if you have suggestions regarding OMEGA.

Archiver II is a fairware archiver program by Barry Boone. It is probably the best archiver program available in that it is fast and full featured. It is a fairware program and if you use it at all (and those of you who are into telecommunications probably do), please send a payment to Boone for this fine effort.

Disk 80. Contents of file OMEGA/REF

OMEGA Reference Card

by Al Hattem, S.C. Midlands 99ers

OMEGA is a terminal program written by Travis Watford that includes on-line RLE viewing, programmable keys, one-time set-up and other features. There are 3 types of command keys:

Function keys, Control keys and Function+Shift keys. (F=Fctn, C=Cntl, F+S=Function+Shift)

Control Keys:

Toggle Baud rate	C-1
Cat Dsk, Save Settings	C-2
Uploads/Downloads	C-3
Toggle RS232 port	C-4
Foreground Color	C-5
Background Color	C-6
View Buffer("S" or space bar to scroll)	C-0

Function Keys:

Abort XMODEM Transfer	F-4
User-defined Programmable keys (up to 2 lines of text)	F-1-0
Buffer mode (Conference, Chat, preparing program keys)	F-B
Toggle LOG buffer (will not overwrite previous files)	F-.
Save LOG File-(8K of log buffer in memory)	F-,
Toggle Clock	F-C
Clear Screen	F-L
View "SNAPSHOT" Screens saved (menus, DL lists) (Any key exits snapshot)	F-J or K

Function-Shift Keys:

QUIT-(Log file is automatically saved)	F+S-Q
Toggle RLE Mode-(Saves Log/press "S" to save RLE pic	F+S-R
Status Line	F+S-T
Clear Clock	F+S-C
Clear Log Buffer	F+S-.
Toggle Word Wrap	F+S-W
Snapshot (Save screen in memory)	F+S-J/F+S-K
Save Programmed keys	F+S-1 thru 0

FILE TRANSFERS

OMEGA supports both CRC and Checksum, self adjusting when uploading to match the receiving system. CRC/Checksum is manually selected from the menu when Control 3 is pressed. Menus are displayed when file transfers selected, the user is prompted for file name, receive/send, etc. Information is displayed(file size, # of blocks, # of sectors, #errors and # of retries). Errors may be seen during file transfers over packet-switching networks(PC Pursuit, Delphi, etc) as OMEGA writes to disk(32 sector blocks), but the transfer continues. The file is received intact. If OMEGA cycles thru ten retries, the transfer will be aborted and the file erased. The default file name is "Transfer". If multiple files are downloaded using the default name, OMEGA will assign 2 letter names to each subsequent file (AA, AB, AC, etc.) and will not overwrite previous files.

If you wish to accept the default name, you can download by hitting <C-3, control 3> and then pressing enter to select "Receive, CRC"

Log files are named "LOGAA", "LOGAB", etc, and are saved as separate 8K blocks, not as a continuous file. In this version of OMEGA, logs are saved as DF 128 files and must be converted to D/V80 files by with a program such as DF128/DV80.

To use programmable keys and set up OMEGA:

1. Load the program (E/A option 5-Run Program File)

2. Turn on the statusline(F+S-T) Function-Shift-"T"

3. Set your Baud rate, RS232 port, text and screen colors

4. Enter Buffer Mode(F-B)

5. Type in text to be saved(Your name, user #, Password, dialing commands, etc. (use C-M <Control-M> for carriage returns — This will allow you to set up a key that will log you on most BBS's with one key press)

6. Press Function-Shift and the number of the key you wish to use to recall this info.

7. F-3 will erase the line to enter text for other keys (Keys saved with F+S-# will not be erased, reprogram them to change)

8. When all desired keys are programmed, press F-B to exit Buffer mode

9. Press C-2 to save programmed keys, baud rate, etc. to dsk

10. Whenever you load OMEGA, it will search for "Program 0" and load it from the boot drive(or drive 1 if using Myarc disk controller)

11. Programmed keys may be sent directly by pressing the Fctn key and the proper number key, or you may enter Buffer mode(F-B), view the info and/or change all or part of the programmed text before sending by pressing "Enter".

12. The cursor is also saved at any position for ease in changing numbers, etc. in dialing commands.

13. Normal function keys(Delete, Insert, Erase) are active in buffer mode. See the TI manual for keypresses for control characters(e.g. Cntl-"M"=carriage return)

TO USE RLE MODE:

To view RLE pictures on line, Enter RLE mode(F+S-"R"). The picture must be sent ASCII to view on line. Some BBS sysops may have to add RLE's to the text area if their program does not allow on line reading of text from the download area. Once the transfer is started, the screen should turn white with side borders. The RLE is displayed one line at a time and my be saved by pressing "S" after the RLE is complete. RLE's may be downloaded with CRC or checksum from BBS's not capable of ASCII transfer from download area, although the RLE can not be viewed on-line. The status line will indicate when OMEGA is in RLE mode.

RLE mode uses all video RAM and any snapshots will be lost. The Log file is closed and saved when RLE mode is entered. RLE mode can be aborted by pressing "enter".

STATUS LINE (Function-Shift "T") This line displays the Baud rate, remaining buffer size, whether RLE mode or Word Wrap is activated, the clock and the RS232 port selected.

Duplex toggle, linefeed toggle, print spooling and line-by-line upload of text are not included in this version of OMEGA.

For more information or bug reports, contact Travis Watford, CI$ # 72466,2174 or at the ORPHANAGE BBS 803-754-4996 (S.C.)

Disk 80. Contents of file READ*ME

If you have an EPSOM compatible printer, run "PRINTDOC" from Extended Basic. If your printer is not EPSOM compatible, set your printer into condensed print, 1/9" line spacing and 90 lines per page before printing the OMEGA/REF file with TI-Writer, Funnelweb, or DM 1000.

OMEGA runs from Editor/Assembler option 5, Memory Image Program.

Disk 81. BASIC Builder

Version: *Author*: Paolo Bagnaresi
Requires: *Language*: *Updated*: 07/15/87

This program lets you write Extended Basic programs in TI-Writer and then quickly convert it to a runnable XB program.

```
dskdir. v2.0. 12-dec-96
Disk name               = BASICBLDR
Sectors total           = 360
Sectors used            = 328
Sectors available       = 30
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BASLISTRA     2 PROGRAM      >022 001
002 >003 BUILD-DOC1   20 DIS/VAR 80   >023 019
003 >004 BUILD-DOC2   44 DIS/VAR 80   >036 043
004 >005 BUILD-DOC3   64 DIS/VAR 80   >061 063
005 >006 BUILD-DOC4   17 DIS/VAR 80   >0a0 016
006 >007 BUILD-DOC5   74 DIS/VAR 80   >0b0 073
007 >008 BUILD-DOC6   17 DIS/VAR 80   >0f9 016
008 >009 BUILD-DOCS    2 DIS/VAR 80   >109 001
009 >00a BUILD-LOAD   18 PROGRAM      >10a 017
010 >00b BUILD-O      48 DIS/FIX 80   >11b 047
011 >00c FROGGER/L    22 DIS/VAR 80   >14a 021
```

Disk 81. Contents of file BUILD-DOC1

BASIC BUILDER

SIMPLY A LISTFILE LOADER NOT A COMPILER

DISKETTE CONTENT

The diskette provided is a "flippie". You will notice that it has two index holes (small round holes toward the central big round hole) and two indented holes for the Write Protect (square holes on each lateral border). This diskette has been recorded on both sides. Each side is 360 sectors single density. To read the second side you will have to extract the diskette from the drive slot, flip it, and insert it back again. As you can understand, each side works exactly as a normal diskette would.

The first side of the diskette contains:

```
BA-BUILDER Free   31 Used  329
Filename   Size Type/No.  7 P
---------- ---- ----------- -
BASLISTRA     2 PROGRAM     U   Basic List Transformer
BUILD-DOC1   20 DIS/VAR  80 U   Formatter documentation
BUILD-DOC2   44 DIS/VAR  80 U      "            "
BUILD-DOC3   64 DIS/VAR  80 U      "            "
BUILD-DOC4   16 DIS/VAR  80 U      "            "
BUILD-DOC5   74 DIS/VAR  80 U      "            "
BUILD-DOC6   17 DIS/VAR  80 U      "            "
BUILD-DOCS    2 DIS/VAR  80 U      "      Include File
BUILD-LOAD   18 PROGRAM     U   Ext. Basic Fast Loader
BUILD-O      48 DIS/FIX  80 U   Ext. Basic Object
FROGGER/L    22 DIS/VAR  80 U   Ext. Sample LISTFILE
```

The second side of the diskette contains:

```
BA-BUILDE2 Free   20 Used  340
Filename   Size Type/No.  9 P
---------- ---- ----------- -
BUILD-A      55 DIS/VAR  80 U   Assembly source: DATA file
BUILD-B      56 DIS/VAR  80 U     "        "   : Main Routine
BUILD-C      47 DIS/VAR  80 U     "        "   : Collect Rec.
BUILD-D      39 DIS/VAR  80 U     "        "   : Utilities
BUILD-E      23 DIS/VAR  80 U     "        "   :    "
BUILD-F      66 DIS/VAR  80 U     "        "   :    "
BUILD-O      48 DIS/FIX  80 U     "   uncompressed object
BUILD-S       2 DIS/VAR  80 U     "   Copy Directive
CHECK         2 PROGRAM     U   Ext. B. checking program
```

VERY IMPORTANT: BASIC BUILDER IS A FAIRWARE PROGRAM

This program is delivered under the fairware concept. If you like it, and you use it, then you can send a donation ($ 5.00) to the author:

Paolo Bagnaresi
Via J.F. Kennedy 17
20097 San Donato Milanese
Italy

Phone (02)-514.202

Cash, checks, any foreign currency, all will be welcome.

You are encouraged to make copies and distribute them to your friends. It cannot be over emphasized the importance of sending a donation: only in this way we will keep distribution costs down and encourage people to write decent software for our beloved TI-99/4A. If rewards don't come, more and more people will abandon our little "orphan", to pass to another computer. Please help the TI-99/4A to survive!

If you want, I can ship directly new copies to you. However, this would cost you $15.00 a copy, that will cover the diskette cost, the shipping from Italy to North America, and the handling.

HOW TO PRINT THE INSTRUCTIONS

To print the instructions you will have to use the TI-Writer or a similar program (BA-Writer would work fine too). Load the Formatter, insert the diskette from the first side (BA-BUILDER side, the second side is named BA-BUILD2). Then, at the "INPUT FILE NAME" prompt, you can type:

```
DSK1.BUILD-DOCS
```

Then follow the normal procedure you use with the formatter. The file BUILD-DOCS will chain-print all the documentation files, starting from BUILD-DOC1 up to BUILD-DOC6.

Disk 81. Contents of file BUILD-DOC2

BASIC BUILDER DOCUMENTATION

PURPOSE OF BASIC BUILDER

BASIC BUILDER will transform a DIS/VAR 80 file (source) in an Extended Basic program. The DIS/VAR 80 file (LISTFILE) must have the same basic instruction format of a common basic program. Also, BASIC BUILDER will transform back to a program a LISTFILE that you might obtain by a <LIST "DSK1.LISTFILE"> basic command.

BASIC BUILDER is an assembly tagged object file. It will be loaded by Extended Basic, with Expansion Memory and Disk Drives.

With BASIC BUILDER you can relax and easily type your Extended Basic program with all the powerful editing capability of TI-Writer. Moving block of lines around, copy them, modifying them, all this will be a simple task now. Because of the special BASIC BUILDER 0 REM line option (more about it later), at last you will be able to full document your Ext. B. programs without fear of losing Ext. B. program memory. These special 0 REM lines will be left out of the Ext. B. program that will eventually result out of your LISTFILE. Thus, the Ext. B. program will be shorter. At the same time, they will always be present in your LISTFILE, to ensure full documentation to your programming efforts.

Also, BASIC BUILDER will enable you to give a better graphic look and a greater readability to your Ext. B. program source file, since you will be able to "structure" a LISTFILE in an indented way:

```
100   FOR T = 1 TO 200
        :: PRINT STR$(T)
          :: NEXT T

 :: IF B=T THEN
       PRINT "EQUAL"
         ELSE PRINT "DIFFERENT"
```

BASIC BUILDER is not a "COMPILER". You should not think that an Ext. B. program built with BASIC BUILDER will run faster than the same ordinary Ext. Basic program, since they both will run exactly at the same speed.

COMPATIBILITY WARNING

I developed BASIC BUILDER with a TI-Extended Basic Cartridge, ATA 1183 (GROM #: CD 2114 NL and CD 2114 CNL).

Since BASIC BUILDER heavily relies on the absolute GROM Addresses of the Extended Basic Module being used, it is entirely possible that BASIC BUILDER will not work with different Extended Basic module. Particularly, BASIC BUILDER has NOT been tested on MicroPAL Extended Basic, nor on Mechatronic Extended Basic. It surely will not work with Myarc Extended Basic.

On the other hand, it will work with Miller Graphics "GRAM CRACKER", when TI-Extended Basic is loaded into GRAMs.

BASIC BUILDER has been developed and tested on this configuration: Black and Silver console, , CorComp DS/DD Disk Controller, Myarc 512 K Expansion Memory, EPSON MX 80 Printer (TI-IMPACT PRINTER). I know for sure that BASIC BUILDER also works with the 1983 V 2.2 consoles.

HOW TO USE BASIC BUILDER

You can load BASIC BUILDER from Extended Basic, with the command:

```
CALL INIT :: CALL LOAD("DSK1.BUILD-O")
```

You can use any drive # you want. Also, you can use the quick Ext. Basic program loader: see at the paragraph A FAST LOADER: BUILD-LOAD.

The file object is AORGed at >2500. Therefore it will erase whatever assembly object is already contained in low memory in the same address range (up to >3317).

Once loaded, only a new <CALL INIT> can erase BASIC BUILDER. A <NEW> command will have no effect on BASIC BUILDER.

To execute BASIC BUILDER you will have to execute a:

```
CALL LINK("BUILD","DSK1.LISTFILE","PIO")
```

"DSK1.LISTFILE" will be the name of your DIS/VAR 80 LISTFILE. You can change "DSK1.LISTFILE" to any file name or device name you want.

Also, you can change "PIO" to any printer device name you will be using.

The printer device will be used by BASIC BUILDER only to give you the chance to print the list of basic line numbers that might have given you errors during the building process. You will see this list on the screen anyway. Therefore, a printer is not necessary for BASIC BUILDER to work.

Also, you can use a "DSK1.ERRLIST" filename instead of the printer device name: then, the error file will be stored in a DIS/VAR 80 file that you can see later with the TI Editor/Assembler or TI-Writer.

DEFAULTS

The following rules for parameter passing apply to the CALL LINK:

1) If you do not supply a LIST FILE NAME the "DSK1.LISTFILE" will be used by BASIC BUILDER the first time you <CALL LINK("BUILD")>. If it is not the first time, then the last file name supplied will be used again by BASIC BUILDER.

2) If you do not supply a PRINTER DEVICE, then "RS232.BA=4800.DA=8" will be used by BASIC BUILDER the first time you <CALL LINK("BUILD",....)>. If it is not the first time, then the last printer device name supplied will be used again by BASIC BUILDER.

3) It is possible to change just the printer device name or only the list file name, by passing only one parameter with the CALL LINK("BUILD"....). For instance:

```
CALL LINK("BUILD","SP.BA=4800.DA=8")
```

will enable BASIC BUILDER to use the Myarc Printer Spooler, and the same LISTFILE that you used previously. For this purpose, BASIC BUILDER will check the first two bytes of the Printer name you have provided.

If the first and only CALL LINK parameter passed begins with any of the following initials: "PI", "RS", "PI", then that device name (full name must be provided), will be considered as a "Printer Device Name" and the "Drive Device name" used at last CALL LINK will be used again.

4) If two parameters are passed with the CALL LINK("BUILD",...), then first parameter will be a "Drive Device name" and the second a "Printer Device name", no matter what their initials are.

Users that want to change LISTFILE and PRINTER defaults can vary the source files of BASIC BUILDER and then reassemble them to create a new object file in a NOT COMPRESSED format ("C" option during assembling is not to be used).

CALLING BASIC BUILDER FROM A PROGRAM BASIC LINE

You can put the command <CALL LINK("BUILD",...)> in a program basic line. However, BASIC BUILDER will start execution only when the basic program comes to the END <* READY>.

WHEN BASIC BUILDER STARTS

When you <CALL LINK("BUILD",....)> you will see the welcome screen (not really impressive):

```
   BASIC BUILDER V 1.0
      Developed by
     Paolo Bagnaresi
     Via Kennedy 17
20097 San Donato Milanese
Italy - Phone (02)-514.202

If you use me, please send
$ 5 to my author!

Call me with:
CALL LINK("BUILD","DSK1.LIST
FILE","RS232/PIO")

Press any key to go on
Press FCTN 4 to stop
```

Then, by pressing **FCTN 4** the you will simply return back to the basic prompt.

By pressing any other key you will notice a screen color change to light yellow. This means that BASIC BUILDER has been put into action.

If the "DSK1.LISTFILE" is found, then you will see the first basic line scrolling on the screen. If the "DSK1.LISTFILE" is not found, then a "File Error", "Press Enter" message will be shown, and you will have to <CALL LINK("BUILD",...)> again after supplying the proper filename.

After the first basic line is assembled into memory, then the next basic lines will be shown to you on the monitor, with a continuous scrolling of the entire screen.

WHEN BASIC BUILDER ENDS

The process comes to an end when the last basic line is found in the file. The basic lines can be in any order in the LISTFILE and they will be assembled into program basic lines in the same time order as they appear on the LISTFILE. However, after BASIC BUILDER has finished its duty, if you LIST the new program to the screen, you will see the normal increasing numeric order in your program lines.

After BASIC BUILDER has finished, and no errors are shown, your new basic program is ready. You can <RUN> it, <SAVE> it, <SAVE><MERGE> it, even <LIST> it back again, with no exception. It is an ordinary basic program by all means, like those obtained by the usual long typing of a line after another.

TIME AND SPEED CONSIDERATIONS

The time needed to process an entire LISTFILE varies, and it will depend on the following factors:

1) Usually, large LISTFILEs will take more time than shorter LISTFILEs.

2) However, a LISTFILE formed by very short basic lines will take much longer than a LISTFILE of the same size formed by multiple statement basic lines .

   When the LISTFILE contains a lot of basic lines, each new line will have to move down in memory all the LINE NUMBER TABLE of the basic program that is being formed, and this takes a lot of time.

3) As a rule of thumb, a multi-statement LISTFILE will take more or less the same time to be built back in a program that has been taken to LIST it to a diskette (<LIST "DSK1.LISTFILE">).

For instance, a multiple statement extended basic program of 112 basic lines and 40 diskette sectors took 118 seconds to be LISTed to a diskette. Then BASIC BUILDER took it back to memory in only 100 seconds (in this case it was faster than the Ext. Basic LIST command!)

Disk 81. Contents of file BUILD-DOC3

LOGIC OF BASIC BUILDER IN PROCESSING THE BASIC LINES

It is well known that when you LIST a basic program to a diskette, a DIS/VAR 80 file is created. In that LISTFILE each line (record) can be as long as 80 characters. If the basic line is longer, then it will be broken into two or more records. BASIC BUILDER has the capability of linking back together these records, to form again the original basic line that was listed to the diskette.

To achieve this goal, BASIC BUILDER uses a logic. It is worthwhile understanding this logic to avoid problems when you will be typing your basic program with an Editor (like TI Editor/Assembler Editor or TI-Writer Editor). With an Editor, each line you see on the screen will form a record on the file when you save it to the drive.

Logic rules:

1) If a record has "UNMATCHED QUOTES", then the next record will be assigned to the this record to form the same basic line. Think of a basic line that contains a <PRINT>, where this <PRINT> actually states the content of a next basic line, as :

```
100 PRINT "THE NEXT BASIC LINE CAN BE CHANGED TO
110 A=2 :: GOTO 2200, IF OPTION 1 IS NOT NEEDED"
110 PRINT "OPTION 1 " : "INPUT USER NAME" :: INPUT A$
```

   If we do not take into account the unmatched quotes in line 100 (first record), we will have almost no means to understand that the second record is NOT a basic line!

   Thus, QUOTES must match, and this is very important to BASIC BUILDER.

2) A valid basic line will start with a positive integer number N, LEFT JUSTIFIED, such that: $0 < N < 32768$. After the basic line number there must be at least one blank. Please note that the N number must begin at the very left of the screen. With TI-Writer, you should always use the TAB SETTING with a Left Margin 0 (zero).

   Also, note that here there is a difference between BASIC BUILDER and GPL Input of Ext. B. Module. In fact, GPL Input of Ext. B. module will accept a line number even if it is not followed by a blank, while BASIC BUILDER won't.

   So, please do not write me to tell me that this is a flaw of BASIC BUILDER. It is not, it's the only way BASIC BUILDER can tell a line number record from a second record of the same basic line. After all, GPL Basic Input KNOWS that the first decimal chars will be the first chars of a new basic line while BASIC BUILDER doesn't, for the above reasons.

After the blank(s) there must be something, that will not be checked by BASIC BUILDER. If a record does not start with a number that is integer, positive, and less than 32768, then this record will not be considered a new basic line. Instead, it will be assigned to the previous basic line.

3) If a record, that starts with a number integer, positive, less than 32768, followed by the one or more blanks, has a "Basic Logic Operator", then this record will be assigned to the previous basic line. The following are considered "Basic Logic Operators" (BLO):
+ - * / = > < <> . , ; : :: " ( )
OR AND XOR TO STEP THEN ELSE

INSTRUCTION FORMAT

The LISTFILE, as said earlier, can be built with an Editor. An Editor like TI-Writer or Editor/Assembler, can do the job. However, these Editors will not save to disk the blanks that are at the end of each line on the screen.

Because of this, you are somehow limited when you are creating your basic program with these editors.

On the other hand, BASIC BUILDER, before processing a record, will remove from the end of it all the <cr> (Carriage Return), <pa> (New Page), <lf> (Line Feed) symbols that can be inserted with the TI-Writer. As a matter of fact, any character with ASCII below 32, located at the very end of the record, will be eliminated.

Some suggestions on how to create a basic program with an Editor can be useful:

1) If a blank has to stay in the basic line, never let it at the end of the line. Rather, put it in the beginning of the next line.

Example:

```
100 PRINT "YOU AND
JOHN"
```

will result in:

```
100 PRINT "YOU ANDJHON"
```

That surely is not what you expected. If you want to use two screen lines to build that basic line you should write:

```
100 PRINT "YOU AND
 JOHN"
```

leaving a blank at the beginning of the second record, which will give you a basic line:

```
100 PRINT YOU AND JOHN"
```

Even worse:

```
100 PRINT A$ :
:B$
```

will result in:

```
100 PRINT A$ :: B$
```

That will give you a basic error message .

Instead, you should write:

```
100 PRINT A$
 : : B$
```

If you use TI-Writer, you can embed a blank at the end of a line by adding a <cr> (Carriage Return) after the last blank you want to have in the record.

2) Since the Editor does away with all the blanks at the end of a record, you cannot simply edit the files that you will obtain with the basic command <LIST "DSK1.LISTFILE">.

Some important blanks can be missed when you save it back to disk, and the resulting program may contain some errors. However, there is a way around it.

You can use the following basic program BASLISTRA on the LISTFILE, before you load it with the Editor. BASLISTRA (BASic LISt TRAnsformer) will change the last blank on each record into an almost unused character, the <|>, CHR$(124), FCTN A, that we will call now the "Blank Alternate".

As a matter of fact, this character, the "Blank Alternate", can be used instead of a blank with the Editor, only if it is the last character on the line. When BASIC BUILDER is called, it will automatically change all the Blank Alternates <|> at the end of each record into a BLANK, thus restoring all the original blanks.

For instance:

```
510 FOR|
T=1 TO 50 :: NEXT T
```

will be assembled as :

```
510 FOR T=1 TO 50 :: NEXT T
```

with a blank in lieu of the <|>.

BASLISTRA: ADAPT A LISTFILE TO AN EDITOR

This is the BASLISTRA program, assuming the LISTFILE you have just obtained with the <LIST> command is named "LISTFILE-A". A new LISTFILE will be formed by BASLISTRA, that will be named "LISTFILE-B".

```
100 REM BASLISTRA :BASIC LIST TRANSFORMER

110 OPEN #1:"DSK3.LISTFILE-A",INPUT

120 OPEN #2:"DSK3.LISTFILE-B",OUTPUT

130 IF EOF(1)=0 THEN LINPUT #1:A$ ::
 IF LEN(A$)=80 AND SEG$(A$,80,1)=CHR$(32)
  THEN A$=SEG$(A$,1,79) ::
   PRINT #2:A$; :: PRINT #2:"|" :: GOTO 130
 ELSE PRINT #2:A$ :: GOTO 130

140 CLOSE #1 :: CLOSE #2 :: END
```

3) Quotes, as said earlier, are very important to BASIC BUILDER. If you miss the second quote (the closing quote), then BASIC BUILDER will look to all next records until it find another quote. Therefore, all the basic lines that follow the line with the unmatched quotes will be missing from the resulting program. Here there is no possible suggestion: you simply have to use quotes in the right manner.

4) Generally speaking, blanks can be inserted at any point in the each line in the same way you would with the basic input. Therefore, you could write:

```
100 INPUT    A$    ::
  B$=SEG$(A$  ,1, 2) ::
    IF  B$=  "XX" THEN 200
       ELSE   IF  B$ = C$ THEN    300
         ELSE 500
```

HALTING BASIC BUILDER : FCTN 4

You can halt BASIC BUILDER at any time during execution by pressing **FCTN 4 (CLEAR)**. Once BASIC BUILDER has been halted, the execution cannot be resumed: it is necessary to start again from the beginning:

```
(CALL LINK("BUILD",..)
```

ERRORS WITH BASIC BUILDER

Three kinds of error, aside from device name errors, may occur with BASIC BUILDER, according to my knowledge.

1) UNMATCHED QUOTES (The second quote is missing).

2) LINE TOO LONG.

3) UNRECOGNIZED CHARACTER (When you use an illegal character for a variable, such as 1$="PAUL").

When an error is encountered by BASIC BUILDER, the basic line # that gave the error and the error type is saved into a STACK. At the same time, the color screen will return to its original cyan color. So, at any time, you will have an easy way to check at once it some error has already occurred.

At the end of the building process, all the errors, if any, will be shown on the screen.

By pressing **FCTN 8 (REDO)**, you will be able to see them all over again.

By pressing **FCTN 4 (CLEAR)**, you will pass to the next option: provided you have a printer, you can hard print the same error list that has been shown to you on the screen. Or, you can save the same error list on diskette, as DIS/VAR 80 file that can be seen later on with an Editor, provided you have given a suitable device name as a Printer name, such as "DSK1.ERRLIST".

Basic lines that issued an error will be missing from the resulting basic program.

When you find any of the first two above errors (UNMATCHED QUOTES and LINE TOO DONG), chances are that the basic lines, that follow IN THE LISTFILE the line that issued the error, will also be missing from the resulting basic program. This means that, if line 220 follows line 500 in the LISTFILE, and if line 500 issued error, then line 500 will be missing in the basic resulting basic program, and probably line 220 will be missing too.

Therefore, in case of error, please carefully check the basic line on the source LISTFILE.

CHECKING IF BASIC BUILDER WORKS PROPERLY

If you want to modify BASIC BUILDER, then you may want to check if the new version you obtain works well. I have a suggestion for it.

A good means to check if the new object just assembled works flawlessly is, obviously, to try it on some known basic program and the <RUN> it. However, this approach will not grant that the program works all-right: some parts of the basic program might have not been used by you during the check, and therefore the flaw will remain hidden to your eyes. There is a way around it. A different approach that will grant a full and complete check.

Suppose you want to check BASIC BUILDER on a known basic program. You first save the basic program in a "MERGE" format, with the name "M1" (SAVE DSK1.M1,MERGE). Then you list to the diskette. Then, after a NEW, you will use your new version of BASIC BUILDER on the listed program. What is taken into memory will be saved again as a "MERGE", with the different filename "M2" (SAVE DSK1.M2,MERGE).

Now you should compare the two MERGE programs with the following basic program:

```
100 OPEN #1:"DSK1.M1",INPUT ,VARIABLE 163
110 OPEN #2:"DSK1.M2",INPUT ,VARIABLE 163
120 IF EOF(1)THEN 130 ELSE A=A+1 ::
    LINPUT #1:A$ :: LINPUT #2:B$ ::
    DISPLAY AT(2 3,1):A ::
    IF A$<>B$ THEN PRINT A$:B$ ::
    B=B+1 :: INPUT C$ :: GOTO 120 ELSE 120
130 CLOSE #1 :: CLOSE #2 ::
    IF B=0 THEN PRINT "THE TWO PROGRAMS ARE" :
    "THE SAME 100 %"
140 END
```

If a difference between the two MERGE programs is found, the two records will be shown and the INPUT C$ will be executed.

A caveat is in order here. In a Basic REM line, the blanks at the end of the line will be kept untouched by the basic interpreter when the input of the line is performed. However, the same doesn't hold true any longer when it comes to Extended basic.

Under the same conditions, the Extended Basic interpreter will do away with all the blanks that are at the end of an Extended Basic line. Therefore, the check suggested above will fail if performed on a basic program (not Extended) that contains blanks at the end of a REM line. You will notice this particular case since the differences will be shown ONLY on records that are REMarks.

A FAST LOADER: BUILD-LOAD

If you don't mind erasing the basic program that you might have in memory, then there is a real fast loader for BASIC BUILDER. It is an Ext. Basic program, that contains inside the object code of BUILD-O. Its name is BUILD-LOAD. Once you have run it, <RUN "DSK1.BUILD-LOAD">, the following actions will take place:

1) Any CALL LINK name with the entry point below >3318 will be erased. This is a safety measure, to avoid possible lock up. The memory area below >3318 is now used by BASIC BUILDER. >3317 is the upper limit of BASIC BUILDER in low memory.

   This action will not totally avoid all possible system crashes. You still can get into one, if your previous Assembly program had some part of itself stored below >3318 and the entry point above >3318. To be sure, please CALL INIT before using BUILD-LOAD.

2) The name "BUILD" will be inserted in the CALL LINK name table.

3) The First Free Address in low memory will be updated only if it was below >3318.

4) The BUILD-O object, stored inside BUILD-LOAD at >F100, will be moved down to >2500 (there are >E18 bytes to move).

5) A "NEW" will be emulated. Once BUILD-LOAD comes to an END, you will not find it any more in memory <"NO PROGRAM PRESENT">.

6) The basic cursor will appear again, and you are now ready to <CALL LINK("BUILD",...)>

Warning: you can edit BUILD-LOAD, add new basic lines, and delete them. However, you cannot RESequence it. If you SAVE-MERGE it, the DIS/VAR 163 file will contain only the basic part of BUILD-LOAD, while the M.L. part will be lost. On the other hand, you can MERGE some external DIS/VAR 163 files to it (thus adding some new basic lines to it).

A SAMPLE PROGRAM: FROGGER

A program I liked a lot, 3 years ago, was FROGGER, an Extended B. program by P. Pheby.

This program has been inserted in the BASIC BUILDER diskette, as a sample LISTFILE, under the name FROGGER/L. This LISTILE has already been modified with the BASLISTRA program.

If you load it with TI-Writer or E/A editor, you may notice a <|> symbol (Blank Alternate), at end of line 9 (basic line 125).

This symbol will ensure a blank between "USE" and "JOYSTICKS" at building time. If it wasn't for the <|> symbol, the original blank would be lost if we saved the FROGGER/L file to a diskette.

You can use this file to see how BASIC BUILDER works. Let's do it. Select Ext. Basic, insert the BA-BUILDER diskette into Drive 1.

Now type:

CALL INIT :: CALL LOAD("DSK1.BUILD-O") and press ENTER. This will load BASIC BUILDER into low memory.

Type now:
CALL LINK("BUILD","DSK1.FROGGER/L") and press ENTER.

This will start BASIC BUILDER. You will see the Drive 1 light go on and the Extended Basic FROGGER will be formed under your eyes.

Disk 81. Contents of file BUILD-DOC4

COMMENT LINES AND BASIC BUILDER

BASIC BUILDER allows the use of comments in the LISTFILE as comment lines. Comment records in the LISTFILE will be skipped by BASIC BUILDER and they will not be shown on the screen during the assembling process. A comment line will begin with a 0 (zero), left justified.

This file is meant to enable the BASIC BUILDER user to understand how to use comments when you will be writing your LISTFILE.

The above lines will not be shown since none of them begins with a valid basic line. BASIC BUILDER starts assembling records from the LISTFILE into an Ext. Basic program only when the first valid Basic Line Number is found.

Therefore, the following lines will be taken to memory.

```
1 ! SAMPLE PROGRAM
100 INPUT A
110 IF A=10
0 THEN 300
ELSE 400
```

120 REM this line will be assembled in the basic program

0 REM This line won't. It begins with zero, therefore it is a comment line.

The next lines will be comment also, since they are after a zero line. They all will be skipped by BASIC BUILDER, like the zero line.

A valid comment line will meet the following requirements:

- It will begin by zero (a 0 line number), left justified like all the valid basic lines.
- The usual rules applied for valid basic line are complied with. This means that:

  1) The line number will be followed by one or more blanks.
  2) After the blanks there must be some other character
  3) This character cannot be a basic logic operator:
     + - * / ^ = > < <> . , ; : :: " ( )
     OR AND XOR TO STEP THEN ELSE

The assembling of a new basic line will be resumed when a valid basic line will be found. A valid basic line number is, as usual, an integer number N such that: 0 < N < 32768. Also, the rules at steps 1, 2, 3, (see above) must be complied with.

For instance, the following line will be a valid basic line:

300 PRINT "A=100" :: STOP

Comment: if you are using TI-Writer to write your LISTFILE, the simplest way to insert adjacent comment lines and forget about rules to select a Left Margin different from zero for the comment lines that follow line zero, like this one. In this way you don't have to worry about what you are typing. There can't possibly be any positive number left justified, since the most left character will always be a blank. Let's complete somehow the sample basic program:

```
400 PRINT "A<>100" :: FOR T=1 TO 100 :: NEXT T
410 PRINT "HAS BEEN INPUT" :"BY THE USER"
```

Another comment line. You should now try to use this file, as is, as a LISTFILE with BASIC BUILDER. You will see that only the real basic lines will be assembled to form an Extended Basic program (lines 1, 100, 110, 120, 300, 400, 410)

The name of this file is "BUILD-DOC4".

Let' do it.

Select Ext. Basic, insert the BA-BUILDER diskette into Drive 1.

Now type: CALL INIT :: CALL LOAD("DSK1.BUILD-O") and Press ENTER. This will load BASIC BUILDER into low memory.

Type now: CALL LINK("BUILD","DSK1.BUILD-DOC4"). Press ENTER.

This will start BASIC BUILDER. You will see the Drive 1 light go on and the basic program (lines 1, 100, 110, 120, 300, 400, 410) will be formed under your eyes.

ACKNOWLEDGMENTS

Credits: BASIC BUILDER has been developed after the idea of John Ford. His XLATE program, that does the job of BASIC BUILDER, was published on *MICROpendium* in 1985 (page 50 of the December 1985 issue, Vol. 2, number 11). XLATE is an Extended Basic program, while BASIC BUILDER is an assembly program. Thank you, John!

Disk 81. Contents of file BUILD-DOC5

FOR THE ASSEMBLER PROGRAMMER: PROGRAM PHILOSOPHY

BASIC BUILDER relies on the following philosophy. Everybody knows that when you are in Command Mode with Extended Basic, you can enter a new basic line by just typing a line number followed by a basic statement.

Now, I thought, if only we could take that basic line from a file (as a record) and put it on the screen, we would have accomplished our task. However, a file cannot usually be read in Extended Basic while in Command Mode.

The trick to solve this problem came to my mind: why not use the Interrupt User Routine? The Interrupt User Routine is usually tested every 60th of a second.

Its address is >83C4 CPU ram. If this location is different from zero, then the value contained will be the address where execution in Machine Language will begin. Once the User wants to release control, then it will simply return by a B *R11.

However, by using this interrupt routine in a careless way, we would end with entering the interrupt routine each time a new GPL instruction is executed. This would happen since the GPL interpreter goes to >0070 CPU RAM each time it has completed a GPL instruction and it needs another. There, at >0070 CPU, a LIMI 2 assembly instruction will be executed, thus enabling our interrupt. An execution of our routine at any interrupt would be too much, since the GPL could be executing something different from what we need (more on this later).

What we need in reality, is to load the records from our file only when the Extended Basic Input routine is waiting for you to type something on the screen. In other words, we do not want to interfere with the GPL interpreter in all moments. We want to fool the GPL interpreter only when it is idle, waiting for the user to give it something to do, and this happens when the GROM address is between >6AAF and >6AD7, in my ATA 1183 Extended Basic Module.

Then, our routine first will check the GROM address being used. If it is not right after the SCAN of the Ext. Basic Input routine, then our routine will restore the GROM address and release control.

Otherwise, it will keep control, and our task can begin. Our routine will load one or more records from our file until a full basic line is formed in CPU RAM. It will then write it to the screen and inform the GPL interpreter that a new line is ready.

This will be accomplished by:

1) Writing to >8320 CPU RAM the address where the basic line starts on the screen.

2) Writing to >832A CPU RAM the end address of our basic line in VDP RAM.

3) It will then fool the Ext. Basic Input routine making it believe that ENTER has been pressed: It will write >0D (ENTER) at >8375 (key pressed in the SCAN routine).

4) Saying to the Ext. Basic GPL software that the prompt, CHR$(30), in already in CPU RAM, by storing >7E to >8301.

5) Now, we will restore the GPL workspace, >83E0 - >83FF, (used by the DSRLNK needed to load records from the file), and the Interrupt workspace, >83C0 - >83DF, (used by the printer), plus a low portion of the Scratch Pad, >8340 - >8370, used by the disk controller software.

6) We will change the old GROM address, to the GROM address where a real **ENTER** key would lead, that is >6BC2, in my ATA 1183 Extended Basic Module.

We are ready now to release control back to the GPL interpreter. We will do so by loading back the workspace that was used when we received control from the user interrupt (it is >83E0), and Branch *R11, thus returning to the interrupt routine.

Now the GPL software will use whatever we have just supplied as something that was typed by the user. Neat, isn't it?

The GPL software will process the basic line: it will tokenize it, store it in memory, udpate the BASIC LINE NUMBER TABLE, moving it down towards >A000. During all this time the GPL Ext. Basic Software is busy, and this is the reason why we have to limit ourselves to supplying new basic lines only when the GPL software is idle, waiting during the Input routine. If we did not wait for this moment, we would end up with supplying basic lines that could not possibly be processed by the Ext. Basic GPL software, because it would not be ready to accept it.

The following is a GPL disassambled op-code, of a part of the GPL INPUT ROUTINE, on my ATA 1183 Extended Basic module. This GPL disassembler has been obtained with the :

TESIO GPL DISASSEMBLER
by TESIOWARE
Riccardo Tesio
Via Sacco e Vanzetti, 5
12100 CUNEO - ITALY
Phone 0171 -60996, calling from Italy.

Thank you, Riccardo!

Anyone interested to TESIO GPL DISASSEMBLER, can contact Riccardo Tesio, at his own address. His program doesn't need a disk drive system. Only Expansion Memory and one of the following Modules: Extended Basic, Mini-Memory, Editor/Assembler.

```
*************************************************************************
  EXTENDED BASIC ATA 1183 FIRST GROM PARTIALLY DISASSEMBLED
*************************************************************************
  GROM  BYTE               MNEMONICS
  ADD   OP
  RESS  CODE
  ====  == == == == == ==  =========
  6A6F  00                 RT
  6A70  4F EE              JNE >6FEE
  6A72  4A 72              JNE >6A72
  6A74  4E 7C              JNE >6E7C
  6A76  4A 8E              JNE >6A8E -> Gets here through a "BL"
  *                                     called from >6424
  *
  6A78  51 5A              JNE >715A
  6A7A  4A 7A              JNE >6A7A
  6A7C  4F BA              JNE >6FBA
  6A7E  52 FD              JNE >72FD
  6A80  53 43              JNE >7343
  6A82  4C FD              JNE >6CFD
  6A84  4D 4E              JNE >6D4E
  6A86  4A 98              JNE >6A98
  6A88  4A A1              JNE >6AA1
  6A8A  4A 95              JNE >6A95
  6A8C  4A 8C              JNE >6A8C
  6A8E  BF 5E 03 7D        MOV >037D,>835E   -> GPL INPUT ENTRY POINT
  6A92  BD 2A 20           MOV >8320,>832A
  6A95  BE 63 FF           MOVB >FF,>8363
  6A98  86 A3 74           CLRB V>0374
  6A9B  BE 60 01           MOVB >01,>8360
  6A9E  BD 61 20           MOV >8320,>8361
  6AA1  86 00              CLRB >8300
  6AA3  86 64              CLRB >8364
  6AA5  BE 01 7E           MOVB >7E,>8301
  6AA8  C0 B0 61 01        SWAPB >8301,V>8361 -> Cursor blinking
  6AAC  86 79              CLRB >8379
 6AAE  03                 SCAN                -> SCAN Keyboard (>000E)
  6AAF  02 63              RND >63             -> Randomize up to 99
  6AB1  6A CC              JEQ >6ACC           -> Any key pressed?
  *                                               Yes, jump
  6AB3  90 00              INCB >8300
  6AB5  D6 75 FF           CB >FF,>8375
  6AB8  6A C5              JEQ >6AC5
  6ABA  CA 00 FE           SLB >FE,>8300
  6ABD  4A C5              JNE >6AC5
  6ABF  A6 00 1E           SB >1E,>8300
```

```
6AC2  05 6A CE           B >6ACE
6AC5  C6 79 0E           SLEB >0E,>8379
6AC8  4A AE              JNE >6AAE
6ACA  4A A8              JNE >6AA8
6ACC  86 00              CLRB >8300
6ACE  D6 01 7E           CB >7E,>8301       -> Cursor in CPU RAM?
6AD1  6A D7              JEQ >6AD7           -> Yes, jump
6AD3  C0 B0 61 01        SWAPB >8301,V>8361 -> Cursor blinking
6AD7  CA 75 20           SLB >20,>8375      -> FUNCTION key?
6ADA  6B D0              JEQ >6BD0           -> No, jump
6ADC  D6 75 02           CB >02,>8375       -> FCTN 4 Key? (CLEAR)
6ADF  4A F4              JNE >6AF4           -> No, jump
6AE1  B2 45 FE           ANDB >FE,>8345
6AE4  8E 44              TESTB >8344
6AE6  6D 4B              JEQ >6D4B
6AE8  DA 45 40           SNCB >40,>8345
6AEB  4A F4              JNE >6AF4
6AED  BD A3 82 1E        MOV >831E,V>0382
6AF1  05 A0 0E           B >A00E
6AF4  D6 75 06           CB >06,>8375       -> FCTN 8 (REDO) key?
6AF7  4B 2A              JNE >6B2A           -> No, jump
6AF9  8E 44              TESTB >8344
6AFB  4A A8              JNE >6AA8
6AFD  B2 45 FE           ANDB >FE,>8345
6B00  BF 20 03 02        MOV >0302,>8320
6B04  0F 83              XMLLNK >83
6B06  A7 20 00 20        S >0020,>8320
6B0A  D5 20 A3 8C        C V>038C,>8320
6B0E  4B 04              JNE >6B04
6B10  BD 2A A3 8E        MOV V>038E,>832A
6B14  BD 4A 2A           MOV >832A,>834A
6B17  A5 4A 20           S >8320,>834A
6B1A  6B 22              JEQ >6B22
6B1C  34 4A B0 20 A8 C0  COPY V>08C0,V>8320,>834A
6B22  BE 63 FF           MOVB >FF,>8363
6B25  BD 61 20           MOV >8320,>8361
6B28  4A A1              JNE >6AA1
6B2A  D6 75 08           CB >08,>8375       -> FCTN S key?
6B2D  6C E7              JEQ >6CE7           -> Yes, jump
6B2F  D6 75 09           CB >09,>8375       -> FCTN D key?
6B32  6C E3              JEQ >6CE3           -> Yes, jump
6B34  D6 75 04           CB >04,>8375       -> FCTN 2 (INSERT) key?
6B37  4B 3C              JNE >6B3C           -> No, jump
6B39  BE 64 01           MOVB >01,>8364
6B3C  D6 75 03           CB >03,>8375       -> FCTN 1 (DELETE) key?
6B3F  4B 94              JNE >6B94           -> No, jump
6B41  86 A3 74           CLRB V>0374
6B44  86 60              CLRB >8360
6B46  D5 2A 61           C >8361,>832A
6B49  6B 8E              JEQ >6B8E
6B4B  D6 B0 2A 7F        CB >7F,V>832A
```

```
6B4F  4B 53              JNE >6B53
6B51  93 2A              DEC >832A
6B53  BD 5C 2A           MOV >832A,>835C
6B56  A5 5C 61           S >8361,>835C
6B59  34 5C B0 61 E0 01 61 COPY V>0001(>8361),V>8361,>835C
6B60  BD 5C 61           MOV >8361,>835C
6B63  B2 5D FC           ANDB >FC,>835D
6B66  B6 5D 1D           ORB >1D,>835D
6B69  C9 5C 2A           SL >832A,>835C
6B6C  6B 7A              JEQ >6B7A
6B6E  C0 E0 04 5C B0 5C  SWAPB V>835C,V>0004(>835C)
6B74  A3 5C 00 20        A >0020,>835C
6B78  4B 69              JNE >6B69
6B7A  93 2A              DEC >832A
6B7C  D6 B0 2A 7F        CB >7F,V>832A
6B80  4B 86              JNE >6B86
6B82  A7 2A 00 04        S >0004,>832A
6B86  D6 B0 2A 80        CB >80,V>832A
6B8A  6A A3              JEQ >6AA3
6B8C  91 2A              INC >832A
6B8E  BE B0 2A 80        MOVB >80,V>832A
6B92  4A A3              JNE >6AA3
6B94  D6 75 07           CB >07,>8375       -> FCTN 3 (ERASE) key?
6B97  4B B3              JNE >6BB3           -> No, jump
6B99  86 A3 74           CLRB V>0374
6B9C  D6 B0 2A 7F        CB >7F,V>832A
6BA0  6B A6              JEQ >6BA6
6BA2  BE B0 2A 80        MOVB >80,V>832A
6BA6  93 2A              DEC >832A
6BA8  C9 2A 20           SL >8320,>832A
6BAB  6B 9C              JEQ >6B9C
6BAD  91 2A              INC >832A
6BAF  86 60              CLRB >8360
6BB1  4A 9E              JNE >6A9E
6BB3  D6 75 0D           CB >0D,>8375       -> ENTER Key?
*                                              THIS IS WHAT WE WANT!
6BB6  6B C2              JEQ >6BC2           -> Yes, jump
6BB8  D6 75 0B           CB >0B,>8375       -> FCTN E Key?
6BBB  6B C2              JEQ >6BC2           -> Yes, jump
6BBD  D6 75 0A           CB >0A,>8375       -> FCTN X (DOWN) key?
6BC0  4A A8              JNE >6AA8           -> No, jump back.
*                                              No valid key has  been
*                                                pressed.
*                                                SCAN will restart
6BC2  D5 2A 5E           C >835E,>832A    -> ENTER key entry point
6BC5  4B CF              JNE >6BCF
6BC7  D6 B0 2A 80        CB >80,V>832A     -> Is last char
*                                                in string a Blank?
6BCB  6B CF              JEQ >6BCF          -> Yes, ok
6BCD  91 2A              INC >832A          -> Pointer to end of
```

```
*                                                  string will go up to
*                                                  the last blank.
6BCF  00                RT                  -> GPL INPUT is over:
```

WHEN BASIC BUILDER TAKES CONTROL

When BASIC BUILDER takes control, as said above, it is because of an Interrupt. At that time, the Micro Processor has jus executed the LIMI 2 instruction at >0070 ROM. At the end of Interrupt, the TMS 9900 Program Counter will be at >0072 ROM. There it will restart execution and the decoding of the next GPL software, beginning with the first byte of the next instruction. On the other hand, until each GPL instruction has not been fully executed, the GPL interpreter will not be allowed to go to >0070.

Therefore, we can assume that BASIC BUILDER will come into action when the GPL instruction pointed by >9802 (GROM Read Address) has NOT been executed yet, while the previous is FULLY executed. This is very handy, because we can modify the return address of GPL without interfering dangerously with completion of each GPL instructions.

Up to a certain extent, and provided we know what we are doing, we can skip some GPL instructions. This is what BASIC BUILDER does before returning: it knows has to go to the ENTER routine (>6BC2 GROM), so it goes directly there, without letting the GPL software decode what key was pressed by the user. I hope that this remark will be handy for some other user.

GPL ERROR HANDLING

When the GPL software issues an error, because of a basic line too long or for any other reason, this error is detected by BASIC BUILDER the next time it takes control. As a matter of fact, the address at >838E, in the GPL return stack, becomes >6E7B (in my ATA 1183 Extended Basic Cartridge).

Well, when this happens, then BASIC BUILDER knows that at >8354 there is the GROM address of the Error Message that has just been issued. BASIC BUILDER will save this GROM address in its ERROR STACK, along with the basic line number that determined the error. At the end of BASIC BUILDER, all these GROM addresses will be used to print all the error messages for the user benefit.

BASIC BUILDER SOURCE FILES

The second side of the diskette contains the BASIC BUILDER SOURCE FILES. The COPY DIRECTIVE is the file BUILD-S. If you use this file when assembling, it will call all the other source files, from BUILD-A through BUILD-E, to form the object file BUILD-O. This file has to be UNCOMPRESSED (do not use "C" option), since it has to be loaded by Extended Basic.

The source files are fully documented. Each line has its own comment. It has been a great effort, but I wanted to make sure assembly programmers could profit from my work.

If you modify or improve BASIC BUILDER, I would be most glad if you sent me your own source files. Thank you!

Disk 81. Contents of file BUILD-DOC6

BASIC BUILDER WARRANTY

BASIC BUILDER is delivered on a "AS IS" basis. It is not surely free from defects and it might not do what you expect it to.

However:

Suppose you find an application where BASIC BUILDER will not build a basic program out of your LISTFILE. Suppose also that if you typed directly the content of that LISTFILE on the keyboard with the Extended Basic Module inserted you would obtain a correct extended basic program. Obviously, in this case BASIC BUILDER does not work as it should.

Then, please drop me a line.

- Make sure to enclose the portion of your LISTFILE (on paper) that has given you problems when assembled by BASIC BUILDER.
- Please add a short explanation of the flaw.
- I will see to it and fix the BASIC BUILDER into a new version that will take care of that flaw.
- If you add an empty diskette and enough postage ($ 2.00 to US from Italy) you will receive the new version, free of charge, on the diskette you provided. In case you send a diskette, please save the LISTFILE to it.

TABLE OF CONTENTS

Disk 81. Contents of file BUILD-DOCS

```
.IF DSK1.BUILD-DOC1
.IF DSK1.BUILD-DOC2
.IF DSK1.BUILD-DOC3
.IF DSK1.BUILD-DOC4
.IF DSK1.BUILD-DOC5
.IF DSK1.BUILD-DOC6
```

Disk 81. Contents of file FROGGER/L

```
100 CALL CLEAR :: CALL SCREEN(12)
110 DISPLAY AT(10,6):"TI-99/4A FROGGER " :: DISPLAY AT(12,4):"EXTENDED BASIC REQ
UIRED"
111 DISPLAY AT(14,6):"JOYSTICKS REQUIRED"
114 FOR A=1 TO 800 :: NEXT A
120 DISPLAY AT(16,4):"program 1983 p.pheby"
121 FOR A=1 TO 800 :: NEXT A
125 CALL CLEAR :: DISPLAY AT(1,5):"   INSTRUCTIONS   " :: DISPLAY AT(13,1):"USE|
JOYSTICKS TO CROSS ROAD"
126 DISPLAY AT(15,1):"THEN USE FIRE BUTTON TO HOP LOGS"
127 DISPLAY AT(20,1):"*** AIM FOR REAR OF LOGS ***"
130 DISPLAY AT(22,7):"PRESS ANY KEY" :: CALL KEY(0,K,S):: IF S=0 THEN 130
135 CALL CLEAR :: CALL SCREEN(12):: DISPLAY AT(5,4):"PRESS KEY FOR SKILL LEVEL"
136 DISPLAY AT(7,4):"1 TO 4"
137 DISPLAY AT(9,4):"HIGHER NUMBERS GIVE FASTER  TRAFFIC AND LESS FROGS"
138 ACCEPT AT(7,12)SIZE(1)VALIDATE("1234"):SK$ :: SK=VAL(SK$)
140 CALL CLEAR :: CALL SCREEN(2):: SP=2+SK :: K=SK :: FRG=10-SK
150 CALL CHAR(112,"FFFFFFFFFFFFFFFF",113,"0000000000000000")
160 CALL CHAR(120,"0000000000087F7C7F777F730000000000000000033FFFF7F3FFFF5200000
000")
170 CALL CHAR(124,"000000000080FFE7FFBFFF73000000000000E0202030FEE6FF66F65C00000
000")
180 CALL CHAR(128,"0000000307552723 3F070F5F233F0100000000C0E0BBE4C4FCE0F0FAC4FC8
000")
190 CALL CHAR(132,"0003075527233F0303030303031F112800C0E0AAE4C4FCC0C0C0C0C0C0F88
814")
200 CALL CHAR(136,"0000000002024363000000000000000000000000000080C000000000000000
000")
210 CALL CHAR(140,"0000002030383C1E1F0D0F0B000000000000000040C1C3C78E0E0E04000000
000")
220 CALL CHAR(95,"00FF00000000FF00")
230 CALL CHAR(96,"000000007077277F")
240 CALL CHAR(97,"7F27777000000000")
250 CALL CHAR(98,"000000000EEEE4FF")
260 CALL CHAR(99,"FFE4EE0E00000000")
270 CALL COLOR(1,2,1,0,6,1,8,8,1,9,8,1,11,6,2)
280 CALL HCHAR(12,1,95,32):: CALL HCHAR(23,1,95,32)
290 FOR D=3 TO 10 :: CALL HCHAR(D,1,112,32):: NEXT D
300 FOR A=3 TO 4 :: CALL COLOR(A,16,1):: NEXT A
310 CALL HCHAR(1,1,113,32):: CALL HCHAR(2,1,113,32):: CALL HCHAR(11,1,113,32)
320 FOR Q=2 TO 32 STEP 2 :: CALL HCHAR(17,Q,95):: NEXT Q
330 DISPLAY AT(1,12):FRG
340 CALL MAGNIFY(3)
350 CALL SPRITE(#1,128,4,178,100)
360 CALL SPRITE(#18,124,11,17,10,0,2*SP,#19,124,11,17,90,0,2*SP)
370 CALL SPRITE(#20,124,11,33,1,0,SP,#21,120,11,49,120,0,3*SP)
380 CALL SPRITE(#10,124,11,33,78,0,SP)
390 CALL SPRITE(#13,120,11,49,90,0,3*SP,#14,124,11,65,168,0,2*SP)
```

```
400 CALL SPRITE(#2,99,12,97,150,0,5*SP)
410 CALL SPRITE(#3,99,10,113,100,0,6*SP,#6,99,14,113,50,0,6*SP)
420 CALL SPRITE(#4,99,5,161,20,0,-4*SP,#7,99,12,161,100,0,-4*SP)
430 CALL SPRITE(#5,99,12,145,100,0,-6*SP,#8,99,16,145,50,0,-6*SP)
440 CALL POSITION(#1,H,J):: IF H<=88 THEN CALL MOTION(#1,0,0):: CALL LOCATE(#1,8
1,J):: GOTO 580
450 CALL JOYST(1,X,Y):: IF X=0 AND Y=0 THEN CALL PATTERN(#1,128)ELSE CALL PATTER
N(#1,132)
460 CALL MOTION(#1,-ABS(Y*4),X*4)
470 CALL COINC(ALL,G):: IF G=0 THEN 440 ELSE 480
480 CALL SOUND(100,-5,1):: CALL MOTION(#1,0,0):: CALL COLOR(#1,9):: FOR G=1 TO
2
00 :: NEXT G
490 FRG=FRG-1 :: DISPLAY AT(1,12):FRG
500 IF FRG=0 THEN 510 ELSE 530
510 CALL DELSPRITE(ALL):: CALL CLEAR :: CALL COLOR(8,2,1):: CALL SCREEN(15):: DI
SPLAY AT(8,4):"NO FROGS LEFT TO PLAY AGAIN     PRESS Y/N"
520 CALL KEY(0,K,S):: IF S=0 THEN 520 ELSE IF K=121 THEN RUN 135 ELSE STOP
530 CALL LOCATE(#1,170,100):: CALL COLOR(#1,4):: GOTO 440
540 FRG=FRG+1 :: DISPLAY AT(1,12):FRG :: K=K+1 :: IF K=12 THEN 1000 ELSE SP=SP+1
550 CALL MOTION(#2,0,5*SP,#3,0,6*SP,#4,0,-4*SP,#6,0,6*SP,#7,0,-4*SP,#5,0,-6*SP,#
8,0,-6*SP)
560 CALL MOTION(#10,0,SP,#13,0,3*SP,#14,0,2*SP,#18,0,2*SP,#19,0,2*SP,#20,0,SP,#2
1,0,3*SP)
570 FOR G=1 TO 100 :: NEXT G :: CALL LOCATE(#1,170,100):: CALL COLOR(#1,4):: GOT
O 440
580 CALL POSITION(#1,H,J)
590 CALL KEY(1,K1,S1):: IF S1=0 THEN CALL PATTERN(#1,128):: GOTO 580 ELSE CALL
P
ATTERN(#1,132):: A=0
600 FL=0 :: H=H-16 :: CALL LOCATE(#1,H,J):: IF H=1 THEN CALL COLOR(#1,1):: GOTO|
540
610 CALL COINC(ALL,G):: IF G<>0 THEN 620 ELSE IF H=81 THEN FL=0 :: GOTO 580 ELSE
 FL=0 :: GOTO 480
620 IF FL=1 THEN 580
630 IF J+3*SP>256 THEN FL=0 :: GOTO 480
640 ON (H-1)/16 GOTO 650,660,670,680,650
650 A=2*SP :: CALL LOCATE(#1,H,J+SP):: CALL MOTION(#1,0,A):: FL=1 :: CALL SOUND(
10,3000,1):: GOTO 580
660 A=SP :: CALL LOCATE(#1,H,J+SP):: CALL MOTION(#1,0,A):: FL=1 :: CALL SOUND(10
,3000,1):: GOTO 580
670 A=3*SP :: CALL LOCATE(#1,H,J+SP):: CALL MOTION(#1,0,A):: FL=1 :: CALL SOUND(
10,3000,1):: GOTO 580
680 A=2*SP :: CALL LOCATE(#1,H,J+SP):: CALL MOTION(#1,0,A):: FL=1 :: CALL SOUND(
10,3000,1):: GOTO 580
1000 CALL CLEAR :: CALL CHARSET
1010 DISPLAY AT(2,1):"YOU HAVE COMPLETED ALL THE SCREENS"
1012 DISPLAY AT(4,1):"NOW TRY AGAIN BUT DON`T KILL ANY FROGS" :: RUN 135
```

Disk 82. Textload/EA5Load

Version: *Author*: Curtis Provance
Requires: *Language*: *Updated*: 07/15/87

Textload is a batch loader for use in Extended BASIC, and EASLoad is a very powerful program loader for Extended BASIC. Complete source code and documentation for both included.

```
dskdir. v2.0. 12-dec-96
Disk name               = TEXTLOADER
Sectors total           = 360
Sectors used            = 304
Sectors available       = 54
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 AUTOEXEC      5 DIS/VAR 80   >022 004
002 >003 EA5LOAD       5 PROGRAM      >026 004
003 >004 EA5LOADSRC   36 DIS/VAR 80   >02a 035
004 >005 LOAD         13 PROGRAM      >04d 012
005 >006 TEXTLOAD/D   76 DIS/VAR 80   >059 075
006 >007 TEXTLOADER   13 PROGRAM      >0a4 012
007 >008 TEXTLOADS1  111 DIS/VAR 80   >0b0 110
008 >009 TEXTLOADS2   45 DIS/VAR 80   >11e 044
```

Disk 82. Contents of file AUTOEXEC

```
REM THIS FILE IS CALLED     AUTOEXEC AND IS BEING LOADEDAUTOMATICALLY
REM YOUR BATCH FILE CAN     CONTAIN ANY COMMANDS        ALLOWABLE
FOR X=1 TO 10::CALL SOUND(1000,110,30)::NEXT X
100 DIM B$(140)::CALL CLEAR
110 PRINT "Would you like to print the documentation? (5 pages) Y"
120 CALL KEY(3,K,S)::ACCEPT AT(23,26)SIZE(-1)VALIDATE("YN"):A$
130 CALL CLEAR::IF A$="N" THEN GOTO 210
140 DISPLAY AT(10,2):"Enter printer name:"::ACCEPT A$::A$=A$&".LF"
150 OPEN #1:"DSK1.TEXTLOAD/D",INPUT::OPEN #2:A$,OUTPUT
160 FOR X=1 TO 140
170 IF EOF(1)THEN CLOSE #1::DONE=-1::GOTO 190
180 LINPUT #1:B$(X)::NEXT X
190 FOR Y=1 TO X-1::PRINT #2:B$(Y)::NEXT Y::IF NOT DONE THEN 160
200 CLOSE #2
210 DISPLAY AT(24,1)ERASE ALL:"Type CALL LINK(""HELP"")"
RUN
```

Disk 82. Contents of file EA5LOADSRC

```
       AORG >FFE7->25D
       EVEN
       DEF  LOAD
LOAD
       CLR  R0
       LI   R1,1
       LI   R2,NAME
       BLWP @>2014       STRING REFERENCE
       MOVB @DRIVE,@CHRNUM    LOAD DRIVE NUMBER FOR DSKx.CHARA1
       MOVB @NAME,@>83E1
       A    R0,R2
       MOV  R2,@ENDCHR
       LI   R0,VDPDAT
       MOVB *R0+,*R15
       MOVB @DRIVE,R1
       MOVB *R0+,*R15
       AI   R1,->3000    REMOVE ASCII CODE FROM DRIVE NUMBER
       MOV  @>8370,R2
       MOVB R1,@>8C00
       AI   R2,->1000
       MOVB *R0+,*R15
       MOVB R2,@MAXSIZ
       MOVB *R0+,*R15
       MOVB @>83E5,@MAXSIZ+1
       MOVB R1,@>8C00
       LI   R1,>300
LOAD1
       MOVB *R0+,*R15
       CI   R0,VDPEND
       JLT  LOAD1
       MOVB *R0+,R2
LOAD2
       MOVB R2,@>8C00
       DEC  R1
       JGT  LOAD2
       MOVB *R0+,@>8C00
LOOP
       MOVB *R0+,R1
       MOVB *R0+,*R15
       SRA  R1,8
       MOVB *R0+,*R15
       MOVB *R0+,R2
LOAD4
       MOVB R2,@>8C00
       DEC  R1
       JGT  LOAD4
       LI   R1,47
LOAD5
```

```
       MOVB *R0+,@>8C00
       DEC  R1
       JGT  LOAD5
       MOVB *R0+,*R15
       MOVB *R0+,R1
       MOVB *R0+,*R15
       MOVB *R0,R2
       SRA  R2,8
LOAD6
       MOVB R1,@>8C00
       DEC  R2
       JGT  LOAD6
       MOV  @UPPERC,@>834A
       BLWP @GPLLNK
       DATA >18
       MOV  @LOWERC,@>834A
       BLWP @GPLLNK
       DATA >4A
       MOV  @CHRPTR,@>8356
       BLWP @DSRLNK
       DATA 8
       LI   R0,>E081
       MOVB R0,*R15
       SWPB R0
       MOVB R0,*R15
       MOV  @POINTR,@>8356
       BLWP @DSRLNK
       DATA 8
       JNE  FILEOK
       BLWP @0
FILEOK
       MOVB @BUFFER+1,*R15
       LI   R1,>83E0+8   R4 OF GPL WORKSPACE
       MOVB @BUFFER,*R15
       LI   R2,6
FILE2
       MOVB @>8800,*R1+
       DEC  R2
       JGT  FILE2
       MOV  @START,@BRANCH
       JNE  FILE3
       MOV  R6,@START
       MOV  R6,@BRANCH
FILE3
       INC  R4           FLAG TO INDICATE LAST FILE
       JNE  FILE4
       MOV  @CONTIN,@BRANCH
FILE4
       MOV  R6,R0
       A    R5,R0
```

```
       CI   R0,LOAD
       JL   FILE5
       DEC  R4
       JEQ  FILE5
       BLWP @0
FILE5
       LI   R0,MOVER
       LI   R1,>8300
       LI   R2,CONTIN-MOVER
FILE6
       MOV  *R0+,*R1+
       DECT R2
       JGT  FILE6
       B    @>8300
MOVER
       MOVB @>8800,*R6+
       DEC  R5
       JGT  MOVER
BRANCH EQU  $+2
       B    @0
CONTIN DATA MORFIL
MORFIL
       LI   R0,COLORS
       MOV  @ENDCHR,R1
       AB   @B01,*R1
       B    @LOOP
GPLWS  EQU  >83E0        GPL WORKSPACE
GR4    EQU  GPLWS+8      GPL WORKSPACE R4
GR6    EQU  GPLWS+12     GPL WORKSPACER6
STKPNT EQU  >8373        GPL STACK POINTER
LDGADD EQU  >60          LOAD AND EXECUTE GROM ADDRESS ENTRY POINT
XTAB27 EQU  >200E        LOW MEMORY XML TABLE LOCATION 27
GETSTK EQU  >166C

GPLLNK DATA GLNKWS       R7 SET UP BLWP VECTORS
       DATA GLINK1       R8

RTNAD  DATA XMLRTN       R9 ADDRESS WHERE GPL XML RETURNS TO US
GXMLAD DATA >176C        R10 GROM ADDRESS FOR GPL XML (0F 27 OPCODE)
       DATA >50          R11 INITIALIZED TO >50 WHERE PUTSTK ADDRESS RESIDES

GLNKWS EQU  $->18        GPLLNK'S WORKSPACE OF WHICH ONLY
       BSS  >08          REGISTERS R7 THROUGH R15 ARE USED

GLINK1 MOV  *R11,@GR4    PUT PUTSTK ADDRESS INTO R4 OF GPL WS
       MOV  *R14+,@GR6   PUT GPL ROUTINE ADDRESS IN R6 OF GPL WS
       MOV  @XTAB27,R12  SAVE THE VALUE AT >200E
       MOV  R9,@XTAB27   PUT XMLRTN ADDRESS INTO >200E
       LWPI GPLWS        LOAD GPL WS
       BL   *R4          SAVE THE CURRENT GROM ADDRESS ON STACK
       MOV  @GXMLAD,@>8302(R4) PUSH GPL XML ADDRESS ON STACK FOR GPL RETURN
```

```
       INCT @STKPNT      ADJUST THE STACK POINTER
       B    @LDGADD      EXECUTE OUR GPL ROUTINE

XMLRTN MOV  @GETSTK,R4   GET GETSTK POINTER
       BL   *R4          RESTORE GROM ADDRESS OFF THE STACK
       LWPI GLNKWS       LOAD OUR WORKSPACE
       MOV  R12,@XTAB27  RESTORE >200E
       RTWP              ALL DONE - RETURN TO CALLER
PUTSTK EQU  >50          PUSH GROM ADD TO STACK POINTER
TYPE   EQU  >836D        DSRLNK TYPE BYTE FOR GPL DSRLNK
NAMLEN EQU >8356         DEVICE NAME LENGTH POINTER IN VDP PAB
VWA    EQU  >8C02        VDP WRITE ADDRESS LOCAION
VRD    EQU  >8800        VDP READ DATA BYTE LOCATION
GR4LB  EQU  >83E9        GPL WORKSPACE R4 LOWER BYTE
GSTAT  EQU  >837C        GPL STATUS BYTE LOCATION

DSRLNK DATA DSRWS,DLINK1 SET BLWP VECTORS

DSRWS  EQU  $            START OF DSRLNK WORKSPACE
DR3LB  EQU  $+7          R3 LOWER BYTE OF DSRLNK WORKSPACE
DLINK1 MOV  R12,R12      R0 HAVE WE ALREADY LOOKED UP THE LINK ADDRESS?
       JNE  DLINK3       R1 YES! SKIP THE LOOK UP ROUTINE

* THIS SECTION OF CODE IS ONLY EXECUTED ONCE TO FINE THE GROM ADDRESS FOR THE
*
* GPL DSRLNK - WHICH IS PLACED AT DSRADD AND R12 IS SET TO >2000 TO INDICATE
*
* THAT THE ADDRESS IS FOUND AND TO BE USED AS A MASK FOR EQ & CND
*

       LWPI GPLWS        R2,R3 ELSE LOAD GPL WORKSPACE
       MOV  @PUTSTK,R4   R4,R5 RESTORE CURRENT GROM ADDRESS ON THE STACK
       BL   *R4          R6
       LI   R4,>11       R7,R8 LOAD R4 WITH ADDRESS OF LINK ROUTINE VECTOR
       MOVB R4,@>402(R13)     R9,R10 SET UP GROM WITH ADDRESS FOR VECTOR
       JMP  DLINK2       R11 JUMP AROUND R12-R15
       DATA 0            R12 CONTAINS >2000 FLAG WHEN SET
       DATA 0,0,0        R13-R15 CONTAINS WS, PC, & ST FOR RTWP
DLINK2 MOVB @GR4LB,@>402(R13) FINISH SETTING UP GROM ADDRESS
       MOV  @GETSTK,R5   TAKE SOME TIME & SET UP GETSTK POINTER
       MOVB *R13,@DSRAD1 GET THE GPL DSR LINK VECTOR
       INCT @DSRADD      ADJUST IT TO GET PADT GPL FETCH INSTRUCTION
       BL   *R5          RESTORE THE GROM ADDRESS OFF THE STACK
       LWPI DSRWS        RELOAD DSRLNK WORKSPACE
       LI   R12,>2000    SET FLAG TO SIGNIFY DSRLNK ADDRESS IS SET
DLINK3 INC  R14          ADJUST R14 TO POINT TO CALLER'S DSR TYPE BYTE
       MOVB *R14+,@TYPE  MOVE IT INTO >836D FOR GPL DSRLNK
       MOV  @NAMLEN,R3   SAVE VDP ADDRESS OF NAME LENGTH
       AI   R3,-8        ADJUST IT TO POINT TO PAB FLAG BYTE
       BLWP @GPLLNK      EXECUTE DSR LINK
```

```
DSRADD BYTE >03          HIGH BYTE OF GPL DSRLNK ADDRESS
DSRAD1 BYTE >00          LOWER BYTE OF GPL DSRLNK ADDRESS
****** ERROR CHECK
       MOVB @DR3LB,@VWA  SET UP LSB OF VDP ADD FOR ERROR FLAG
       MOVB R3,@VWA      SET MSB OF VDP ADD FOR ERROR FLAG
       SZCB R12,R15      VLEAR EQ BIT FOR ERROR REPORT
       MOVB @VRD,R3      GET PAB ERROR FLAG
       SRL  R3,5         ADJUST IT TO 0-7 ERROR CODE
       MOVB R3,*R13      PUT IT INTO CALLER'S R0 (MSB)
       JNE  SETEQ        IF IT NOT ZERO, SET EQ BIT
       COC  @GSTAT,R12   ELSE TEST CND BIT FOR LINK ERROR (00)
       JNE  DSREND       NO ERROR, JUST RETURN
SETEQ  SOCB R12,R15      ERROR SO SET CALLER'S EQ BIT
DSREND RTWP              ALL DONE - RETURN TO CALLER
START  DATA 0            STORES THE STARTING ADDRESS FOR THE PROGRAM
ENDCHR DATA 0            ADDRESS OF LAST CHARACTER WILL BE STORED HERE
CHRPTR DATA >3A9         POINTS TO LENGTH BYTE FOR DSKx.CHARA1
POINTR DATA >3A9+22      POINTS TO LENGTH BYTE FOR E/A #5 PROGRAM
UPPERC DATA >900
LOWERC DATA >B00
VDPDAT DATA >EB7E        LAST DRIVE ACCESSED ADDRESS (REVERSED) WITH WRITE BIT
       DATA >F57F        LAST DRIVE ACCESSED ADDRESS (REVERSED) WITH WRITE BIT
       DATA >A081        TURN OFF SCREEN
       DATA >0E83        DEFAULT FOR COLOR TABLE
       DATA >0184        PATTERN DEFINITION TABLE
       DATA >F587        WHITE ON BLUE FOR TEXT MODE
       DATA >0040        SCREEN START ADDRESS (REVERSED) WITH WRITE BIT
VDPEND BYTE >20          SPACE CHARACTER TO FILL THE SCREEN
       BYTE >D0          NO SPRITES IN THE BEGINNING
COLORS BYTE >20
       BYTE >80,>43,>F5
       BYTE 5,0,7,>FA,0,0,8,0,0,11   LOAD CHARA1 FROM SAME DRIVE INTO VDP >7FA
       TEXT 'DSK'
CHRNUM TEXT '1.CHARA1'
B01    BYTE 1
       BYTE >05,>00      LOAD OPCODE
BUFFER BYTE >10,>00      BUFFER ADDRESS
       BYTE 0,0          EXTRANEOUS
MAXSIZ BYTE >00,>00,>00  MAXIMUM BYTES TO BE LOADED - DEPENDS ON CALL FILES(X)
NAME   BYTE 15           MAXIMUM LENGTH
       TEXT 'DSK'
DRIVE  TEXT '1'
       BSS  11
       BYTE >F0,>FF,>48,8
       END
```

Disk 82. Contents of file TEXTLOAD/D

TEXTLOADER and EA5LOAD

by PARAGON COMPUTING
17 Constance Street
Merrimack, New Hampshire 03054

Programs and documentation by Curtis Alan Provance

A minimum donation of $5.00 puts you on our list for future releases which include a Flight Simulator and a true DOS.

GENERIC

TEXTLOADER is a machine language routine which reads a text file (such as might be created by TI-Writer) and loads each line into the Extended BASIC command interpreter. TEXTLOADER preprocesses the lines of text and ignores the following:

1) Leading and trailing spaces
2) Lines whose first non-space character is a remark (!)
3) Carriage returns and any characters following them
4) The TAB record written to text files by some editors

Another special character is the 'required space' code denoted by '^'. This code is only active at the end of a line and signals the loader that a space should be written there. Any preceding spaces will also be written. This feature is useful for ensuring a space between the last word of a line and the first word of the next line. TEXTLOADER runs until

1) you press the CLEAR key (FCTN-4),
2) there are no more lines of text or
3) the first three non-space characters in a line are RUN or OLD. Three options let you select between batch files, loading a new program from text, or merging program lines into an existing program. Two other routines provide a help menu and let you continue processing if you halted execution with the CLEAR key (FCTN-4).

Minimum requirements include an Extended BASIC module and memory expansion. Any display, variable 80 files — such as disk files and RS232/PIO files — may be loaded.

Compatibility with Myarc and New Horizon RAM disks has been checked.

This program taps directly into certain areas of the Extended BASIC module and may therefore not work with your particular module. It does work well with TI's Extended BASIC version 110 and MICROPAL's Extended BASIC version 110. If this program doesn't work with your module, please contact Paragon Computing at the above address and we will develop a version for you at no charge.

FILES ON THIS DISK:

AUTOEXEC	batch file run with LOAD
EA5LOAD	loads EA #5files in Extended BASIC
EA5LOADSRC	source code for EA5LOAD
LOAD	copy of TEXTLOADER set up to load AUTOEXEC
TEXTLOAD/D	this documentation
TEXTLOADER	the TEXTLOADER program

ROUTINES

CALL LINK("BATCH",dsk#.name)

BATCH opens the file specified and reads the entire file into memory. Any program already in memory is discarded. Control is then passed back to the Extended BASIC command interpreter. At this point, each line of text from your text file will be written to the command line one at a time. Batch files are useful for initializing your system upon powerup, or configuring your system to run a particular program. For example, if you own a Myarc or Foundation RAM disk, you could use a batch file to initialize the disk before loading and running another program. A batch file for the Myarc 128K RAM disk might look like this:

```
CALL PART(90,6)
CALL EMDK(2)
RUN "DSK1.SECONDLOAD"
```

If you have a program which requires more string space than is normally available in VDP RAM, you may have to do a CALL FILES.. etc. A batch file can do this for you:

```
CALL FILES(1)
NEW
RUN "DSK1.BIGPROGRAM"
```

With the exception of remarks beginning with '!', anything you can type in the command line can be put in a batch file and be loaded for you.

You may link to BATCH from the command mode or from within a program. The LOAD program is nothing more than TEXTLOADER configured to load a batch file called AUTOEXEC.

CALL LINK("OLD",dsk#.name)

OLD operates identically to BATCH with one major exception: lines are not necessarily written one at a time. A special feature of OLD is the ability to tell when a new program line starts. It does this by looking for an integer number from 1 to 32767, separated from the rest of the text by at least one space. For example, the following lines would all be considered as the start of a program line:

```
1 !This is a program line.
31555 PRINT "So is this!"
28 FOR X=1 TO 10::
```

All three examples are indeed valid program line starts. However, the third example would need at least one additional line to finish it - or else the colons (::) deleted.

If no number or an invalid number is encountered, that line of text will be added to the previous line before being written to the screen. This feature lets you develop programs with indentations or multi-line statements. Compare the 'readability' of this short program in its listed form, and its improved form (next page):

```
100 DISPLAY AT(2,1)ERASE ALL:"NAME OF ORIGINAL 80 COLUMN
LISTING?" :: ACCEPT AT (3,10)BEEP:OLD$
110 DISPLAY AT(5,1):"NAME OF NEW 28 COLUMN LISTING?" ::
ACCEPT AT(6,10)BEEP:NEW$
120 OPEN #1:OLD$,INPUT :: OPEN #2:NEW$
130 PRINT #2:CHR$(27);"G";CHR$(27);"N";CHR$(6);CHR$(15)
140 IF EOF(1)THEN 210
150 LINPUT #1:A$
160 ON ERROR 170 :: A=VAL(SEG$(A$,1,POS(A$," ",1))):: IF
T$<>"" THEN 190 ELSE 180
170 RETURN 180
180 T$=T$&A$ :: IF LEN(A$)=80 THEN 140 ELSE A$=""
190 IF LEN(T$)THEN PRINT #2:CHR$(14);TAB(2);SEG$(T$,1,28)::
T$=SEG$(T$,29,255):: GOTO 190
200 IF A$<>"" THEN 180 ELSE 140
210 CLOSE #1 :: PRINT #2:CHR$(27);"O";CHR$(12):: CLOSE #2
```

The same program indented and broken:

```
100 DISPLAY AT(2,1)ERASE ALL:
"NAME OF ORIGINAL 80 COLUMN LISTING?" ::
ACCEPT AT (3,10)BEEP:OLD$
110 DISPLAY AT(5,1):
"NAME OF NEW 28 COLUMN LISTING?" ::
ACCEPT AT(6,10)BEEP:NEW$
120 OPEN #1:OLD$,INPUT ::
OPEN #2:NEW$
130 PRINT #2:CHR$(27);"G";CHR$(27);"N";CHR$(6);CHR$(15)
140 IF EOF(1)THEN 210
```

```
150 LINPUT #1:A$
160 ON ERROR 170 ::
A=VAL(SEG$(A$,1,POS(A$," ",1))):: 
IF T$<>""
THEN 190
ELSE 180
170 RETURN 180
180 T$=T$&A$ ::
IF LEN(A$)=80
THEN 140
ELSE A$=""
190 IF LEN(T$)
THEN PRINT #2:CHR$(14);TAB(2);SEG$(T$,1,28)::
T$=SEG$(T$,29,255)::
GOTO 190
200 IF A$<>""
THEN 180
ELSE 140
210 CLOSE #1 ::
PRINT #2:CHR$(27);"O";CHR$(12)::
CLOSE #2
```

Imagine how indenting FOR-NEXT loops could improve your programs!

```
100 FOR X=1 TO 10 :: A(X)=0 :: FOR Y=1 TO 10
110 B(X,Y)=0 :: C$(X,Y)="" :: FOR Z=1 TO 10
120 D(X,Y,Z)=0 :: NEXT Z :: CALL SCREEN((X+Y)/2) :: NEXT Y
130 NEXT X :: !JUST AN EXAMPLE - THIS IS A NONSENSE PROGRAM
```

Compared to this!->

```
100 FOR X=1 TO 10 ::
A(X)=0 ::
FOR Y=1 TO 10
110 B(X,Y)=0 ::
C$(X,Y)="" ::
FOR Z=1 TO 10
120 D(X,Y,Z)=0 ::
NEXT Z ::
CALL SCREEN((X+Y)/2) ::
NEXT Y
130 NEXT X ::
!JUST AN EXAMPLE - THIS IS A NONSENSE PROGRAM
```

You may also indent line numbers:

```
100 FOR X=1 TO 10 ::
A(X)=0 ::
FOR Y=1 TO 10
110 B(X,Y)=0 ::
```

```
C$(X,Y)="" ::
FOR Z=1 TO 10
120 D(X,Y,Z)=0 ::
NEXT Z ::
CALL SCREEN((X+Y)/2) ::
NEXT Y
130 NEXT X ::
!JUST AN EXAMPLE - THIS IS A NONSENSE PROGRAM
```

Note that the last line would not be written to your program because the first non-space character is a remark (!).

CALL LINK("MERGE",dsk#.name)

This functions identically to the OLD routine described above, except that any program currently in memory is retained. This routine will merge program lines into your already existing program.

CALL LINK("HELP")

This routine prints a help screen which shows the other four links and briefly describes each of them. You may link to HELP from the command line or from inside a program.

CALL LINK("CONT")

If you press the CLEAR key (FCTN-4) text loading will halt. To resume, type CALL LINK("CONT") and enter.

WARNING! If you have modified the program in memory, processing may not continue.

If you have not modified the program and there are lines remaining to be loaded, processing will resume. If there are no more lines, or you have changed the size of the existing program, then you will be returned to the command line (Extended BASIC monitor).

ERROR HANDLING

If an error occurs while opening a file or reading text from it, TEXT LOADER will:

1) Halt the operation and close the file
2) Clear the screen
3) Display an appropriate warning message and
4) Return control to the Extended BASIC interpreter

If you accidentally specified RS232 or PIO and there is no file being input, your system may appear to be locked up. Press CLEAR (FCTN-4) and you should get the 'BAD DEVICE NAME' error. You may try again with the correct name.

EA5LOAD

This program loads EA option 5 files from Extended BASIC. EA5LOAD does four things most other loaders don't: 1) it loads the complete character set plus the cursor, 2) it then looks for a CHARA1 file. If CHARA1 exists, it will be loaded next. 3) The disk number is stored in two different VDP addresses to compensate for those cards that don't act like the old TI controller. 4) Best of all, it can load programs anywhere in memory — even over itself! The second line of EA5LOAD contains a sample link. You may modify this to load any program you wish.

Paragon Computing is a registered non-profit corporation in the state of New Hampshire. Our charter is to provide programs, education, and assistance to owners of the TI-99/4A computer. Your donations help us purchase the equipment we need to develop flight simulators, DOS's, etc. Thank you for your contribution.

Disk 82. Contents of file TEXTLOADS1

```
LINNUM EQU  7            LINE NUMBER CALCULATED DURING OLD AND MERGE
NEXTLN EQU  8            ADDRESS OF START OF NEXT LINE READ INTO CPU
NEXTAD EQU  9            POINTS TO NEXT ADDRESS IN LIST OF LINES LOADED TO CPU
VDPRD  EQU  13           VDP READ DATA ADDRESS STORED HERE
VDPWD  EQU  14           VDP WRITE DATA ADDRESS STORED HERE
VDPWA  EQU  15           VDP WRITE ADDRESS ADDRESS  STORED HERE
STRREF EQU  >2014        USED TO GET DEVICE.FILENAME FROM XBASIC
XMLLNK EQU  >2018        USED FOR GET STRING SPACE ROUTINE
*
LOWER  EQU  >6AA8        LOWER ADDRESS OF ACCEPT GROM ROUTINE
UPPER  EQU  >6BB6        UPPER ADDRESS OF ACCEPT GROM ROUTINE
CONTIN EQU  >6BCF        CONTINUE AFTER ACCEPT IS COMPLETED
*
MYWSP  BSS  32           SELF EXPLANATORY
R0LB   EQU  MYWSP+1      LOWER BYTE OF THIS REGISTER
R1LB   EQU  MYWSP+3      LOWER BYTE OF THIS REGISTER
R2LB   EQU  MYWSP+5      LOWER BYTE OF THIS REGISTER
LINCNT DATA 0            TRACKS THE NUMBER OF LINES TO LOAD
BATCHF DATA 0            FLAG TO INDICATE  WHETHER  PROGRAM OR  BATCH FILE
HOOKAD DATA HOOK         ADDRESS WHERE HOOK ROUTINE RESIDES
D28    DATA 28           28 CHARACTERS PER PRINTED LINE
POINTR DATA 0            POINTER TO FILE NAME LENGTH BYTE IN PAB
TEN    DATA 10          USED TO MULTIPLY LINE NUMBERS: CONVERT FROM INT TO STR
MERGEF
MORFIL DATA 0            IF MORE FILES, THEN THIS WILL BE ZERO
MOR2DO DATA 0
BRKCHK DATA 0            STORES THE LAST FREE SPACE POINTER DURING A BREAK
COUNT  BYTE 80           COUNT OF BYTES ACTUALLY READ GOES HERE
EDGE   BYTE >7F          EDGE CHARACTER WITH OFFSET ADDED
TABREC
SPACE  BYTE >80         SPACE CHARACTER WITH OFFSET ADDED (START OF TAB RECORD)
REQSPC BYTE >BE          REQUIRED SPACE CHARACTER (^) PLUS OFFSET
REMARK BYTE >81          REMARK (!) CHARACTER WITH OFFSET ADDED
ZERO   BYTE >90          ASCII FOR ZERO "      "     "      "
NINE   BYTE >99          ASCII FOR NINE "      "     "      "
D8     BYTE 8            USED FOR DSRLNK
H7E    BYTE >7E          CHECKS FOR THE CURSOR
HFE    BYTE >FE          USED TO STRIP RESET ALL BUT THE LAST BIT OF A BYTE
STATUS BYTE 9            STATUS OPCODE FOR FILE CHECKING
CR     BYTE >6D          CARRIAGE RETURN PLUS OFFSET
ENTER  BYTE 13           ENTER (ASCII CODE)
B01    BYTE 1            USED TO CHECK FOR 'CLOSE' OPERATIONS
B09    BYTE 9            USED TO CJECK FOR 'STATUS' OPERATIONS
       EVEN
*******************************************************************************
* HOOK: CHECKS INTERPRETER FOR START OF  INPUT ROUTINE.  WRITES LINE TO SCREEN
*
*       AND RETURNS TO INTERPRETER.  AFTER LAST LINE IS WRITTEN, HOOK ADDRESS
```

```
*
*       IS CLEARED TO PREVENT FURTHER INTERRUPTS
*
********************************************************************************
HOOK
       MOVB @2(R13),R8   HIGH BYTE OF CURRENT GPL ADDRESS - CHECKING FOR INPUT
       NOP
       MOVB @2(R13),@>83F1  R8 LOWER BYTE
       DEC  R8           RESTORE TO CORRECT VALUE
       CI   R8,LOWER     ARE WE PAST THE COMMAND LINE'S LOWER BOUNDARY?
       JLT  RTURNZ       NO, WE AREN'T DOING THE COMMAND LINE YET
       CI   R8,UPPER     ARE WE PAST THE COMMAND LINE'S UPPER BOUNDARY?
       JGT  RTURNZ       YES, WE'RE NOT DOING THE COMMAND LINE AT THIS POINT
DOIT
       MOV  R11,@LINKRT  SAVE THE RETURN ADDRESS
       BL   @>20         CHECK FOR THE CLEAR KEY
       JNE  DOIT1        WAN'T PRESSED, SO LOAD THE NEXT LINE
       MOV  @>8330,@BRKCHK    SAVE THE CURRENT END OF LINE NUMBER TABLE
       CLR  @>83C4       CLEAR OUT THE HOOK ADDRESS
       JMP  LINKRT-2     SKIP THIS PART
DOIT1
       LWPI MYWSP        DON'T USE GPL WORKSPACE
       BL   @TOSCRN      MOVE TEXT TO SCREEN
NEXT
       LWPI >83E0        RESTORE GPL WORKSPACE
       LI   R8,CONTIN    PREPARE TO SET GPL ADDRESS AT END OF INPUT ROUTINE
LINKRT EQU  $+2
       LI   R11,0        WILL BE REPLACED BY REAL RETURN ADDRESS
RTURNZ
       MOVB R8,@>402(R13)     FOOL GPL ADDRESS TO THINK ROUTINE IS OVER
       MOVB @>83F1,@>402(R13)  "    "     "     "   "      "    "    "
       RT                RETURN TO INTERRUPT
********************************************************************************
* HELP:   PRINTS THE HELP SCREEN
*
* CONT:   CONTINUES TEXT LOADER AFTER BREAK IN PROGRAM
*
* BATCH:  HANDLES ONLY ONE LINE AT A TIME - OTHERWISE SAME AS OLD
*
* MERGE:  DOESN'T ERASE PROGRAM ALREADY LOADED INTO MEMORY
*
* OLD:    RESETS LINE NUMBER POINTERS AND BEGINS GENERIC PROCESS OF LOADING
*
* CONTRT: RESERVES EIGHT BYTES SO ADDRESS STACK DOESN'T RUN OVER DEFINTIONS
*
********************************************************************************
       DEF  HELP,CONT,BATCH,MERGE,OLD,CONTRT
CONT
       MOV  @LINCNT,R0   GET COUNT OF NUMBER OF LINES REMAINING
       JLT  CONTRT       NO LINES LEFT, SO RETURN TO INTERPRETER
       JEQ  CONTRT       NO LINES LEFT, SO RETURN TO INTERPRETER
```

```
       C    @>8330,@BRKCHK    HAVE ANY LINES BEEN CHANGED SINCE THE BREAK?
       JNE  CONTRT       YES, DON'T CONTINUE. LINES MAY HAVE BEEN MESSED UP
      MOV  @HOOKAD,@>83C4    RESTORE HOOK ADDRESS TO PREPARE FOR NEXT INTERRUPT
CONTRT
       RT                RETURN TO PROGRAM/INTERPRETER
*******************************************************************************
HELP
      LI   R0,>0040     SCREEN STARTING ADDRESS - BYTES REVERSED, WRITE BIT SET
       MOV  R0,R1        MAKE A COPY OF USE LATER
       MOV  R0,R2        MAKE A COPY OF USE LATER
       MOVB R0,*R15      WRITE LOWER BYTE
       SWPB R0           POSITION AND DELAY
       MOVB R0,*R15      WRITE HIGH BYTE
       SLA  R0,1         MAKE R0=>80 WHICH IS SPACE CHARACTER WITH OFFSET
HELP1
       MOVB R0,@>8C00    WRITE SPACE CHARACTERS TO THE SCREEN
       DEC  R1           ONLY NEED TO CLEAR THE TOP TWO LINES AT THIS POINT
       JGT  HELP1
       LI   R1,HELPTX    CPU ADDRESS OF HELP TEXT
HELP2
       MOVB *R1+,@>8C00  MOVE TEXT TO SCREEN - OFFSET HAS BEEN ADDED BY SETUP
       CI   R1,ENDTXT    DON'T COUNT EACH BYTE, LOOK FOR THE END OF THE TEXT
       JLT  HELP2
HELP3
       MOVB R0,@>8C00    START CLEARING WITH SPACES AGAIN
       DECT R2           ONLY NEED TO CLEAR THE LAST TWO LINES THIS TIME
       JGT  HELP3
       RT                ALL DONE
*******************************************************************************
BATCH
       SETO @BATCHF      BATCHF WILL END UP EQUAL TO ZERO IF YOU LINK HERE
       JMP  OLD          SKIP THE MERGE PART - WE WANT TO RESET THE POINTERS
MERGE
       SETO @MERGEF      FLAG TO INDICATE THAT THIS IS NOT AN 'OLD'
OLD
       MOV  R11,@BATCRT  SAVE RETURN ADDRESS
       MOVB @2(R13),@GPL SAVE GPL ADDRESS IN CASE RS232 HAS BEEN CALLED
       MOVB @2(R13),@GPL+1 "   "     "     "   "    "    "    "     "
       DEC  @GPL         RESTORE ADDRESS TO PROPER VALUE
       LWPI MYWSP        GO TO MY WORKSPACE
       INC  @BATCHF      THIS WON'T BE ZERO UNLESS WE LINKED TO BATCH
       INC  @MERGEF      DID WE LINK TO A MERGE FILE?
       JEQ  LOADIT       YES, SO SKIP THIS PART
       MOV  @>8384,@>8330     RESET LINE POINTER TABLES
       MOV  @>8384,@>8332     RESET LINE POINTER TABLES
LOADIT
       CLR  @LINCNT      SO FAR, NO LINES TO LOAD
       LI   NEXTAD,DONE  START OF ADDRESS STACK FOR TEXT LINE POINTERS
       MOV  @>8330,NEXTLN  ADDRESS OF LAST FREE SPACE IN HIGH MEMORY
       AI   NEXTLN,-256  BREATHING ROOM FOR LINES WITH COMMENTS
```

```
        LI   VDPRD,>8800  VDP READ DATA ADDRESS IN MEMORY MAP
        LI   VDPWD,>8C00  VDP WRITE DATA ADDRESS IN MEMORY MAP
        LI   VDPWA,>8C02  VDP WRITE ADDRESS ADDRESS IN MEMORY MAP
        CLR  R0           NOT LOOKING FOR AN ARRAY ELEMENT
        LI   R1,1         FIRST PARAMETER
        LI   R2,FILNAM    PLACE WHERE NAME WILL GO
        MOVB @D28+1,*R2   MAXIMUM FILE LENGTH (A LITTLE LONGER THAN POSSIBLE)
        BLWP @STRREF      GET THE DEVICE.NAME OF FILE TO BE LOADED
        LI   R0,120       80 BYTES FOR BUFFER, 10 FOR PAB, 30 FOR DEVICE/FILE
        MOV  R0,R1        WILL USE THIS AS A COUNTER
        MOV  R0,@>830C    SET UP POINTER FOR GET STRING ROUTINE
       BLWP @XMLLNK      XBASIC USES XMLLNK - NOT GPL - FOR STRING SPACE ROUTINE
        DATA 2            GET STRING SPACE ROUITINE
        MOV  @>831C,R0    VDP BUFFER  ADDRESS IS  IN >831C
        MOV  R0,@BUFFAD   STORE BUFFER ADDRESS IN PAB
        MOVB @R0LB,*VDPWA LOW BYTE OF ADDRESS IN VDP
        ORI  R0,>4000     SET THE WRITE BIT
        MOVB R0,*VDPWA    HIGH BYTE OF ADDRESS
        ANDI R0,>3FFF     ORIGINAL ADDRESS WITHOUT WRITE BIT SET
        AI   R0,89        POINTS TO FILE NAME LENGTH IN VDP
        MOV  R0,@POINTR   START OF BUFFER IN VDP
        LI   R2,FIRSTL    FIRST LINE TO BE LOADED PLUS PAB
BATCH1
        MOVB *R2+,*VDPWD  WRITE FIRST  LINE TO BUFFER
        DEC  R1           COUNT OFF EACH BYTE
        JGT  BATCH1       AND LOOP BACK UNTIL DONE
        BL   @DOFILE      ATTEMPT TO OPEN THE FILE IN USE
        DATA OPEN         ERROR WILL GENERATE A PROMPT TO ABORT OR RETRY
        MOVB @B80,@COUNT  THE FIRST 'RECORD' PLUS BREATHING ROOM (EDGE=>7F)
READLN
        LI   R12,LINBUF   POINTER AT FIRST FREE SPACE
        MOV  R12,R10      WILL USE THIS AS A REFERENCE LATER
READA
        CLR  LINNUM       CLEAR OUT THE LINE NUMBER REGISTER
        MOV  R12,R3       MAKE A COPY OF THE CURRENT LINBUF POSITION
        MOVB @BUFFAD+1,*VDPWA  SET THE VDP FOR A READ
        MOVB @COUNT,R2    GET THE COUNT FROM THE LAST READ FILE
        MOVB @BUFFAD,*VDPWA    FINISH SETTING VDP ADDRESS
        SRA  R2,8         MAKE IT A WORD
        JEQ  EOF          IF NO CHARACTERS, THEN READ IN NEXT FILE
READB
        MOVB *VDPRD,R4    READ BYTE INTO TEMPORARY STRING BUFFER
        CB   R4,@TABREC   TAB RECORD FROM TI-WRITER?
        JEQ  EOF          YES, CLOSE FILE AND CONTINUE
        AI   R4,>6000     ADD SCREEN OFFSET
        CB   R4,@REMARK   IS THIS A REMARK? ('!' CHARACTER?)
        JEQ  EOF          YES, SO READ IN NEXT LINE
        CB   R4,@SPACE    IS THIS A SPACE CHARACTER?
        JNE  STRIPD       NO, ALL SPACES HAVE BEEN STRIPPED OFF
        DEC  R2           COUNT OFF EACH BYTE
        JGT  READB        LOOP BACK UNTIL ALL LEADING SPACES HAVE BEEN REMOVED
```

```
       JMP  EOF          WAS A BLANK LINE, SO LOAD ANOTHER
STRIPD
       CB   R4,@SPACE    GROUP OF NUMBERS SEPARATED BY A SPACE FROM OTHER TEXT?
       JEQ  NEWLIN       YES, CHECK TO SEE IF THIS IS A NEW LINE
       CB   R4,@ZERO     IS THE CHARACTER LESS THAN ZERO?
       JLT  CONT0        YES, ADD OFFSET AND WRITE THE REST OF THIS LINE
       CB   R4,@NINE     IS THE CHARACTER GREATER THAN NINE?
       JGT  CONT0        YES, WRITE THE REST OF THIS LINE
       MOV  LINNUM,R6    PREPARE TO MULTIPLY BY TEN, PRODUCT TO BE IN LINNUM
       MPY  @TEN,R6      MULTIPLY OLD NUMBER BY TEN TO MAKE ROOM FOR NEXT NUMBER
       MOVB R4,R6        GET NEXT NUMBER
       SRL  R6,8         MAKE IT A WORD
       AI   R6,->90      REMOVE SCREEN AND ASCII OFFSET
       A    R6,LINNUM    ADD IT TO THE OLD LINE NUMBER
       MOVB R4,*R3+      MOVE NUMBER INTO STRING BUFFER
       MOVB *VDPRD,R4    READ IN NEXT BYTE (IF ANY)
       AI   R4,>6000     ADD SCREEN OFFSET
       DEC  R2           COUNT OFF EACH BYTE
       JGT  STRIPD       GO BACK FOR MORE
NEWLIN
       DEC  LINNUM       IF LINNUM WERE 0, THEN WILL MAKE IT NEGATIVE
       JLT  CONT0        IF NEGATIVE, THEN IT'S NOT A VALID LINE NUMBER
       INC  LINNUM       IF GREATER THAN >7FFF (32767), WILL BE NEGATIVE >8000
       JLT  CONT0        NOT A VALID LINE NUMBER
       C    R10,R12      ARE WE AT THE START OF A NEW LINE?
       JNE  READZ        NO, SO DON'T WRITE THIS LINE. MOVE BUFFER TO SCREEN
CONT0
       CB   R4,@CR       REACHED A CARRIAGE RETURN?
       JEQ  CONT1        YES, CHECK FOR END OF FILE AND READ NEXT RECORD
       MOVB R4,*R3+      MOVE THIS BYTE INTO POSITION
       MOVB *VDPRD,R4    READ IN NEXT BYTE (IF ANY)
       AI   R4,>6000     ADD SCREEN OFFSET
       DEC  R2           COUNT OFF EACH BYTE
       JGT  CONT0        AND KEEP LOADING UNTIL DONE
CONT1
EOF
       CB   @-1(R3),@REQSPC   WAS THE LAST CHARACTER A REQUIRED SPACE?
       JNE  CONT2        NO, CONTINUE WITH THE MOVING ROUTINE
       MOVB @SPACE,@-1(R3)    REPLACE THE REQUIRED SPACE WITH A SPACE
CONT2
       MOV  R3,R12       SET R12 TO POINT TO THE NEXT FREE SPACE IN LINBUF
       CLR  @MOR2DO      ASSUME THAT THERE IS NO MORE TO DO
       MOVB @MORFIL,R0   MORE FILES?
       JNE  READZ        NO, MOVE THE LAST RECORD THEN CLOSE THE FILE
       BL   @DOFILE      READ IN THE NEXT RECORD
       DATA READ
       SETO @MOR2DO      FLAG TO INDICATE THAT THERE IS AT LEAST ONE OTHER LINE
       MOV  @BATCHF,R0   IS THIS A BATCH FILE
       JNE  READA        NO, JUMP BACK AND LOOK FOR THE NEXT LINE
READZ
```

```
       S    R10,R12      TOTAL NUMBER OF BYTES TO BE MOVED
       JEQ  MORCHK       NO BYTES TO MOVE, SO CHECK FOR MORE FILES TO BE READ
       S    R12,NEXTLN   RESERVE ENOUGH SPACE FOR THIS LINE
       DEC  NEXTLN       PLUS A LENGTH BYTE
       CI   NEXTLN,>A000 ARE WE STILL IN THE HIGH MEMORY AREA?
       JL   CLOSIT       NO. SCRAP THIS LINE AND CLOSE FILE
       C    NEXTAD,@>2004      ARE WE ABOUT TO INTRUDE ON THE DEF SPACE?
       JGT  CLOSIT       YES. SCRAP THIS LINE AND CLOSE FILE
       MOV  NEXTLN,*NEXTAD+    PUSH ADDRESS ONTO LIST ADDRESS STACK
       INC  @LINCNT      MAKING ANOTHER LINE
       MOV  NEXTLN,R11   USE R11 TO POINT TO NEXT BYTE IN CPU SPACE
      MOVB @MYWSP+25,*R11+   R12 LOWER BYTE. MOVE SIZE BYTE INTO STORAGE SPACE
MOVEIT
       MOVB *R10+,*R11+  MOVE TEXT INTO BUFFER
       DEC  R12          COUNT OFF EACH BYTE
       JGT  MOVEIT       UNTIL ALL DONE
MORCHK
       INC  @MOR2DO      IF THIS IS ZERO, THEN WE AREN'T AT THE END YET
       JEQ  READLN       YES, READ NEXT LINE
CLOSIT
       LI   NEXTAD,DONE  RESET TO BEGINNING OF LIST ADDRESS STACK
       CLR  @BATCHF      INITIALIZE PROGRAM FLAG FOR NEXT CALL LINK
       SWPB @>83C2       SAVE THE CURRENT INTERRUPT FLAGS
      MOVB @NOQUIT,@>83C2    ENSURE THAT INTERRUPT CHECKS FOR ALL BUT QUIT KEY
      MOV  @HOOKAD,@>83C4    SET UP THE INTERRUPT SEQUENCE TO READ OTHER LINES
ABORT1
       BL   @DOFILE      CLOSE THE FILE AND STOP THE INTERRUPTS
       DATA CLOSE        CLOSE DOWN THIS FILE
       LWPI >83E0        RESTORE GPL WORKSPACE
GPL    EQU  $+2
       LI   R0,0         ACTUAL GPL ADDRESS WILL BE STORED HERE
       MOVB R0,@>402(R13)     RESTORE HIGH BYTE OF GPL ADDRESS
       SWPB R0           DELAY
       MOVB R0,@>402(R13)     RESTORE LOW BYTE OF GPL ADDRESS
BATCRT EQU  $+2
       B    @0           ACTUAL RETURN ADDRESS WILL BE SAVED HERE
********************************************************************************
TOSCRN
       SB   @>8360,@>8360     FLAG TO INDICATE LINE HAS BEEN CHANGED
       CB   @>8301,@H7E  IS THE CURSOR ON THE SCREEN?
       JEQ  SCROLL       NO, SKIP THIS PART
       MOV  @>8320,R0    SCREEN START ADDRESS (SHOULD BE >2E2)
       MOVB @R0LB,*VDPWA LOW BYTE OF ADDRESS OF CURSOR ON SCREEN
       ORI  R0,>4000     SET THE WRITE BIT
       MOVB R0,*VDPWA    HIGH BYTE OF ADDRESS WHERE CURSOR SITS
       MOVB @H7E,@>8301  MOVE CURSOR INTO HOLDING AREA
       MOVB @SPACE,*VDPWD     WRITE SPACE TO SCREEN WHERE CURSOR WAS
SCROLL
       MOV  *NEXTAD+,NEXTLN MOVE ADDRESS INTO REGISTER ( 0 ADDRESS WHEN DONE )
       MOVB @ENTER,@>8375 "        "        "      "    ENTER HAS BEEN PRESSED
       MOVB *NEXTLN+,R12   SIZE OF TEXT TO BE MOVED
```

```
       MOV  NEXTLN,R0    COPY POSITION FOR USE DURING CHECK OF OLD AND RUN
      LI   R1,RUNOLD    CPU ADDRESS OF OLD AND RUN TEXT (OFFSET ADDED BY SETUP)
OLDRUN
       CB   *R0+,*R1+    CHECK THE FIRST BYTE
       JEQ  CHECK1       THE 'R' MATCHES, SO KEEP CHECKING FOR 'RUN'
       DEC  R0           BACK UP TO THE FIRST CHARACTER AGAIN
       INCT R1           MOVE PAST 'UN' AND POINT TO 'O' OF 'OLD'
       CB   *R0+,*R1+    CHECK FOR 'O'
       JNE  NOTRUN       NO MATCH, SKIP THE REST OF THE CHECK
CHECK1
       CB   *R0+,*R1+    CHECK SECOND LETTER (EITHER 'U' OR 'L')
       JNE  NOTRUN       DOESN'T MATCH - SKIP THE REST
       CB   *R0+,*R1+    CHECK THIRD LETTER (EITHER 'N' OR 'D')
       JNE  NOTRUN       DOESN'T MATCH SO SKIP NEXT LINE
       CLR  @LINCNT      MAKE SURE THIS IS THE LAST LINE LOADED
NOTRUN
       SRA  R12,8        MAKE IT A WORD SIZE INSTEAD OF BYTE SIZE
       INC  R12          WILL WRITE OVER THE CURSOR, SO SAVE A SPACE
       MOV  R12,R1       NEED TO WORK WITH COUNT BUT ALSO NEED ORIGINAL LATER
       CLR  R0           TO PREVENT POSSIBLE OVERFLOW ERROR
       DIV  @D28,R0      DETERMINE NUMBER OF LINES OF TEXT
       A    R1,@>832A    POINTS AFTER THE END OF THE LINE
       SLA  R0,5         NUMBER OF LINES * 32 = NUMBER OF BYTES TO SCROLL UP
       MOV  R0,R6        COPY FOR USE LATER
       JEQ  SCROL3       NONE TO SCROLL, SO START THE WRITE ROUTINE
       S    R0,@>8320    NEW LINE STARTING POSITION
       LI   R7,32        NUMBER OF BYTES PER LINE
SCROL0
       MOVB @R0LB,*VDPWA NUMBER OF BYTES ALSO HAPPENS TO BE STARTING ADDRESS
       LI   R1,SCRBUF    CPU BUFFER TO HOLD DATA
       MOVB R0,*VDPWA    HIGH BYTE - WRITE BIT NOT SET (READING FIRST)
       MOV  R7,R2        R2 USED TO COUNT DOWN EACH BYTE
SCROL1
       MOVB *VDPRD,*R1+  READ BYTE INTO BUFFER
       DEC  R2           COUNT OFF 32 OF THEM
       JGT  SCROL1       AND LOOP UNTIL WHOLE LINE IS READ
       S    R6,R0        BACK UP THE APPROPRIATE NUMBER OF BYTES
       ORI  R0,>4000     SET THE WRITE BIT OF THE VDP ADDRESS
       MOVB @R0LB,*VDPWA WRITE THE LOW BYTE OF THE NEW ADDRESS
      LI   R1,SCRBUF    RESET R1 TO THE SCROLL BUFFER WHERE OLD LINE IS STORED
       MOVB R0,*VDPWA    WRITE HIGH BYTE (WITH WRITE BIT SET)
       ANDI R0,>3FFF     RESTORE TO REGULAR VDP ADDRESS WITH WRITE BIT RESET
       MOV  R7,R2        MAKE A COPY OF 32 IN THE BYTE COUNTER (R2)
SCROL2
       MOVB *R1+,*VDPWD  WRITE EACH BYTE
       DEC  R2           32 IN ALL
       JGT  SCROL2       LOOP UNTIL DONE
       A    R6,R0        CHANGE ADDRESS TO ORIGINAL LINE (JUST MOVED THIS)
       A    R7,R0        GO ON TO THE NEXT LINE
       CI   R0,>300      ARE WE OFF THE SCREEN?
```

```
       JLT  SCROL0       NO, GO BACK AND MOVE THIS ONE TOO
SCROL3
       MOV  @>8320,R0    WILL START WRITING OUR TEXT AT THIS POINT
       MOVB @R0LB,*VDPWA WRITE THE LOW BYTE OF THE VDP ADDRESS
       ORI  R0,>4000     SET WRITE BIT FOR THIS ADDRESS
       MOVB R0,*VDPWA    WRITE THE HIGH BYTE FOR TEXT ADDRESS
       ANDI R0,>3FFF     RESTORE TO PROPER ADDRESS
SCROL4
       LI   R6,28        28 CHARACTERS PER LINE OF TEXT
SCROL5
       DEC  R12          COUNT OFF EACH BYTE BEFORE IT'S LOADED
       JGT  SCROL6       STILL SOME TEXT LEFT, DON'T STOP YET
       MOVB @SPACE,*VDPWD     WRITE A SPACE CHARACTER TO THE SCREEN
       JMP  SCROL7       COUNT OFF EACH SPACE
SCROL6
       MOVB *NEXTLN+,*VDPWD   WRITE A BYTE OF TEXT TO THE SCREEN
SCROL7
       INC  R0           KEEP TRACK OF WHERE WE ARE ON THE SCREEN
       DEC  R6           STILL SPACE LEFT ON THE LINE?
       JGT  SCROL5       YES, CONTINUE TO WRITE CHARACTERS TO LINE
       LI   R6,4         FOUR EDGE CHARACTERS BETWEEN EACH LINE
SCROL8
       CI   R0,>2FD      PAST THE LAST CHARACTER OF LAST ROW?
       JGT  OUT          YES, WE'RE DONE WITH THIS LINE, SO GET OUT
SCROL9
       MOVB @EDGE,*VDPWD WRITE THE EDGE CHARACTERS TO THE SCREEN
       INC  R0           KEEP TRACK OF THE ADDRESS
       DEC  R6           DO FOUR OF THEM
       JGT  SCROL8       LOOP FOR FOUR
       JMP  SCROL4       THEN GO BACK FOR THE NEXT LINE
OUT
       DEC  @LINCNT      COUNT OFF EACH LINE AS IT IS LOADED
       JGT  OUT1         STILL SOME LINES LEFT?
       CLR  @>83C4       CLEAR INTERRUPTS
       SWPB @>83C2       RESTORE INTERRUPT FLAGS
OUT1
       RT
*****************************************************************************
* DSRLNK: FROM BY MILLERS GRAPHICS - MANY THANKS!!!
*
*****************************************************************************
PUTSTK EQU  >50           PUSH GROM ADD TO STACK POINTER
TYPE   EQU  >836D         DSRLNK TYPE BYTE FOR GPL DSRLNK
NAMLEN EQU  >8356         DEVICE NAME LENGTH POINTER IN VDP PAB
VWA    EQU  >8C02         VDP WRITE ADDRESS LOCAION
VRD    EQU  >8800         VDP READ DATA BYTE LOCATION
GR4LB  EQU  >83E9         GPL WORKSPACE R4 LOWER BYTE
GSTAT  EQU  >837C         GPL STATUS BYTE LOCATION

DSRLNK DATA DSRWS,DLINK1 SET BLWP VECTORS
```

```
DSRWS  EQU  $            START OF DSRLNK WORKSPACE
DR3LB  EQU  $+7          R3 LOWER BYTE OF DSRLNK WORKSPACE
DLINK1
       MOV  R12,R12      R0 HAVE WE ALREADY LOOKED UP THE LINK ADDRESS?
       JNE  DLINK3       R1 YES! SKIP THE LOOK UP ROUTINE
       LWPI GPLWS        R2,R3 ELSE LOAD GPL WORKSPACE
       MOV  @PUTSTK,R4   R4,R5 RESTORE CURRENT GROM ADDRESS ON THE STACK
       BL   *R4          R6
       LI   R4,>11       R7,R8 LOAD R4 WITH ADDRESS OF LINK ROUTINE VECTOR
       MOVB R4,@>402(R13)     R9,R10 SET UP GROM WITH ADDRESS FOR VECTOR
       JMP  DLINK2       R11 JUMP AROUND R12-R15
       DATA 0            R12 CONTAINS >2000 FLAG WHEN SET
       DATA 0,0,0        R13-R15 CONTAINS WS, PC, & ST FOR RTWP
DLINK2
       MOVB @GR4LB,@>402(R13) FINISH SETTING UP GROM ADDRESS
       MOV  @GETSTK,R5   TAKE SOME TIME & SET UP GETSTK POINTER
       MOVB *R13,@DSRAD1 GET THE GPL DSR LINK VECTOR
       INCT @DSRADD      ADJUST IT TO GET PADT GPL FETCH INSTRUCTION
       BL   *R5          RESTORE THE GROM ADDRESS OFF THE STACK
       LWPI DSRWS        RELOAD DSRLNK WORKSPACE
       LI   R12,>2000    SET FLAG TO SIGNIFY DSRLNK ADDRESS IS SET
DLINK3
       INC  R14          ADJUST R14 TO POINT TO CALLER'S DSR TYPE BYTE
       MOVB *R14+,@TYPE  MOVE IT INTO >836D FOR GPL DSRLNK
       MOV  @NAMLEN,R3   SAVE VDP ADDRESS OF NAME LENGTH
       AI   R3,-8        ADJUST IT TO POINT TO PAB FLAG BYTE
       BLWP @GPLLNK      EXECUTE DSR LINK
DSRADD BYTE >03          HIGH BYTE OF GPL DSRLNK ADDRESS
DSRAD1 BYTE >00          LOWER BYTE OF GPL DSRLNK ADDRESS
****** ERROR CHECK
       MOVB @DR3LB,@VWA  SET UP LSB OF VDP ADD FOR ERROR FLAG
       MOVB R3,@VWA      SET MSB OF VDP ADD FOR ERROR FLAG
       SZCB R12,R15      VLEAR EQ BIT FOR ERROR REPORT
       MOVB @VRD,R3      GET PAB ERROR FLAG
       SRL  R3,5         ADJUST IT TO 0-7 ERROR CODE
       MOVB R3,*R13      PUT IT INTO CALLER'S R0 (MSB)
       JNE  SETEQ        IF IT NOT ZERO, SET EQ BIT
       COC  @GSTAT,R12   ELSE TEST CND BIT FOR DEVICE ERROR (00 IN R0)
       JNE  DSREND       NO ERROR, JUST RETURN
SETEQ
       SOCB R12,R15      ERROR SO SET CALLER'S EQ BIT
DSREND
       RTWP              ALL DONE - RETURN TO CALLER
********************************************************************************
* GPLLNK: ALSO FROM MILLERS GRAPHICS - THANKS AGAIN!!!!
*
********************************************************************************
GPLWS  EQU  >83E0         GPL WORKSPACE
GR4    EQU  GPLWS+8       GPL WORKSPACE R4
GR6    EQU  GPLWS+12      GPL WORKSPACER6
```

```
STKPNT EQU  >8373        GPL STACK POINTER
LDGADD EQU  >60          LOAD AND EXECUTE GROM ADDRESS ENTRY POINT
XTAB27 EQU  >200E        LOW MEMORY XML TABLE LOCATION 27
GETSTK EQU  >166C

GPLLNK DATA GLNKWS       R7 SET UP BLWP VECTORS
       DATA GLINK1       R8

RTNAD  DATA XMLRTN       R9 ADDRESS WHERE GPL XML RETURNS TO US
GXMLAD DATA >176C        R10 GROM ADDRESS FOR GPL XML (0F 27 OPCODE)
       DATA >50          R11 INITIALIZED TO >50 WHERE PUTSTK ADDRESS RESIDES

GLNKWS EQU  $->18        GPLLNK'S WORKSPACE OF WHICH ONLY
       BSS  >08          REGISTERS R7 THROUGH R15 ARE USED
GLINK1
       MOV  *R11,@GR4    PUT PUTSTK ADDRESS INTO R4 OF GPL WS
       MOV  *R14+,@GR6   PUT GPL ROUTINE ADDRESS IN R6 OF GPL WS
       MOV  @XTAB27,R12  SAVE THE VALUE AT >200E
       MOV  R9,@XTAB27   PUT XMLRTN ADDRESS INTO >200E
       LWPI GPLWS        LOAD GPL WS
       BL   *R4          SAVE THE CURRENT GROM ADDRESS ON STACK
       MOV  @GXMLAD,@>8302(R4) PUSH GPL XML ADDRESS ON STACK FOR GPL RETURN
       INCT @STKPNT      ADJUST THE STACK POINTER
       B    @LDGADD      EXECUTE OUR GPL ROUTINE
XMLRTN
       MOV  @GETSTK,R4   GET GETSTK POINTER
       BL   *R4          RESTORE GROM ADDRESS OFF THE STACK
       LWPI GLNKWS       LOAD OUR WORKSPACE
       MOV  R12,@XTAB27  RESTORE >200E
       RTWP              ALL DONE - RETURN TO CALLER
SCRBUF BSS  32           USED TO HOLD LINES SCROLLED UP SCREEN
LINBUF BSS  300          EXTRA SPACE JUST IN CASE
FIRSTL
       TEXT 'REM   PARAGON COMPUTING     '
       TEXT '     17 CONSTANCE STREET    '
       TEXT '     MERRIMACK, NH 03054'
       BYTE 0,>14        OPEN A DISPLAY VARIABLE 80 FILE AS INPUT (SEQUENTIAL)
BUFFAD DATA 0            THE BUFFER ADDRESS IN VDP GOES HERE
NOQUIT
B80    BYTE 80,0,0,0,0   MAXIMUM OF 80 CHARACTERS READ PER RECORD
FILNAM BSS  28           SIZE MAXIMUM IS 25 (PLUS ONE FOR LENGTH) PLUS SPARE
       EVEN
       COPY "DSK2.TEXTLOADS2"
```

Disk 82. Contents of file TEXTLOADS2

```
*****************************************************************************
* DOFILE: PERFORMS THE OPERATION REQUESTED ON FILE BASED ON POINTR
*
*         USES THE WORD AFTER THE BL AS THE OPERATION TO BE PERFORMED
*
*         IF AN ERROR OCCURS, ERROR MESSAGE IS DISPLAYED - RETURNS TO XBASIC
*
*         IF AT END OF FILE, MORFIL WILL BE SET TO >0100 OTHERWISE >0000
*
*****************************************************************************
DOFILE
       MOV  @POINTR,R2   GET POINTER TO FILE NAME LENGTH BYTE LOCATION
       MOV  R2,@>8356    SET POINTER FOR DSR LINK
       AI   R2,-9+>4000  OPCODE ADDRESS, PLUS WRITE BIT SET
       MOVB @R2LB,*VDPWA WRITE LOW BYTE OF ADDRESS
       NOP               STANDARD DELAY WHEN ACCESSING MEMORY MAPPED I/O
       MOVB R2,*VDPWA    WRITE HIGH BYTE OF ADDRESS
       MOVB *R11+,R1     OPCODE IN HIGH BYTE
       MOVB R1,*VDPWD    WRITE OPCODE TO THE BUFFER IN USE
       BLWP @DSRLNK      MODIFIED FORM OF MILLERS GRAPHICS DSRLNK ROUTINE
       DATA 8            LINKING TO A DEVICE ROUTINE, NOT A SUBPROGRAM
       JNE  FILEOK       OPERATION WAS SUCCESSFUL - CONTINUE
       CB   R1,@B01      ARE WE TRYING TO PERFORM A CLOSE OPERATION?
       JEQ  FILEOK       YES, DON'T WORRY ABOUT IT
       CB   R1,@B09      ARE WE TRYING TO PERFORM A STATUS OPERATION?
       JEQ  FILEOK       YES, DON'T WORRY ABOUT IT
       MOVB R0,R2        MUST HAVE BEEN AN OPEN OR READ - SAVE THE ERROR TYPE
       SRA  R2,7         MAKE THE ERROR BYTE A WORD*2
PRINT
       MOV  @ERRMSG(R2),R7    GET ADDRESS OF ERROR MESSAGE
      LI   R0,>0040     ADDRESS OF SCREEN START, BYTES REVERSED (WRITE BIT SET)
       LI   R2,>300      >300 SPACES TO BE WRITTEN TO THE SCREEN
       MOVB R0,*VDPWA    LOAD THE LOWER BYTE IF THE ADDRESS
       SWPB R0           SWAP BYTES AND DELAY
       MOVB R0,*VDPWA    LOAD THE UPPER BYTE OF THE SCREEN ADDRESS
       SLA  R0,1         MAKE THE >40 INTO AN >80 - SPACE WITH SCREEN OFFSET
CLEARS
       MOVB R0,*VDPWD    MOVE SPACE TO SCREEN
       DEC  R2           COUNT OF >300 OF THEM
       JGT  CLEARS       AND LOOP UNTIL THE ENTIRE SCREEN IS CLEARED
ERRSHO
       LI   R0,>8241     ROW 12, COLUMN 2, REVERESED - WITH WRITE BIT SET
       MOVB R0,*VDPWA    SET LOWER BYTE OF THIS ADDRESS
       SWPB R0           SWAP AND DELAY
       MOVB R0,*VDPWA    SET HIGH BYTE
       MOVB *R7+,R0      GET LENGTH BYTE OF ERROR MESSAGE
       SRA  R0,8         MAKE THE BYTE INTO A WORD
ERRS1
```

```
       MOVB *R7+,*VDPWD  MOVE EACH CHARACTER TO THE SCREEN  (ALREADY OFFSET)
       DEC  R0           COUNT OFF THE LENGTH
       JGT  ERRS1        LOOP UNTIL DONE
       BLWP @GPLLNK      LINK TO GPL ROUTINE
       DATA >36          HONK
       B    @ABORT1      ABORT THE PROCESS BY TRYING TO CLOSE THE FILE AND QUIT
FILEOK
       CB   R1,@STATUS   HAVE WE CHECKED THE STATUS YET?
       JEQ  FILERT       YES, PREPARE TO RETURN
       AI   R2,5->4000   ADDRESS OF CHARACTER COUNT, WRITE BIT REMOVED
       MOVB @R2LB,*VDPWA LOWER BYTE OF ADDRESS OF BYTE COUNT IN PAB
       NOP               STANDARD DELAY
       MOVB R2,*VDPWA    HIGH BYTE OF ADDRESS OF BYTE COUNT IN PAB
       NOP               ANOTHER DELAY BEFORE THE READ
       MOVB *VDPRD,@COUNT     GET THE NUMBER OF BYTES READ - MOVE TO COUNT BYTE
       JMP  DOFILE       NOW CHECK THE STATUS - END OF FILE CHECK
FILERT
       AI   R2,8->4000   ADDRESS OF STATUS BYTE IN PAB, WRITE BIT REMOVED
       MOVB @R2LB,*VDPWA LOWER BYTE OF ADDRESS OF STATUS BYTE
       NOP               STANDARD DELAY
       MOVB R2,*VDPWA    HIGH BYTE OF ADDRESS OF STATUS BYTE
       NOP               ANOTHER DELAY
       MOVB *VDPRD,@MORFIL    READ STATUS BYTE INTO MORFIL
       SZCB @HFE,@MORFIL IF WE'RE AT THE EOF, THEN >FE SZCB MORFIL = >0100
       RT                WERE EVER WE CAME FROM
OPEN   EQU  >0009        OPEN OPCODE WITH STATUS OPCODE ON THE END
CLOSE  EQU  >0109        CLOSE OPCODE WITH STATUS OPCODE ON THE END
READ   EQU  >0209        READ OPCODE WITH STATUS OPCODE ON THE END
*                        TABLE OF ADDRESSES FOR EACH ERROR MESSAGE
ERRMSG DATA BADNAM       >00 - BAD DEVICE NAME
       DATA BADNAM       >01 - WRITE PROTECTED: WHO CARES?
       DATA BADOPN       >02 - BAD OPEN ATTRIBUTES: FOR RS232 AND PIO
       DATA BADOPR       >03 - ILLEGAL OPERATION, PROBABLY CAN'T READ
       DATA BADNAM       >04 - BUFFER PROBLEM: SHOULDN'T GET THIS ONE
       DATA BADNAM      >05 - END OF FILE ERROR: SHOULDN'T GET THIS ONE EITHER
       DATA BADDVC       >06 - DEVICE ERROR
       DATA BADFIL       >07 - FILE MISMATCH OR NON-EXISTENT
RUNOLD TEXT 'RUNOLD'     TEXT USED TO CHECK THE FIRST THREE CHARACTERS IN LINE;
BADNAM BYTE 15->60       >60 WILL BE ADDED TO EVERY BYTE, SO NEED TO OFFSET
       TEXT 'BAD DEVICE NAME' EACH MESSAGE IS PRECEEDED BY ITS LENGTH
BADOPN BYTE 19->60
       TEXT 'INCORRECT FILE TYPE'
BADOPR BYTE 27->60
       TEXT 'CAN''T READ FROM THIS DEVICE'
BADDVC BYTE 21->60
       TEXT 'DEVICE OR DISK IS BAD'
BADFIL BYTE 21->60
       TEXT 'NO FILE OR WRONG TYPE'
HELPTX TEXT '  CALL LINK("BATCH",dsk#.name)  '   HELP SCREEN DISPLAYED WITH
       TEXT '       Loads a text file into   '   CALL LINK("HELP")
       TEXT '       the command line one     '
```

```
       TEXT '       line at a time.          '
       TEXT '  CALL LINK("OLD",dsk#.name)    '
       TEXT '       Creates programs from    '
       TEXT '       a listing.  Leading      '
       TEXT '       spaces and any lines     '
       TEXT '       whose first non-space    '
       TEXT '       character is the remark  '
       TEXT '       sign (!) are ignored.    '
       TEXT '       Multiple lines will be   '
       TEXT '       loaded.  Valid lines     '
       TEXT '       start with a number set  '
       TEXT '       off by a space.          '
       TEXT '  CALL LINK("MERGE",dsk#.name)  '
       TEXT '       Same as OLD, except any  '
       TEXT '       program already loaded   '
       TEXT '       is not erased.           '
       TEXT '  CALL LINK("CONT")             '
       TEXT '       Continues loading after  '
       TEXT '       a BREAK with CLEAR key.'
ENDTXT EVEN
DONE   DATA 0            LINE ADDRESS LIST STARTS HERE IN LOW MEMORY!
*******************************************************************************
* INITIAL HOOK ROUTINE WHICH SETS UP DEF TABLE IN EXTENDED BASIC.
*
* >FF00 IS 255,0.  CALL LINK(-31804,255) WILL SET USER HOOK TO >FF00
*
* THIS ROUTINE MOVES THE MACHINE CODE INTO LOW MEMORY AND SETS UP THE DEF'S
*
* THE LOW MEMORY POINTERS ARE ADJUSTED TO SHOW NO MORE ROOM.  THIS ROUTINE
*
* 'FREE SPACE' IN LOW MEMORY TO HOLD TEXT ADDRESSES
*
*******************************************************************************
       AORG >FF00
HOOK1
       CLR  @>83C4       ONLY NEED TO HOOK HERE ONCE
       LI   R0,>24F4     START OF ROUTINE IN LOW MEMORY
HIGHAD EQU  $+2
       LI   R1,0         ACTUAL ADDRESS IN HIGH MEMORY GOES HERE: SEE SETUP
COUNT1 EQU  $+2
       LI   R2,0         ACTUAL COUNT (SIZE) OF ROUTINE GOES HERE: SEE SETUP
LOOP1
       MOV  *R1+,*R0+    MOVE HIGH MEMORY BACK INTO LOW MEMORY LOCATION
       DECT R2           COUNT OFF WHATEVER THE SIZE IS
       JGT  LOOP1        LOOP UNTIL DONE
DEFTBL EQU  $+2
       LI   R0,0         START OF DEF TABLE GOES HERE: SEE SETUP
       MOV  R0,@>2002    SET START OF FREE SPACE AT END OF FREE SPACE
       MOV  R0,@>2004    POINTS TO START OF DEF TABLE IN LOW MEMORY
LOOP2
```

```
       MOV  *R1+,*R0+    MOVE DEF'S BACK INTO TABLE
       CI   R0,>4000     DON'T WORRY ABOUT CHECKING THE SIZE - QUIT AT >4000
       JLT  LOOP2        LOOP UNTIL DONE - TWO BYTES AT A TIME
       RT                ALL DONE! OK TO RETURN TO INTERPRETER
********************************************************************************
* SETUP: WHEN THE OBJECT CODE IS LOADED INTO EXTENDED BASIC, SETUP IS USED TO
*
* ADD THE SCREEN OFFSET TO THE TEXT AND MOVE THE CODE INTO HIGH MEMORY
*
* ADDRESSES AND LENGTH OF CODE ARE SAVED IN THE HOOK1 ROUTINE FOR RESTORING
*
* THE START AND END LINE POINTERS (>8330 AND >8332) ARE ADJUSTED TO ALLOW
*
* SAVING THE CODE WITH: 1 CALL INIT::CALL LINK(-31804,255)
*
********************************************************************************
       DEF  SETUP
SETUP
       LI   R0,RUNOLD    THIS IS THE START OF TEXT TO WHICH >60 IS ADDED
       LI   R1,>6060     SCREEN OFFSET IN EXTENDED BASIC
SETLUP
       A    R1,*R0+      ADD THE OFFSET TO EACH BYTE OF TEXT
       CI   R0,DONE      ARE WE AT THE END OF THE TEXT?
       JLT  SETLUP       NO, LOOP BACK UNTIL ALL TEXT IS OFFSET
       MOV  @>2004,R0    GET THE START OF THE DEF TABLE
       AI   R0,8         DON'T SAVE 'SETUP' WITH THE OTHER LINKS
       MOV  R0,@DEFTBL   SAVE THE DEF TABLE ADDRESS IN THE HOOK1 ROUTINE
       MOV  R0,R1        COPY DEF TABLE ADDRESS - USE TO CALCULATE SIZE
       AI   R1,>FF00->4000    ENDS UP COUNTING BACK FROM HOOK1 - MAKE SPACE
       MOV  R1,R3        SAVE THIS FOR USE LATER - START OF DEF IN HIGH MEMORY
LOOP3
       MOV  *R0+,*R3+    MOVE LINK DATA INTO HIGH MEMORY FOR STORAGE
       CI   R0,>4000     REACHED THE END OF THE DEF TABLE?
       JLT  LOOP3        NO, KEEP MOVING DATA UNTIL ALL LINKS ARE IN HIGH MEM
       LI   R0,>24F4     START OF MACHINE CODE ROUTINES IN LOW MEMORY
       MOV  @>2002,R2    END OF MACHINE CODE ROUTINES IN LOW MEMORY
       S    R0,R2        GIVES SIZE OF MACHINE CODE ROUTINES
       MOV  R2,@COUNT1   STORE SIZE IN COUNT DATA OF HOOK1
       S    R2,R1        BACK OFF REQUIRED SPACE FROM PREVIOUS DATA
       MOV  R1,@HIGHAD   STORE HIGH ADDRESS IN HOOK1
       DEC  R1           BACK OFF ONE MORE SPACE - LAST FREE SPACE IN HIGH MEM
       MOV  R1,@>8330    STORE THIS IN END OF LINE NUMBER TABLE POINTER
       MOV  R1,@>8332    STORE THIS IN START OF LINE NUMBER TABLE POINTER
       INC  R1          RESTORE R1 TO START OF REQUIRED SPACE FOR MACHINE CODE
LOOP4
       MOV  *R0+,*R1+    MOVE MACHINE CODE INTO HIGH MEMORY
       DECT R2           COUNT OFF TWO BYTES AT A TIME
       JGT  LOOP4        AND LOOP UNTIL DONE
       RT
       END
```

Disk 83. Sort and Word Count

Version:
Requires: EA
Author: J. Peter Hoddie
Language: AL
Updated: 07/15/87

Sort is one of the most capable and fast sorts written for the 4A. Word Count is a really quick way to count the words and lines in a TI-Writer file. Both programs are fairware.

```
dskdir. v2.0. 12-dec-96
Disk name               = SORT/COUNT
Sectors total           = 360
Sectors used            = 286
Sectors available       = 72
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       6 DIS/VAR 80   >022 005
002 >003 CIF/S         4 DIS/VAR 80   >027 003
003 >004 DSRLNK/S     21 DIS/VAR 80   >02a 020
004 >005 INPUT/S      12 DIS/VAR 80   >03e 011
005 >006 MG/S         23 DIS/VAR 80   >049 022
006 >007 SAVE/S        4 DIS/VAR 80   >05f 003
007 >008 SORT/DOC     52 DIS/VAR 80   >062 051
008 >009 SORT/EXP     12 PROGRAM      >095 011
009 >00a SORT/EXP/S   96 DIS/VAR 80   >0a0 095
010 >00b UTILS/S       6 DIS/VAR 80   >0ff 005
011 >00c VDP/S         7 DIS/VAR 80   >104 006
012 >00d WC            9 PROGRAM      >10a 008
013 >00e WC/S         34 DIS/VAR 80   >112 033
```

Disk 83. Contents of file -README

Boston Computer Society
TI-99/4A User Group
Public Domain Software Library

SORT and WORD COUNT
Fairware by J. Peter Hoddie

A few notes

Word Count was written prior to the writing of GRAM Packer as part of an experiment on my part to create a 100% assembly program. Prior to Word Count all of my programs were either in Extended BASIC or Extended BASIC with assembly language subroutines, with the exception of a couple games. Since Word Count worked, I went on and wrote GRAM Packer (and a few other things since . . .). Word Count is pretty simple and quite fast. Run it as an EA5 program. The filename is WC. The source code is included in the files CIF/S, MG/S, VDP/S, and WC/S.

The remaining files on the disk make up the Sort program and its source files. Complete docs on this program can be found in the file SORT/DOC.

Please note that SORT and WORD COUNT are considered fairware programs and if you use them a fairware payment is expected. For my complete feelings on fairware read the notes in SORT/DOC.

J. Peter Hoddie
12 Paul Revere Road
Lexington, MA 02173

Disk 83. Contents of file CIF/S

```
CIF    DATA CIFWS,CIF0

CIFWS  BSS  32

C100   DATA 100

CIF0   LWPI >83E0
       LI   R4,>834A
       MOV  *R4,R0
       MOV  R4,R6
       CLR  *R6+
       CLR  *R6+
       MOV  R0,R5
       JEQ  CIFA
       ABS  R0
       LI   R3,>0040
       CLR  *R6+
       CLR  *R6
       CI   R0,>0064
       JL   CIFB
       CI   R0,>2710
       JL   CIFC
       INC  R3
       MOV  R0,R1
       CLR  R0
       DIV  @C100,R0
       MOVB @>83E3,@>0003(R4)
CIFC   INC  R3
       MOV  R0,R1
       CLR  R0
       DIV  @C100,R0
       MOVB @>83E3,@>0002(R4)
CIFB   MOVB @>83E1,@>0001(R4)
       MOVB @>83E7,*R4
       INV  R5
       JLT  CIFA
       NEG  *R4
CIFA   LWPI CIFWS
       RTWP
```

Disk 83. Contents of file DSRLNK/S

```
* DSRLNK/S FROM J. PETER HODDIE

SCNAME EQU  >8356
SADDR  EQU  >83D2
SCLEN  EQU  >8355
GPLWS  EQU  >83E0
ERRCOD EQU  >8322

*H20    TEXT ' '
HAA    BYTE >AA
DECMAL TEXT '.'
       EVEN
DSRLNK DATA DLNKWS,DLENTR
*
DLENTR MOV  *R14+,R5          FETCH PROGRAM TYPE FOR LINK

       MOV  @SCNAME,R0        FETCH POINTER INTO PAB

       MOV  R0,R9             SAVE POINTER
       AI   R9,-8             ADJUST POINTER TO FLAG BYTE

       BLWP @VSBR           READ DEVICE NAME LENGTH

       MOVB R1,R3             STORE IT ELSEWHERE
       SRL  R3,8              MAKE IT A WORD VALUE
       SETO R4                INITIALIZE A COUNTER
       LI   R2,NAMBUF         POINT TO NAMBUF

LNK$LP INC  R0                POINT TO NEXT CHAR OF NAME
       INC  R4                INCREMENT CHARACTER COUNTER
       C    R4,R3             END OF NAME?
       JEQ  LNK$LN            YES
       BLWP @VSBR           READ CURRENT CHARACTER
       MOVB R1,*R2+           MOVE TO NAMBUF
       CB   R1,@DECMAL        IS IT A DECIMAL POINT?
       JNE  LNK$LP            NO
LNK$LN MOV  R4,R4             IS NAME LENGTH ZERO?
       JEQ  LNKERR            YES, ERROR
       CLR  @CRULST
       MOV  R4,@SCLEN-1       STORE NAME LENGTH FOR SEARCH
       MOV  R4,@SAVLEN        SAVE DEVICE NAME LENGTH
       INC  R4                ADJUST IT
       A    R4,@SCNAME        POINT TO POSITION AFTER NAME
       MOV  @SCNAME,@SAVPAB   SAVE POINTER INTO DEVICE NAME

*
*** SEARCH ROM CROM GROM FOR DSR
*
```

```
SROM    LWPI GPLWS              USE GPL WORKSPACE TO SEARCH
        CLR  R1                 VERSION FOUND OF DSR ETC
        LI   R12,>0F00          START OVER AGAIN
NOROM   MOV  R12,R12            ANYTHING TO TURN OFF
        JEQ  NOOFF              NO
        SBZ  0
NOOFF   AI   R12,>0100          NEXT ROM'S TURN ON
        CLR  @CRULST            CLEAR IN CASE WE'RE FINISHED
        CI   R12,>2000          AT THE END
        JEQ  NODSR              NO MORE ROM'S TO TURN OFF
        MOV  R12,@CRULST        SAVE ADDRESS OF NEXT CRU
        SBO  0                  TURN ON ROM
        LI   R2,>4000           START AT BEGINNING
        CB   *R2,@HAA           IS IT A VALID ROM?
        JNE  NOROM              NO
        A    @TYPE,R2           GO TO FIRST POINTER
        JMP  SGO2

SGO     MOV  @SADDR,R2          CONTINUE WHERE WE LEFT OFF
        SBO  0                  TURN ROM BACK ON
SGO2    MOV  *R2,R2             IS ADDRESS A ZERO
        JEQ  NOROM              YES, NO PROGRAM TO LOOK AT
        MOV  R2,@SADDR          REMEMBER WHERE WE GO NEXT
        INCT R2                 GO TO ENTRY POINT
        MOV  *R2+,R9            GET ENTRY ADDRESS
*
*** SEE IF NAME MATCHES
*
        MOVB @SCLEN,R5          GET LENGTH AS COUNTER
        JEQ  NAME2              ZERO LENGTH, DON'T DO MATCH
        CB   R5,*R2+            DOES LENGTH MATCH?
        JNE  SGO                NO
        SRL  R5,8               MOVE TO RIGHT PLACE
        LI   R6,NAMBUF          POINT TO NAMBUF
NAME1   CB   *R6+,*R2+
        JNE  SGO                NO MATCH
        DEC  R5
        JNE  NAME1              YES
NAME2   INC  R1                 NEXT VERSION FOUND
        MOV  R1,@SAVVER         SAVE VERSION NUMBER
        MOV  R9,@SAVENT         SAVE ENTRY ADDRESS
        MOV  R12,@SAVCRU        SAVE CRU ADDRESS
DSRENT  BL   *R9                MATCH, CALL SUBROUTINE
        JMP  SGO                NOT RIGHT VERSION
        SBZ  0                  TURN OFF ROM
        LWPI DLNKWS             SELECT DSRLNK WORKSPACE
        MOV  R9,R0              POINT TO FLAG BYTE IN PAB
        BLWP @VSBR             READ BYTE FLAG
        SRL  R1,13              JUST WANT THE ERROR FLAGS
        JNE  IOERR1
```

```
       SZCB @H20,R15          NO ERROR (RESET EQU BIT)
       RTWP
*
*** ERROR HANDLING
*
NODSR  LWPI DLNKWS            SELECT DSRLNK WORKSPACE
LNKERR CLR  R1                CLEAR THE ERROR FLAGS
IOERR1 SWPB R1
       MOVB R1,*R13           STORE ERROR IN CALLING R13
       SOCB @H20,R15          INDICATE AN ERROR OCCURRED
       RTWP                   BACK TO CALLER
       PAGE
SVGPRT DATA 0                 RETURN FROM ASSM LANG
SAVCRU DATA 0                 SAVE CRU ADDRESS WORD OR BYTE
SAVENT DATA 0                 SAVE ENTRY ADDRESS INTO DSR WORD
SAVLEN DATA 0                 DEVICE NAME LENGTH BYTE
SAVPAB DATA 0                 POINTER TO POSITION AFTER NAME  CALCULATE
SAVVER DATA 0                 SAVE WHICH DEVICE 1,2,3 OF SAME NAME BYTE
ENTADD DATA 0
FLGPTR DATA 0
CHKSAV DATA 0
NAMBUF DATA 0,0,0,0
DLNKWS DATA 0,0,0,0,0
TYPE   DATA 0,0,0,0,0,0,0,0,0,0,0
CRULST BSS  2
```

Disk 83. Contents of file INPUT/S

```
* INPUT/S FROM J. PETER HODDIE

INPUT  DATA INPWS,INPUT0
INPUTS DATA INPWS,INPUT9
INPWS  BSS  32
CR     BYTE 13           A CARRIAGE RETURN
BACKSP BYTE 08+>80       BACK SPACE CHARACTER
LOWLIM BSS  2            HOLD FOR THE LOWER LIMIT CHARACTER.
       EVEN

* R0 IS ADDRESS ON SCREEN
* R1 IS ADDRESS OF BUFFER
* R2 IS LENGTH

INPUT9
       LI   R0,32
       JMP  INPUTA

INPUT0
       LI   R0,33
INPUTA
       MOV  R0,@LOWLIM
       MOV  *R13,R0
       MOV  @4(R13),R2   GET THE LENGTH
       LI   R1,>2000
INPT01 BLWP @VSBW
       INC  R0
       DEC  R2
       JNE  INPT01

       MOV  *R13,R7      ADDRESS ON SCREEN
       MOV  @2(R13),R8   ADDRESS OF BUFFER
       INC  R8           LEAVE SPACE FOR THE LENGTH
       CLR  R9           LENGTH COUNTER

INPT02
       LIMI 2
       LIMI 0

       MOV  R7,R0        GET SCREEN ADDR ES
       LI   R1,>1E00     CURSOR CHARACTER
       BLWP @VSBW        PUT UP THE CURSOR

       BLWP @KSCAN
       MOVB @>837C,R0    GET THE STATUS BYTE
       ANDI R0,>2000
       JEQ  INPT02       NO NEW KEY . . . KEEP WAITING
```

```
       MOV  R7,R0
       LI   R1,>2000
       BLWP @VSBW          TAKE DOWN THE CURSOR

       MOVB @>8375,R1      GET THE KEY

       CB   R1,@CR         IS IT A C/R?
       JEQ  INPT03         GET OUT!

       CB   R1,@BACKSP     IS IT A BACK SPACE?
       JNE  INPT04         NOPE. SKIP AHEAD

       MOV  R9,R9          IS LENGTH 0?
       JEQ  INPT02         YES. KEEP WAITING.

       DEC  R9             DECREMENT LENGTH
       MOV  R7,R0          GET SCREEN ADDRESS
       LI   R1,>2000       GET A SPACE CHARACTER
       BLWP @VSBW          PUT UP THE SPACE
       DEC  R7             DECREMENT THE SCREEN ADDRESS
       DEC  R8             DECREMENT THE BUFFER ADDRESS
       JMP  INPT02         AND WAIT FOR NEXT CHARACTER

INPT04
       MOV  R1,R5          *
       SRL  R5,8           * IF CHAR IS LESS THAN AN
       C    R5,@LOWLIM     * ASCII 33, DON'T TAKE IT!
       JLT  INPT02         *

       CI   R5,126         * IF CHAR IS GREATER THAN AN
       JGT  INPT02         * ASCII 126, DON'T TAKE IT!

       MOVB R1,*R8+        PUT CHARACTER IN THE BUFFER

       MOV  R7,R0          GET SCREEN ADDRESS
       BLWP @VSBW          PUT UP THE CHARACTER

       INC  R7             INCREMENT SCREEN BUFFER

       INC  R9             INCREMENT THE COUNTER
       C    R9,@4(R13)     COMPARE WITH LENGTH
       JNE  INPT02         IF NOT DONE, GO BACK

INPT03
       SWPB R9
       MOV  @2(R13),R1
       MOVB R9,*R1         PUT IN LENGTH BYTE

       RTWP                AND RETURN
```

Disk 83. Contents of file MG/S

```
GPLWS  EQU  >83E0        GPL workspace
GR4    EQU  GPLWS+8      GPL workspace R4
GR6    EQU  GPLWS+12     GPL workspace R6
STKPNT EQU  >8373        GPL Stack pointer
LDGADD EQU  >60          Load & Execute GROM address entry point
XTAB27 EQU  >200E        Low Mem XML table location 27
GETSTK EQU  >166C

GPLLNK DATA GLNKWS  R7   Set up BLWP Vectors
       DATA GLINK1  R8

RTNAD  DATA XMLRTN  R9   Address where GPL XML returns to us
GXMLAD DATA >176C   R10  GROM Address for GPL XML (0F 27 Opcode)
       DATA >50     R11  Initialized to >50 where PUTSTK address resides

GLNKWS EQU  $->18                GPLLNK's workspace of which only
       BSS  >08     R12-R15      registers R7 through R15 are used

GLINK1 MOV  *R11,@GR4            Put PUTSTK Address into R4 of GPL WS
       MOV  *R14+,@GR6           Put GPL Routine Address in R6 of GPL WS
       MOV  @XTAB27,R12          Save the value at >200E
       MOV  R9,@XTAB27           Put XMLRTN Address into >200E
       LWPI GPLWS                Load GPL WS
       BL   *R4                  Save current GROM Address on stack
       MOV  @GXMLAD,@>8302(R4)   Push GPL XML Add on stack for GPL RTurn
       INCT @STKPNT              Adjust the stack pointer
       B    @LDGADD              Execute our GPL Routine

XMLRTN MOV  @GETSTK,R4           Get GETSTK pointer
       BL   *R4                  Restore GROM address off the stack
       LWPI GLNKWS               Load our WS
       MOV  R12,@XTAB27          Restore >200E
       RTWP                      All Done - Return to Caller


PUTSTK EQU  >50          Push GROM Add to stack pointer
TYPE   EQU  >836D        DSRLNK Type byte for GPL DSLLNK
NAMLEN EQU  >8356        Device name length pointer in VDP PAB
VWA    EQU  >8C02        VDP Write Arress location
VRD    EQU  >8800        VDP Read Data byte location
GR4LB  EQU  >83E9        GPL Workspace R4 Lower byte
GSTAT  EQU  >837C        GPL Status byte location

DSRLNK DATA DSRWS,DLINK1         Set BLWP Vectors

DSRWS  EQU  $                    Start of DSRLNK workspace
DR3LB  EQU  $+7                  R3 lower byte of DSRLNK workspace
DLINK1 MOV  R12,R12       R0     Have we already looked up the LINK address?
```

```
       JNE  DLINK3        R1       YES! Skip look up routine
       LWPI GPLWS         R2,R3    Else load GPL workspace
       MOV  @PUTSTK,R4    R4,R5    Store current GROM address on the stack
       BL   *R4           R6          .
       LI   R4,>11        R7,R8    Load R4 with address of LINK routine vector
       MOVB R4,@>402(R13) R9,R10   Set up GROM with address for vector
       JMP  DLINK2        R11      Jump around R12-R15
       DATA 0             R12      contains >2000 flag when set
       DATA 0,0,0         R13-R15  contains WS, PC & ST for RTWP
DLINK2 MOVB @GR4LB,@>402(R13)      Finish setting up GROM address
       MOV  @GETSTK,R5             Take some time & set up GETSTK pointer
       MOVB *R13,@DSRAD1           Get the GPL DSR LINK vector
       INCT @DSRADD                Adjust it to get past GPL FETCH instruction
       BL   *R5                    Restore the GROM Address off the stack
       LWPI DSRWS                  Reload DSRLNK workspace
       LI   R12,>2000              Set flag to signify DSRLNK address is set
*<<-------------------------------------------------------------------------->>*

DLINK3 INC  R14                   Adjust R14 to point to Callers DSR Type byte
       MOVB *R14+,@TYPE            Move it into >836D for GPL DSRLNK
       MOV  @NAMLEN,R3             Save VDP address of Name Length
       AI   R3,-8                  Adjust it to point to PAB Flag byte
       BLWP @GPLLNK                Execute DSR LINK
DSRADD BYTE >03                    High byte of GPL DSRLNK address
DSRAD1 BYTE >00                    Lower byte of GPL DSRLNK address

*------Error Check & Report to Callers R0 and EQU bit ----------------------
       MOVB @DR3LB,@VWA            Set up LSB of VDP Add for Error Flag
       MOVB R3,@VWA                Set up MSB of VDP Add for Error Flag
       SZCB R12,R15                Clear EQ bit for Error Report
       MOVB @VRD,R3                Get PAB Error Flag
       SRL  R3,5                   Adjust it to 0-7 error code
       MOVB R3,*R13                Put it into Callers R0 (msb)
       JNE  SETEQ                  If its not zero set EQ bit
       COC  @GSTAT,R12             Else test CND bit for Link Error (00)
       JNE  DSREND                 No Error Just return
SETEQ  SOCB R12,R15                Error so set Callers EQ bit
DSREND RTWP                        All Done - Return to Caller
```

Disk 83. Contents of file SAVE/S

```
* SAVE/S FROM J. PETER HODDIE

       AORG >A000

       DEF  SAVE

       REF  DSRLNK
       REF  VSBW,VMBW
       REF  VSBR,VMBR

BEGIN  EQU  >2E00
END    EQU  >38A0
PAB    EQU  >1000
PABBUF EQU  PAB+>20
PABDAT DATA >0600,PABBUF,0,END-BEGIN+6,13
       TEXT 'DSK1.SORT/EXP'
       EVEN

HEADER DATA 0,END-BEGIN+6,BEGIN

SAVE
       LI   R0,PABBUF
       LI   R1,HEADER
       LI   R2,6
       BLWP @VMBW

       LI   R0,PABBUF+6
       LI   R1,BEGIN
       LI   R2,>1000
       BLWP @VMBW

       LI   R0,PAB
       LI   R1,PABDAT
       LI   R2,>1F
       BLWP @VMBW
       AI   R0,9
       MOV  R0,@>8356
       BLWP @DSRLNK
       DATA 8

       BLWP @0

       END
```

Disk 83. Contents of file SORT/DOC

SORT EXPERIMENT

V1.0 April 26, 1987, 3:45 AM

J. Peter Hoddie
12 Paul Revere Road
Lexington, MA 02173

Features:

* Sort any type of file of any record size (except PROGRAM)
* Sort on up to 8 different user defined fields
* Handles up to 1000 records or 24K of data (whichever comes first)
* Sort in either ascending or descending order
* Extremely fast compared to other /4A sorts
* Choose between Shell and Shuttle sorts

The Files:

The file SORT/EXP is the actual program itself. The documentation (this file) is called SORT/DOC. The source code files to the program are SORT/EXP/S, INPUT/S, UTILS/S, and DSRLNK/S. The UTILS file is a piece of a file from GRAM Packer, and the INPUT routine is the same one used in GRAM Packer and some code I wrote for Myarc. The DSRLNK is a standard TI DSRLNK, not the Millers Graphics GROM based DSRLNK (which I actually prefer) because of the system this program was developed on. The MG DSRLNK could easily be substituted. The remaining file SAVE/S is a junk program to create the PROGRAM image version of this program (in the file DSK1.SORT/EXP) given a DIS/FIX 80 file created from the source files.

Loading the Program:

The program itself is in Program Image format. To run it simply use any program image loader such as Editor/Assembler option 5, TI-Writer option 3, or Funnelweb. The filename to use is DSKn.SORT/EXP. The program will autostart.

Using the Program:

When the program starts up you will be presented with a title screen. Press any key to bypass this screen. First you will be asked for an input filename. This is the name of the file that you want to sort. It can be of any file type except PROGRAM. If the file cannot be loaded you will be returned to the filename prompt; otherwise the file will be loaded. Up to 1000 records or about 24K of data (about 96 sectors) can be loaded, whichever comes first.

Next you will be prompted for the fields to sort on. You begin with field 1. You are first asked for starting position of the field. This is a number from 1 to whatever the maximum record length is in your file. Next you are prompted for field length. This is the number of characters in the field. If you want to use the rest of the record, enter a value of 0. Pressing ENTER for length is the same as entering a zero. You will then be asked the same information for field 2, and so on up to field 8. You may end this input at any time by just hitting ENTER for the START value of any field. If you just want to sort the file as one big record enter 1 for Start of field 1 for DISplay files, ENTER for length, and ENTER for Start of field 2. (For INTernal files, you should enter 2 for Start of field 1, since the first byte in an internal file is a length byte.) After you have entered the field data you will be asked if the information is correct. If it is, hit Y; otherwise hit N and you will be able to re-enter the information.

You will next be prompted for Descending or Ascending sort. Usually you will want Ascending, which means that the largest values in the file are sorted to the bottom.

Finally you are prompted for the type of sort to use. The options are Shell and Shuttle. In general you should use Shell for long files and Shuttle for short files. This rule is not always true, so you should experiment with both types of sorts to see which works best for your files.

After entering all this information, the file will be sorted. This can take anywhere from 1 second to about 4 minutes depending on the number of records, the number of fields, and the length of the fields.

When the sort is complete, you will be prompted for a SAVE filename. If you enter the same filename as the LOAD file then the original file will be destroyed. In any case, the file created will be of the same type as the file loaded so that it can easily be used in the application that created it.

An Example:

The way this program was tested was using one of the source code files. Assembly code is usually arranged in 3 columns so it is an easy way to test a multi-field sort. Run SORT/EXP and use DSKn.INPUT/S as the "LOAD filename." This file is about 100 records. For the first field enter Start=8 and Length=4. This will sort the so called "op-code" field first. For the second field enter Start=1 and Length=6. This will sort the "label" field second. Finally enter Start=13 and Length=0 for the third field to use the remaining portion of the record to sort. For the fourth field simply hit enter. You will be asked if the data is correct. If it is, hit Y; if it is not, as explained above, hit N. Next you will be prompted for Ascending or Descending sort. Hit 2 to choose Ascending sort as this is generally easier to verify visually. Ascending essentially means alphabetical order whereas Descending is reverse alphabetical. Finally you will be prompted for the type of sort to use. Select 1 for Shell sort although in this case a Shuttle sort would probably work as well. The sorting will now begin. It should only take about 6 seconds or so. When it is done you will be prompted for a save filename. Enter a filename that is different from the one you created such as DSKn.SORTED. When the save is complete the program will restart. Hit **FCTN =** to get out of the program. Now load up TI-Writer and look at the file you saved to see how the sort worked out.

Notes on the Program:

The way this program works is fairly typical of data manipulation programs for the /4A. The program itself resides in low memory (>2000 - >3FFF) and uses high memory (>A000 - >FFFF) as a data buffer. The load routine first does a STATUS call to the disk drive to determine what type of file the user has asked to load and then opens the file. The file is loaded into high memory, one record after another separated by length bytes. For each string a pointer is entered into a table in low memory so that each string may be looked up quickly based on its number. The sort routines themselves are quite straightforward and the BASIC code that they are based on is included in the source files. It would be very easy to add other types of sorts to this package (a "quick sort" routine would be most useful) because of the way it is structured. There are 2 subroutines which should be used by all sort routines: COMPAR, which compares 2 strings based on the field data supplied by the user, and SWAP which simply swaps the values of 2 strings. Furthermore, the handling of ascending/descending sort is handled within the COMPAR routine itself by essentially negating the result of the compare if the user requests a descending sort. It is my feeling that the COMPAR routine could be sped up a bit more, but final exams and lack of patience prohibits that at this time.

The main reason, I believe, that this sort package is noticeably faster than any other sort package for the TI is because the SWAP and COMPAR routines really never move any data. The COMPAR routine compares the two strings where they are in memory without ever moving either one. The SWAP routine only has to swap pointers (4 bytes) instead of the entire string (typically 160 bytes in most programs). By reducing the amount of data that has to be moved, a significant speed increase is obtained. Another reason for the speed of this program is that it makes extensive use of workspace registers rather than "variables." The /4A is capable of accessing registers faster then "variables" and because these registers have been placed in scratch pad RAM (zero wait state) they are extremely fast. However, there is certainly room for improvement in this program, particularly in the way of improving the COMPAR routine which is the most used code in the program, and by using more efficient sort routines. I am willing to work with any programmer who is interested in modifying this program.

Why I Wrote This Program:

This program is called "Sort Experiment" because in my mind it is just that. I didn't write it because I needed a sort routine. I wrote it to see how fast I could get the /4A to handle a multi-field sort. I was interested in this because I am considering writing a database program of some kind. I believe a sort routine is a central part of this. Of course a sort for a data base must allow for many more options such as true numerical comparisons, ability to ignore case, selection of ascending/descending for each field, and much more. If/when I do begin work on this database program it will probably use none of the code in this routine, only the knowledge I gained in writing it. One of the reasons for releasing the source code is to see if some brilliant programmer out there can find a way to speed this stuff up significantly, using some techniques I may have missed. Many people have thoughts on a database. I would like to hear from any of you that are interested in such a product. I am not planning on creating dBase III, but something that is significantly more powerful then what is currently available. I would create versions of this program for both the /4A and the Geneve, since the memory and speed capabilities of the Geneve allow for a degree of power not possible on the standard /4A system.

Why Fairware?

In general I don't release fairware programs. Either my programs are commercial releases such as Pre-Scan It!, Font Writer, XB:Bug, and GRAM Packer, or are essentially "given away" in MICROpendium or Genial TRAVelER. This program is fairware because it is too major an effort to just "give away" but I don't consider it to be worthy of a major marketing effort. There have been many people in the /4A community who have requested more powerful sorting tools. I believe this program is a major step in that direction. It is significantly more powerful than the excellent Romer/Clulow "TI-Sort" package which has been the standard for quite some time now. That routine only sorted DIS/VAR 80 files and could only handle a maximum of 300 records. It could also only sort on 2 fields and was on the order of 3 times slower than this package. If you find this program useful (that means that you USE it, perhaps not every day, but that it performs a task better than what you used before, or that you could not do before), please send a fairware donation. In general, I understand from the experiences of many of my _friends_, fairware does not work. Please show me that it can. I have ideas for several other programs that could be released as fairware if this program proves to be worthwhile.

Thank you.

Closing Note:

There are a couple reasons why I insist on written permission before any organization distributes this program. First is so that I can get some idea of the distribution that this program is getting. Second is to keep organizations like XXX-XXXX and the XXXXXXXX XXXXXXXXXX XXXXX from selling copies to people for $10 (or more), a practice which is an insult and a rip off to both the author and the customer. If your user group is interested in distributing the program, just drop me a quick note saying so. I'll get written permission out to you ASAP.

Disk 83. Contents of file SORT/EXP/S

```
* SORT/EXP/S FROM J. PETER HODDIE

       DEF  START,STOP

*      REF  INPUT             Aorg-ed at >2E00

*      REF  DSRLNK
*      REF  KSCAN
*      REF  VSBW,VMBW
*      REF  VSBR,VMBR
       REF  VDPWD

RECS   EQU  1000              total number of records
RECLEN BSS  2                 length of each record (include length byte)
BUFBAS EQU  >A000             base address of the data buffer
BUFTOP EQU  >FFF8-256         top address of the data buffer
*                              -256 is because of way BUFTOP is checked

PAB    EQU  >2000             address of pab
PABBUF EQU  PAB+>40           address of pab buffer

WS     EQU  >8300             a nice _fast_ ws for the /4A.

       AORG >2E00             make sure it is low memory
       B    @START

* CONSTANTS AND THAT SORT OF THING
H00
OPEN   DATA 0
HFFFF  DATA -1
D10    DATA 10
CLOSE
H01    BYTE >01
READ
H02    BYTE >02
WRITE
H03    BYTE >03
STATUS
H09    BYTE >09
OUTMOD BYTE >02               data for opening PAB for output
INMOD  BYTE >04               data for opening PAB for input
SAVMOD BSS  1                 place to save constructed mode base byte
SPACE
H20    BYTE >20
ZERO   TEXT '0'
NUMBUF BSS  6                 buffer for fielding numeric input
       EVEN
```

```
*TABLE  BSS  RECS*2           create a table of record pointer
TABLE  EQU  >2100            put the table where it won't get SAVEd

RECHI  BSS  2                space to hold high record number loaded

PABDAT DATA >0000,PABBUF,>0000,0,9
       TEXT 'DSK2.FILE'
       TEXT '                        '       24 spaces?
       TEXT '    '                          and some for safety
       EVEN

* CONSTANTS RELATING TO TEXT LAYOUT
WIDTH  EQU  40               screen width
LMAR   EQU  3                left margin
LINE1  EQU  3                value of first used line

TEXT01 TEXT 'LOAD Filename:'                14 bytes
TEXT02 TEXT 'SAVE Filename:'
TEXT03 TEXT '1=Shell 2=Shuttle'             17 bytes
TEXT04 TEXT '1=Descending 2=Ascending'      24 bytes
TEXT05 TEXT 'Sorting . . .'                 13 bytes
TEXT06 TEXT 'Sort Experiment'               15 bytes
TEXT07 TEXT 'Field #  Start  Length'        22 bytes
TEXT08 TEXT '(C) 1987 J. Peter Hoddie'      24 bytes
TEXT10 TEXT 'Is this correct?'              16 bytes
TEXT09 TEXT 'This  is a  Fairware  program.' 30 bytes
       TEXT 'It  may  only  be  distributed'
       TEXT 'with  all   documentation  and'
       TEXT 'source  files.    It   may  be'
       TEXT 'copied by any individual.  Any'
       TEXT 'commercial   organization   or'
       TEXT 'user group must obtain written'
       TEXT 'permission  from the author to'
       TEXT 'distribute.  If you  find this'
       TEXT 'program useful please send $10'
       TEXT 'to  the  author.  Comments and'
       TEXT 'suggestions always welcome.   '
       TEXT '        J. Peter Hoddie       '
       TEXT '      12 Paul Revere Road     '
       TEXT '      Lexington, MA 02173     '
*            01234567890123456789012345678 9
       EVEN

* field record format is Start,Length
*  Start counts starting with 1, not 0.
*  Terminate list of records with a -1,x for data
*  To indicate length of "rest of field" use Length=0
FSTART BSS  10*2*2           allow for 10 fields

START
```

```
       LWPI WS                 give us some fast memory to work with

       LI   R0,>01F0           40 columns
       BLWP @VWTR
       MOVB @WS+1,@>83D4       tell the OS we are in 40 columns

       LI   R0,>07F5           E/A color scheme
       BLWP @VWTR

       MOVB @H03,@>8375        set keyboard number to uppercase only

* CLEAR THE POINTER TABLE
       LI   R0,TABLE
       LI   R1,RECS
CB10   CLR  *R0+
       DEC  R1
       JNE  CB10

       BL   @CLS               clear the screen

* PUT UP THE TITLE TEXT
       LI   R0,5*WIDTH+5
       LI   R1,TEXT09
       LI   R2,30
       LI   R3,15
TITLE0 BLWP @VMBW
       A    R2,R1
       AI   R0,WIDTH
       DEC  R3
       JNE  TITLE0

TITLE1
       BLWP @KSCAN
       CB   @>8375,@HFFFF
       JNE  TITLE1             wait to get off old key

TITLE2 BLWP @KSCAN             wait for any keypress
       CB   @>8375,@HFFFF
       JEQ  TITLE2

       BL   @CLS

* GET THE INPUT FILENAME
       LI   R0,LINE1*WIDTH+LMAR
       LI   R1,TEXT01
       LI   R2,14
       BLWP @VMBW              prompt
       AI   R0,WIDTH+1         location of field on screen
       BL   @GETFIL            get filename

* DO A STATUS CHECK ON THE FILE
```

```
       LI   R0,PAB
       LI   R1,PABDAT
       LI   R2,>30
       BLWP @VMBW
       MOVB @STATUS,R1
       BLWP @VSBW             change PAB to STATUS opcode
       BL   @DSR              perform the STATUS call
       LI   R0,PAB+8
       BLWP @VSBR             get the status byte
       MOVB R1,R2             make a copy of the byte
       CLR  R3                a byte to build the file flags in
       ANDI R1,>0400          isolate Variable/Fixed flag
       JEQ  STA10             flag is reset=FIXED
       AI   R3,>1000          set bit indicating Variable
STA10  MOVB R2,R1             get back copy of STATUS byte
       ANDI R1,>1000          isolate Internal/Display flag
       JEQ  STA15             flag is reset=DISPLAY
       AI   R3,>0800          set bit indicating INTERNAL
STA15
       MOVB R3,@SAVMOD        save the created base mode byte
       AB   @INMOD,R3         now set for INPUT mode
       MOVB R3,R1
       LI   R0,PAB+1
       BLWP @VSBW             install byte in the PAB

* WE WILL ASSUME SEQUENTIAL ACCESS
* RECORD LENGTH IS DETERMINED FROM THE OPEN DSR

* OPEN THE FILE FOR INPUT
       LI   R0,PAB
       MOVB @OPEN,R1
       BLWP @VSBW             change PAB to OPEN mode
       BL   @DSR              open the file

       LI   R0,PAB+4          address of record length byte
       BLWP @VSBR             get the byte
       SRL  R1,8              make number a word
       MOV  R1,@RECLEN        remember this value

* PREPARE TO LOAD THE FILE
       CLR  R5                current record number
       LI   R6,BUFBAS         first entry into buffer
       LI   R7,TABLE+2        first entry into table

       LI   R0,PAB
       MOVB @READ,R1          get READ opcode
       BLWP @VSBW             change pab

* LOOP TO ACTUALLY LOAD THE FILE
LOAD20
```

```
       BL   @DSR              get next record

       LI   R0,PAB+1
       BLWP @VSBR             get error byte
       ANDI R1,>E000          isolate error bits
       JNE  LOAD90            error. we're done.

       LI   R0,PAB+5          address of length byte
       BLWP @VSBR             get length byte
       SRL  R1,8              make it a word
       MOV  R1,R2             put length byte in place

       MOV  R6,*R7+           put the address in the table
       MOV  R6,R1             address to put string in cpu
       A    R2,R6             update the address for next string
       INC  R6                include length byte in the count

       LI   R0,PABBUF         address of string in vdp
       MOV  R2,R2             null string?
       JEQ  LOAD25            don't move it
       SWPB R2
       MOVB R2,*R1+           install length byte
       SWPB R2
       BLWP @VMBR             get the string

LOAD25
       CI   R6,BUFTOP         are we pushing the top of the buffer
       JH   LOAD90            yup. kill rest of the load
       INC  R5                update record count
       CI   R5,RECS           see if this is last record . .
       JNE  LOAD20            not done yet

* LOADING IS COMPLETE
LOAD90
       MOVB @CLOSE,R1
       LI   R0,PAB
       BLWP @VSBW             change opcode to CLOSE
       BL   @DSR              CLOSE the file

       MOV  R5,@RECHI         zero record count?/remember hi rec num
       JNE  LOAD95            nope.
       B    @START            restart the program
LOAD95

* PREPARE TO SORT THE DATA

* GET THE FIELD DATA (THIS IS A ROYAL PAIN ....)
FINP00 LI   R0,LINE1+3*WIDTH+LMAR
       LI   R1,TEXT07
       LI   R2,22
       BLWP @VMBW             put up the fielding text
```

```
       CLR  R10               field number=0
       LI   R9,LINE1+4*WIDTH+LMAR            first line to use

FINP10
       MOV  R10,R1
       SWPB R1
       AI   R1,'1'*256        display field number +1
       MOV  R9,R0             screen address
       C    *R0+,*R0+         move it over a bit
       BLWP @VSBW             put up the number

       AI   R0,5              point to first field (start)
       LI   R1,NUMBUF
       LI   R2,4
       BLWP @INPUT            get the number
       MOV  R1,R0             get buffer pointer
       BL   @CNS              get an integer
       MOV  R0,R0             zero?
       JEQ  FINP10            get it again.
       C    R0,@HFFFF         error?
       JEQ  FINP90            done with input
       DEC  R0                take it back by 1.
*                              lets user enter from 1->80 rather then 0->79
       MOV  R10,R5            get field number
       SLA  R5,2              *4
       MOV  R0,@FSTART(R5)    put value in place

       MOV  R9,R0
       AI   R0,17             address of second field
       LI   R1,NUMBUF
       LI   R2,4
       BLWP @INPUT
       MOV  R1,R0
       BL   @CNS
       C    R0,@HFFFF         error?
       JNE  FINP20            nope.
       CLR  R0                assume a zero then
FINP20
       MOV  R0,@FSTART+2(R5)  put value in place

       AI   R9,WIDTH          next line
       INC  R10               next element
       CI   R10,8             done?
       JNE  FINP10            nope

FINP90
       MOV  R10,R10           did the user enter _anything_?
       JEQ  FINP00            nope. make them try again.
       MOV  R10,R5
       SLA  R10,2
```

```
       SETO @FSTART(R10)      indicate end of list

       LI   R0,18*WIDTH+LMAR
       LI   R1,TEXT10
       LI   R2,16
       BLWP @VMBW             check if user _thinks_ data is correct.

FINP95 BLWP @KSCAN
       MOVB @>8375,R3
FINP97 BLWP @KSCAN            wait for key to be released.
       CB   @>8375,R3
       JEQ  FINP97
       SRL  R3,8
       CI   R3,'Y'
       JEQ  SRTT10            ok
       CI   R3,'N'
       JNE  FINP95            not vaild key. try again
       MOVB @H20,R1
       LI   R0,LINE1+3*WIDTH+LMAR
FINP96 BLWP @VSBW             clear out
       INC  R0
       CI   R0,19*WIDTH
       JNE  FINP96
       B    @FINP00           restart fielding info

* DETERMINE IF ASCENDING OR DESCENDING SORT
SRTT10 LI   R0,18*WIDTH+LMAR
       LI   R1,TEXT04
       LI   R2,24
       BL   @GETNUM           prompt for sort order
       CI   R1,2
       JGT  SRTT10            out of range
       AI   R1,>0459          B *R9 +1 for 10, +2 for 11
       MOV  R1,@COMP97        patch out the code with user choice

* FIND OUT THE TYPE OF SORT THAT USER WANTS
SRTT20 LI   R0,20*WIDTH+LMAR
       LI   R1,TEXT03
       LI   R2,17
       BL   @GETNUM
       CI   R1,2
       JGT  SRTT20

       MOV  R1,R9             save the number for compare
       LI   R0,22*WIDTH+LMAR
       LI   R1,TEXT05
       LI   R2,13
       BLWP @VMBW             tell user we are busy

       CI   R9,1
       JEQ  SHELL
```

```
       JMP  SHUTLE            if not 1, must be 2

*         *** SHELL SORT ***

* USING THE RUSS WALTERS ALGORITHM
* FROM PAGE 82 OF "SECRET GUIDE TO COMPUTERS, VOLUME TWO
*  42 D=N
*  44 D=INT(D/5)+1
*  50 FOR I=1 TO N-D
*  60  FOR J=I TO 1 STEP -D
*       IF X$(J)>X$(J+D) THEN SWAP X$(J),X$(J+D)
*      NEXT J
*  70 NEXT I
*  75 IF D>1 THEN 44
*
* REGISTER USAGE
*   R7=D
*   N = @RECHI
*   R8 = I
*   R9 = J
*

SHELL

SRT42
       MOV  @RECHI,R7         D=N

SRT44
       MOV  R7,R1
       CLR  R0
       LI   R4,5
       DIV  R4,R0             do the division
       INC  R0                +1
       MOV  R0,R7             D=INT(D/5)+1

SRT50
       LI   R8,1              begin FOR I=1 to N-D

SRT60
       MOV  R8,R9             begin FOR J=I TO 1 STEP -D

SRT60B
       MOV  R9,R0             R0=J
       MOV  R9,R1
       A    R7,R1             R1=J+D
       BL   @COMPAR           compare these two strings
       JMP  SRT60A            R0=<R1 . . .no swap

       MOV  R9,R2             R2=J
       MOV  R9,R1
```

```
       A    R7,R1             R1=J+D
       BL   @SWAP

SRT60A
       S    R7,R9             J=J-D (NEXT starting .  . . . .)
       CI   R9,1              done?
       JGT  SRT60B            nope

SRT70
       INC  R8                I=I+1 (NEXT starting)
       MOV  @RECHI,R0         get N
       S    R7,R0             R0=N-D
       C    R8,R0
       JGT  SRT75             done with loop
       JMP  SRT60             continue loop

SRT75
       CI   R7,1              are we done?
       JGT  SRT44             not yet . . . .
       B    @SAVE


*  ***** SHUTTLE SORT ******
*
* BASED ON RUSS WALTERS "SUPER-FANCY 3 LINE SORT"
*
* 50 FOR I=1 TO N-1
* 60   FOR J=I TO 1 STEP -1 : IF X$(J)>X$(J+1) THEN SWAP : NEXT J
* 70 NEXT I
*
*  R7 = I
*  R8 = J
*  N = @RECHI
*

SHUTLE

SHUT50
       LI   R7,1              FOR I=1 TO N-1 ... start

SHUT60
       MOV  R7,R8             FOR J=I TO 1 . . . start

SHUT6B
       MOV  R8,R0             R0=J
       MOV  R0,R1
       INC  R1                R1=J+1
       BL   @COMPAR           is X(J)>X(J+1)
       JMP  SHUT6A            nope
       MOV  R8,R1
       MOV  R1,R2
```

```
       INC  R2
       BL   @SWAP

SHUT6A
       DEC  R8                next ,. .. sort of
       JNE  SHUT6B            not done with loop
SHUT70
       INC  R7                next I . . sort of . .
       C    R7,@RECHI         done?
       JNE  SHUT60            no. continue loop.
*      B    @SAVE             done.


SAVE
* HAVING SORTED THE DATA WE NOW MUST DUMP IT BACK TO DISK

* GET A FILENAME FROM THE USER
       LI   R0,22*WIDTH+LMAR  put around the _bottom_ of the screen
       LI   R1,TEXT02
       LI   R2,14
       BLWP @VMBW
       AI   R0,WIDTH+1
       BL   @GETFIL

*  NEXT UPDATE THE PAB IN VDP
       LI   R0,PAB
       MOVB @OPEN,R1
       BLWP @VSBW             change opcode to OPEN _again_
       INC  R0
       MOVB @OUTMOD,R1
       AB   @SAVMOD,R1        add into that the file type info
       BLWP @VSBW             change the mode to output
       LI   R0,PAB+9
       LI   R1,PABDAT+9
       LI   R2,>20
       BLWP @VMBW
       BL   @DSR              open the file

* PREPARE TO LOOP TO SAVE THE FILE
       LI   R0,PAB
       MOVB @WRITE,R1         get opcode for write
       BLWP @VSBW             change opcode

       CLR  R5                set record number to zero

* LOOP TO SAVE THE FILE
SAVE10
       MOV  R5,R1
       SLA  R1,1              *2
       MOV  @TABLE+2(R1),R1   get pointer to the next string
```

```
       JEQ  SAVE90            end of file . . .error or something

       MOVB *R1+,R2
       SRL  R2,8              get length of the string
*      MOV  R2,R2             is it a null string?
       JEQ  SAVE30

       LI   R0,PABBUF
       BLWP @VMBW             80 character to move to VDP for save

SAVE30
* PUT THE CHARACTER COUNT WHERE IT BELONGS
       SWPB R2
       MOVB R2,R1             put count in R1
       LI   R0,PAB+5          where count goes in VDP
       BLWP @VSBW             update the count

       BL   @DSR

       LI   R0,PAB+1          address of error byte
       BLWP @VSBR             read byte
       ANDI R1,>E000          error?
       JNE  SAVE90            yup

       INC  R5                next record
       C    R5,@RECHI         done?
       JNE  SAVE10            nope

* WE ARE DONE. CLOSE UP SHOP.
SAVE90
       MOVB @CLOSE,R1
       LI   R0,PAB
       BLWP @VSBW             change opcode to CLOSE
       BL   @DSR

* AND RESTAR THE PROGRAM
       B    @START

**********************
* COMPAR
* R0 is number of first record
* R1 is number of second
*  RT normal if R0=<R1
*  RT+2 if R0>R1

COMPAR
       MOV  R11,R10           make a copy of the return address

       LI   R12,FSTART        pointer to field table
       LI   R3,COMP90
```

```
       MOV  R0,R14            save this info
       MOV  R1,R15            ""

COMP00
       CB   *R12,@HFFFF       are we done with fielding loop?
       JNE  COMP02            nope
COMP20 B    *R3               yup. All fields were equal. Exit based on length
COMP02
       MOV  R14,R2            restore values
       MOV  R15,R1            ""

       SLA  R2,1
       SLA  R1,1
       MOV  @TABLE(R2),R0     get pointers to strings
       MOV  @TABLE(R1),R1

* GET THE LENGTH BYTES
       MOVB *R0+,R2
       MOVB *R1+,R3
       SRL  R2,8
       SRL  R3,8

       MOV  *R12,R13          get field start
       A    @2(R12),R13       add in length

       C    R2,R3
       JGT  COMP05            R2 is longer
*      MOV  R2,R2             use R2 length . . shorter
       LI   R3,COMP95
       JMP  COMP07            do it
COMP05
       MOV  R3,R2             use R3 length . . shorter
       LI   R3,COMP90

* CHECK IF FIELD IS CONTAINED IN THIS STRING
COMP07
       C   *R12,R2            is field beginning beyond end of string?
       JLT  COMP0A            ok
       JEQ  COMP0A            ok
       CB   @-1(R1),@1(R12)   compare field position vs length
       JGT  COMP20            includes field. compare done.
       JEQ  COMP20             ""
       MOV  R0,R4
       CB   @-1(R4),@1(R12)   compare field position vs length
       JGT  COMP20            includes field
       JEQ  COMP20            ""
       C    *R12+,*R12+       field was not in either string. next field.
       JMP  COMP00            go to next field.

COMP0A
```

```
       A    *R12,R0           add in field offset to position counter
       A    *R12,R1            ""

       C    R13,R2            is field end beyond end of string
       JLT  COMP08            nope
       JEQ  COMP08            nope
       S    *R12+,R2          get adjusted field length
       JMP  COMP10            and start the comparison

COMP08
       S    *R12+,R2          adjust length to account for new start
       MOV  *R12,*R12         is field length given as 0?
       JEQ  COMP10            yes. so use user length remaining
       MOV  *R12,R2           use the field width

COMP10 CB   *R0+,*R1+         compare next byte
       JGT  COMP90            if R0 is greater we're done
       JLT  COMP95            if R1 is greater we're done
       DEC  R2                decrement counter
       JNE  COMP10            not done yet
       INCT R12               point to next set of field data
       JMP  COMP00            done with this field. go to next one
COMP90
       INCT R11               exit for R0>R1
       JMP  COMP97
COMP95
       INCT R10
* The following statement gets patched for ascending/descending
*  sorts.  R11 is for descending. R10 for ascending.
COMP97
       RT                     exit for R0<=R1

************************
*  SWAP
*  R1 IS FIRST REC #
*  R2 IS SECOND
*  R0,R1,R2 ARE DESTROYED
*
SWAP
       SLA  R1,1
       SLA  R2,1
       MOV  @TABLE(R1),R0
       MOV  @TABLE(R2),@TABLE(R1)
       MOV  R0,@TABLE(R2)
       RT

*************************

DSR
       LI   R12,PAB+9
       MOV  R12,@>8356
```

```
       BLWP @DSRLNK
       DATA 8
       RT

************************
* GETFIL
*  this routine gets a filename for
*   _the_ PAB and that is about it
*   R0 is screen location

GETFIL
       LI   R2,25             length
       LI   R1,PABDAT+9       buffer address (includes length byte)
       BLWP @INPUT            get the line

       RT                     return having forgotten

*****************************
* GETNUM
* R0,R1,R2 ARE INPUT AS IN VMBW FOR PROMPT
* R1 RETURNS A NUMBER AS AN INTEGER (1 TO 9)
*

GETNUM
       BLWP @VMBW
       A    R2,R0
       INCT R0                where to display users key press

GETN10
       BLWP @KSCAN
       MOVB @>8375,R1
       MOVB R1,R2             save for debounce
       ANDI R1,>7F00
       CI   R1,>7F00
       JEQ  GETN10            don't show a "no key press" character
       BLWP @VSBW                          show the character they typed
       SRL  R1,8              make it a word
       AI   R1,-'0'           make it a number . . . if possible
       CI   R1,1
       JLT  GETN10            no good (small)
       CI   R1,9
       JGT  GETN10            no good  (too big)
* DEBOUNCE. WAIT FOR USER TO GET OFF KEY
GETN20 BLWP @KSCAN
       CB   R2,@>8375
       JEQ  GETN20
       RT                     good value. return

************************
*  CNS (CONVERT NUMBER TO STRING)
```

```
*   R0 IS POINTER TO STRING W/ LENGTH
*   R0 IS RETURNED WITH THE NUMBER
*   R0=-1 INDICATED AN ERROR
*   R1,R2,R3,R4  ARE TRASHED

CNS
       CLR  R1               r1=0 required for multiply
       CLR  R2               start out with a zero in accumulator
       MOVB *R0+,R4          get length
       JEQ  CNS89            length is zero. return an error.
       SRL  R4,8             make it a word
CNS10
       MOVB *R0+,R3          get next byte
       SB   @ZERO,R3         subtract off ASCII offset
       SRL  R3,8             make it a word
       MOV  R2,R1            Thank you, Tom Freeman!!!!!
       MPY  @D10,R1          multiply current value times 10
       MOV  R3,R3            compare r3 to 0
       JLT  CNS89            less then zero= error
       CI   R3,9
       JGT  CNS89            greater than 9=error
       A    R3,R2            put this number into the accumulator
       DEC  R4               down the length counter
       JEQ  CNS90            done
       JMP  CNS10            keep looping

CNS89  SETO R2               indicate an error

CNS90
       MOV  R2,R0            pass back return value
       RT                    and return

*****************************
* CLS
*  CLEAR THE SCREEN & PUT UP HEADER
*
CLS
       CLR  R0
       MOVB @SPACE,R1
       BLWP @VSBW            set VDP write address and do first byte
CLS10
       INC  R0               counter
       MOVB R1,@VDPWD        do other bytes _fast_
       CI   R0,WIDTH*24      done?
       JNE  CLS10            nope.

       LI   R0,12
       LI   R1,TEXT06
       LI   R2,15
       BLWP @VMBW            put up an ugly title
       LI   R0,WIDTH+8
```

```
       LI   R1,TEXT08
       LI   R2,24
       BLWP @VMBW             put up an ugly name . . .
       RT

       COPY "DSK1.INPUT/S"
       COPY "DSK1.DSRLNK/S"
       COPY "DSK1.UTILS/S"

STOP   END
```

Disk 83. Contents of file UTILS/S

```
* UTILS/S FROM J. PETER HODDIE

VSBW   DATA UTILWS,VSBW0
VMBW   DATA UTILWS,VMBW0
VSBR   DATA UTILWS,VSBR0
VMBR   DATA UTILWS,VMBR0
VWTR   DATA UTILWS,VWTR0
KSCAN  DATA UTILWS,KSCAN0

UTILWS BSS  32
AK     BSS  2
AR     EQU  2*2+1+UTILWS LOW BYTE OF R2 OF UTILWS

KSCAN0
       LWPI >83E0
       BL   @>000E
       LWPI UTILWS
       RTWP
VSBW0
       BL   @AM
       MOVB @>0002(R13),@>8C00
       RTWP

VMBW0
       BL   @AM
AN     MOVB *R1+,@>8C00
       DEC  R2
       JNE  AN
       RTWP

VSBR0
       BL   @AO
       MOVB @>8800,@>0002(R13)
       RTWP

VMBR0
       BL   @AO
AP     MOVB @>8800,*R1+
       DEC  R2
       JNE  AP
       RTWP

VWTR0
       MOV  *R13,R1
       MOVB @>0001(R13),@>8C02
       ORI  R1,>8000
       SWPB R4
       MOVB R1,@>8C02
```

```
        RTWP

AM      LI   R1,>4000
        JMP  AQ
AO      CLR  R1
AQ      MOV  *R13,R2
        MOVB @AR,@>8C02
        SOC  R1,R2
        SWPB R4
        MOVB R2,@>8C02
        MOV  @>0002(R13),R1
        MOV  @>0004(R13),R2
        B    *R11
```

Disk 83. Contents of file VDP/S

```
* XMLLNK AND VDP ACCESS UTILS

XMLLNK DATA UTILWS,XMLLN0
KSCAN  DATA UTILWS,KSCAN0
VSBW   DATA UTILWS,VSBW0
VMBW   DATA UTILWS,VMBW0
VSBR   DATA UTILWS,VSBR0
VMBR   DATA UTILWS,VMBR0
VWTR   DATA UTILWS,VWTR0

UTILWS BSS  32
AK     BSS  2
AJ     BSS  32
AR     EQU  2*2+1+UTILWS LOW BYTE OF AJ

XMLLN0
       MOV  *R14+,@>83E2
       LWPI >83E0
       MOV  R11,@AK
       MOV  R1,R2
       CI   R1,>8000
       JH   AL
       SRL  R1,12
       SLA  R1,1
       SLA  R2,4
       SRL  R2,11
       A    @>0CFA(R1),R2
       MOV  *R2,R2
AL     BL   *R2
       LWPI UTILWS
       MOV  R11,@>83F6
       RTWP

KSCAN0
       LWPI >83E0
       MOV  R11,@AK
       BL   @>000E
       LWPI UTILWS
       MOV  R11,@>83F6
       RTWP

VSBW0
       BL   @AM
       MOVB @>0002(R13),@>8C00
       RTWP

VMBW0
       BL   @AM
```

```
AN      MOVB *R1+,@>8C00
        DEC  R2
        JNE  AN
        RTWP

VSBR0
        BL   @AO
        MOVB @>8800,@>0002(R13)
        RTWP

VMBR0
        BL   @AO
AP      MOVB @>8800,*R1+
        DEC  R2
        JNE  AP
        RTWP

VWTR0
        MOV  *R13,R1
        MOVB @>0001(R13),@>8C02
        ORI  R1,>8000
        MOVB R1,@>8C02
        RTWP

AM      LI   R1,>4000
        JMP  AQ
AO      CLR  R1
AQ      MOV  *R13,R2
        MOVB @AR,@>8C02
        SOC  R1,R2
        MOVB R2,@>8C02
        MOV  @>0002(R13),R1
        MOV  @>0004(R13),R2
        B    *R11
```

Disk 83. Contents of file WC/S

```
       DEF  WC,SFIRST,SLOAD,SLAST

SFIRST
SLOAD
       B    @WC

       COPY "DSK2.MG/S"
       COPY "DSK2.VDP/S"
       COPY "DSK2.CIF/S"

* COLOR TABLE AT >20*>40
* PATTERN TABLE AT >800

VDPREG DATA >00E0,>0020,>0106,>0007
CURSOR DATA >003E,>3E3E,>3E3E,>3E3E
       DATA >3C42,>99A1,>A199,>423C  COPYRIGHT SIGN CHARACTER
PABDAT DATA >0014,PABBUF,>5000,0,>0000
       BSS  16
SPACE  TEXT ' '          SPACE CHARACTER
PERIOD TEXT '.'          A PERIOD
REQSPC TEXT '^'          FORMATTER REQUIRED SPACE CHARACTER
LET$Y  TEXT 'Y'          THE LETTER Y
REDO   BYTE 15           FCTN 8
BACK   BYTE 6            FCTN 9
TXT0   TEXT '** Word Count **' 16 BYTES
TXT1   TEXT 'Input File:'  11 BYTES
TXT2   TEXT 'Ignore Formatter commands?'   26 BYTES
TXT3   TEXT 'Scanning File' 13 BYTES
TXT4   TEXT 'Word Count:' 11 BYTES
TXT5   TEXT 'Line Count:' 11 BYTES
TXT6   TEXT 'BACK, REDO or QUIT' 18 bytes
TXT7   TEXT '                                  '  32 SPACES <BYTES>
C31
TXT8   BYTE >1F          COPYRIGHT SIGN
       TEXT '1986 J. Peter Hoddie' 21 bytes
       EVEN

FAC    EQU  >834A
PAB    EQU  >1000
PABBUF EQU  >1020
DSRPTR EQU  >8356

WS     BSS  32

FLGTIW BSS  2            TI-WRITER EDITOR FLAG
FLGFMT BSS  2            FORMATTER FLAG

BUFFER BSS  80
```

```
COUNT  BSS  2
LINCNT BSS  2

*  FLAGS:
*    TIW - CHANGES ALL CHARACTERS
*           BELOW THE SPACE TO A
*           SPACE.  I.E IF ASCII OF THE
*           CHARACTER IS 31 OR LESS
*           IS BECOMES A SPACE.
*
*    FMT - IGNORES ALL LINES BEGINNING
*           WITH A '.' AND REPLACES
*           REQUIRED SPACE WITH A
*           REAL SPACE.
*

WC
       LWPI WS

       CLR  @COUNT       CLEAR THE COUNTER
       SETO @FLGTIW      TI-WRITER TYPE FILE FLAG
       CLR  @FLGFMT      FORMATTER FILE FLAG
       CLR  @LINCNT      LINE COUNTER

* SET UP THE VDP REGISTERS
       CLR  R0
       CLR  R1           COUNTER
INIT0  MOVB @VDPREG(R1),@WS+1 INTO LOW BYTE OF R0
       BLWP @VWTR
       AI   R0,>0100
       INC  R1
       CI   R1,8
       JNE  INIT0
       MOVB @VDPREG+1,@>83D4  DON'T LET KSCAN SCREW US UP.

* LOAD CHARACTER SETS
       LI   R1,32*8+>800
       MOV  R1,@FAC
       BLWP @GPLLNK
       DATA >0018
       LI   R1,96*8+>800
       MOV  R1,@FAC
       BLWP @GPLLNK
       DATA >004A
       LI   R1,CURSOR
       LI   R0,30*8+>800 MAKE THE CURSOR AND A COPYRIGHT SIGN
       LI   R2,16
       BLWP @VMBW

* LOAD COLOR TABLE
```

```
       LI   R1,>1000
       LI   R0,>20*>40
       LI   R2,32
INIT1  BLWP @VSBW
       INC  R0
       DEC  R2
       JNE  INIT1

       CLR  R0
       LI   R1,TXT7
       LI   R2,32
CLS1   BLWP @VMBW
       A    R2,R0
       CI   R0,768
       JNE  CLS1

       LI   R0,>0500     KEY UNIT 5
       MOVB R0,@>8374    SET THE KEY UNIT

       LI   R0,32*23+5
       LI   R1,TXT8
       LI   R2,21
       BLWP @VMBW

       LI   R0,7
       LI   R1,TXT0
       LI   R2,16
       BLWP @VMBW

       LI   R0,32*4+2
       LI   R1,TXT1
       LI   R2,11
       BLWP @VMBW
       LI   R0,32*4+14
       LI   R1,PABDAT+9
       LI   R2,15
       BLWP @INPUT

       LI   R0,32*6+2
       LI   R1,TXT2
       LI   R2,26
       BLWP @VMBW
       LI   R0,32*6+29
       LI   R1,BUFFER
       LI   R2,1
       BLWP @INPUT
       MOVB @BUFFER,R1
       JEQ  WCA1
       CB   @BUFFER+1,@LET$Y
       JEQ  WCA1
       LI   R1,'N'*256
```

```
        LI   R0,32*6+29
        BLWP @VSBW
        JMP  WCA2
WCA1    MOVB @LET$Y,R1
        LI   R0,32*6+29
        BLWP @VSBW
        SETO @FLGFMT

WCA2
        LI   R0,32*10+8
        LI   R1,TXT3
        LI   R2,13
        BLWP @VMBW

        LI   R0,PAB
        LI   R1,PABDAT
        LI   R2,32
        BLWP @VMBW

        LI   R9,PAB+9
        MOV  R9,@DSRPTR

        BLWP @DSRLNK
        DATA 8

        LI   R1,>0200     READ
        LI   R0,PAB
        BLWP @VSBW

WC1
        MOV  R9,@DSRPTR
        BLWP @DSRLNK
        DATA 8

        LI   R0,PAB+1
        BLWP @VSBR        GET THE ERROR BYTE
        ANDI R1,>E000     ISOLATE ERROR BITS
        JNE  WCX          GET OUT
        MOVB @>837C,R1
        ANDI R1,>2000     CHECK THE COND BIT FOR BAD DEVICE NAME
        JNE  WCX          IF BIT IS SET, GET OUT.

        INC  @LINCNT      INCREMENT LINE COUNTER

        LI   R0,PAB+5
        BLWP @VSBR        GET CHARACTER COUNT
        SRL  R1,8         MAKE IT A WORD
        MOV  R1,R4        SAVE IT

        LI   R0,PABBUF
```

```
       LI   R1,BUFFER
       MOV  R4,R2
       BLWP @VMBR

       CLR  R5           OFFSET INTO BUFFER
       SETO R8           SET THE SPACE FLAG

       MOV  @FLGFMT,R0   IS FORMATTER FLAG SET?
       JEQ  WC2
       CB   @BUFFER,@PERIOD   IS IT A FORMATTER COMMAND?
       JEQ  WC1          YES.  GET NEXT RECORD. IGNORE THIS ONE.

WC2
       MOVB @BUFFER(R5),R1    GET THE BYTE

       MOV  @FLGTIW,R0   TI-WRITER FLAG SET?
       JEQ  TIWX
* TI WRITER FLAG HANDLING.
       CB   R1,@C31      COMPARE TO AN ASCII 31
       JGT  TIWX
       MOVB @SPACE,R1    MAKE IT A SPACE
TIWX
       MOV  @FLGFMT,R0   FORMATTER FLAG SET?
       JEQ  FMTX

       CB   R1,@REQSPC   REQUIRED SPACE?
       JNE  FMTX
       MOVB @SPACE,R1    MAKE IT A SPACE
FMTX

       CB   R1,@SPACE    IS IT A SPACE?
       JNE  WC3

       MOV  R8,R8        IS R8 SET?
       JNE  WC4          IF SET, DON'T COUNT. SKIP AHEAD
       INC  @COUNT
       SETO R8           SET FLAG
       JMP  WC4          INCREMENT. SKIP FLAG CLEAR

WC3
       CLR  R8
WC4
       INC  R5           INCREMENT OFFSET
       C    R5,R4        COMPARE TO LENGTH
       JNE  WC2          IF NOT EQUAL, WE AIN'T DONE.
* WE ARE AT THE END OF THE STRING . . .
* IF R8 IS SET . . . DON'T ADD ONE . . OTHERWISE DO.
       MOV  R8,R8        IS R8 SET?
       JNE  WC5          IT IS SET. SKIP INCREMENT

       INC  @COUNT       INCREMENT
```

```
WC5
       JMP  WC1

WCX
       LI   R1,>0100
       LI   R0,PAB
       BLWP @VSBW        SET PAB TO CLOSE

       MOV  R9,@DSRPTR
       BLWP @DSRLNK      CLOSE THE FILE
       DATA 8

* MAKE THE NUMBER A FLOATER
       MOV  @COUNT,@FAC
       BLWP @CIF         MAKE IT A FLOATER

*  MAKE THE NUMBER A STRING
       CLR  R0
       MOVB R0,@FAC+11   INDICATE TI BASIC FORMAT
       BLWP @GPLLNK
       DATA >0014        CONVERT NUMBER TO STRING

       LI   R0,32*10
       LI   R1,TXT7
       LI   R2,32        * SPACE OUT THE LINE . .
       BLWP @VMBW

       MOVB @FAC+11,R1   GET MOST OF ADDRESS
       SRL  R1,8         MAKE IT A WORD
       AI   R1,>8300     AND GET THE REST OF IT
       MOVB @FAC+12,R2   GET MOST OF LENGTH
       SRL  R2,8         MAKE IT A WORD
       LI   R0,32*10+19  ADDRESS ON SCREEN
       BLWP @VMBW        WRITE THE NUMBER
       LI   R0,32*10+8
       LI   R1,TXT4
       LI   R2,11
       BLWP @VMBW

* MAKE THE NUMBER A FLOATER
       MOV  @LINCNT,@FAC
       BLWP @CIF         MAKE IT A FLOATER

*  MAKE THE NUMBER A STRING
       CLR  R0
       MOVB R0,@FAC+11   INDICATE TI BASIC FORMAT
       BLWP @GPLLNK
       DATA >0014        CONVERT NUMBER TO STRING

       MOVB @FAC+11,R1   GET MOST OF ADDRESS
```

```
       SRL  R1,8          MAKE IT A WORD
       AI   R1,>8300      AND GET THE REST OF IT
       MOVB @FAC+12,R2    GET MOST OF LENGTH
       SRL  R2,8          MAKE IT A WORD
       LI   R0,32*12+19   ADDRESS ON SCREEN
       BLWP @VMBW         WRITE THE NUMBER
       LI   R0,32*12+8
       LI   R1,TXT5
       LI   R2,11
       BLWP @VMBW

       LI   R0,23*32
       LI   R1,TXT7
       LI   R2,32
       BLWP @VMBW

       LI   R0,23*32+5
       LI   R1,TXT6
       LI   R2,18
       BLWP @VMBW

WAIT   BLWP @KSCAN
       LIMI 2
       LIMI 0
       MOVB @>8375,R1
       CB   R1,@REDO
       JEQ  RESET
       CB   R1,@BACK
       JEQ  RESET
       JMP  WAIT

RESET  B    @WC

       COPY "DSK2.INPUT/S"    GET MY INPUT ROUTINE

SLAST  END
```

Disk 84. South Pacific

Version:
Requires: XB

Author: Ken Gilliland
Language: XB

Updated: 07/15/87

Selections from the musical that your TI actually sings and plays for you. Requires a Speech Synthesizer.

```
dskdir. v2.0. 12-dec-96
Disk name               = S/PACIFIC
Sectors total           = 360
Sectors used            = 335
Sectors available       = 23
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 *READTHIS!   13 DIS/VAR 80   >022 012
002 >003 ASMB         21 PROGRAM    Y >02e 020
003 >004 BALI3        35 PROGRAM    Y >042 034
004 >005 BALI4        10 PROGRAM    Y >064 009
005 >006 BALI5        10 DIS/VAR 80   >06d 009
006 >007 COMINGSOON    4 PROGRAM    Y >076 003
007 >008 DSKLABEL_P   25 PROGRAM    Y >079 024
008 >009 ENCHANT3     41 PROGRAM    Y >091 040
009 >00a ENCHANT4     10 PROGRAM    Y >0b9 009
010 >00b ENCHANT5      9 DIS/VAR 80   >0c2 008
011 >00c ENCHANT_P    25 PROGRAM    Y >0ca 024
012 >00d LOAD         22 PROGRAM    Y >0e2 021
013 >00e MENU1        10 PROGRAM    Y >0f7 009
014 >00f PACIFIC      49 DIS/FIX128 Y >100 048
015 >010 SP            2 DIS/VAR 80   >130 001
016 >011 SPRING3      32 PROGRAM    Y >131 031
017 >012 SPRING4      10 PROGRAM    Y >150 009
018 >013 SPRING5       7 DIS/VAR 80   >159 006
```

Disk 84. Contents of file *READTHIS!

Here's briefly what files you may wish to access on this disk:

LOAD
This loads the Program and accesses all program not mention in this note.

DSKLABEL_P
This is the artwork for the Disk Label. This is a TI-ARTIST picture file, but can be accessed by the public domain program MAX-RLE and printed.

ENCHANT_P
This is your mini-poster for the South Pacific Album. Yes, it's also a TI-ARTIST Picture file.

*READTHIS!
If you don't know what this is, I can't help you!

IMPORTANT NOTE: Since this disk writes to a FILE on the DISK, YOU CAN NOT PUT A WRITE-PROTECT TAB on this DISK. If you do so, you will be greeted by an annoying honk and error statement, and nobody wants this to happen.

For the Technical Minded — or those just plain interested:

Here's how the disk works. When the LOAD program boots, it also loads assembly language for an assembly language graphic screen loader and for lowercase characters. Now after a selection is made, it loads the Screen data for the appropriate piece and write the selection letter ("A,B, or C") to the SP file, that's why no write protect tab-- It then loads the ASMB file. Now the first thing this file does is to tell the 32K memory to forget everything about the screen loader and lowercase, since text-to-speech assembly requires almost the entire 32K memory. It then remembers the Speech Assembly hidden in the ASMB file and takes a look at the SP to see what music you selected. It then loads it, loads more Speech Assembly and the words to the song. After the piece is finished it then forgets all that Speech Assembly and goes to the LOAD program, thus remembering the screen loader and lowercase. That's how it works in a nut shell!

How I got the computer to sing on key is which my own personal singing editor, tat I may released someday, if there's a demand. If you don't really want to send me cash for this program-- I understand. What I do ask is that you at least write and tell me what to think. . . even if it's negative. The address boot in the load program if you need it.

Here's a quick list of what's available to date from me:

(all disks are $5 donations)
The TE2 Encoder (SSSD)
Richie Wagner's Greatest Hits (SSDD or DSSD)
The 1986 Girlie Calendar (DSDD, or 3 SSSD)
The Star Trek Album (SSSD)
The South Pacific Soundtrack I w/Speech (SSSD)
The Best of Patsy Cline w/Speech (SSSD)

Coming Soon Projects:

Wagner Lyrics for the TI Tenor
Oklahoma!
Porgy and Bess
Carousel
Richie Wagner's Greatest Hits II
The Music Man
Patsy Cline II
South Pacific II
The Wizard of Oz
Hits from the 50's
The 1988 Girlie Calendar
The Star Wars Saga

Thanks for your time! — Ken Gilliland (818) 507-6219

Disk 85. Catalog Library

Version: 1.5 *Author*: Marty Kroll
Requires: XB, EA *Language*: AL *Updated*: 08/16/87

This program lets you maintain a catalog of up to 900 files on up to 123 disks. They can be easily searched, sorted, and updated, and reports can be printed in a variety of formats. Contains special versions for Okidata and Epson printers. Complete documentation included

```
dskdir. v2.0. 12-dec-96
Disk name               = CATLIB
Sectors total           = 360
Sectors used            = 337
Sectors available       = 21
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CATLIB/CAT    8 DIS/VAR 80 Y >022 007
002 >003 CATLIB/DOC  105 DIS/VAR 80 Y >029 104
003 >004 CATLIB/EPS   33 PROGRAM    Y >091 032
004 >005 CATLIB/EPT   11 PROGRAM    Y >0b1 010
005 >006 CATLIB/EPU   28 PROGRAM    Y >0bb 027
006 >007 CATLIB/OKD   33 PROGRAM    Y >0d6 032
007 >008 CATLIB/OKE   11 PROGRAM    Y >0f6 010
008 >009 CATLIB/OKF   28 PROGRAM    Y >100 027
009 >00a CATLIB/XXX   33 PROGRAM    Y >11b 032
010 >00b CATLIB/XXY   11 PROGRAM    Y >13b 010
011 >00c CATLIB/XXZ   28 PROGRAM    Y >145 027
012 >00f DISKDATA01    1 DIS/FIX 24   No data sectors in file
013 >00e FILEDATA01    1 DIS/FIX 24   No data sectors in file
014 >00d LOAD          6 PROGRAM    Y >160 005
```

Disk 85. Contents of file CATLIB/CAT

CATALOGING LIBRARY

VERSION 1.5

June 20, 1987

```
 FILENAME  SIZE    TYPE    P                    DESCRIPTION
---------- ---- ---------- -  ------------------------------------------------

CATLIB/CAT    8 DIS/VAR 80 P   This catalog listing of files on the disk
CATLIB/DOC  105 DIS/VAR 80 P   Documentation (E/A or TIW editor prints 12 pages)
CATLIB/EPS   33 PROGRAM    P   Part 1 of program for Gemini/Epson printers
CATLIB/EPT   11 PROGRAM    P   Part 2 of program for Gemini/Epson printers
CATLIB/EPU   28 PROGRAM    P   Part 3 of program for Gemini/Epson printers
CATLIB/OKD   33 PROGRAM    P   Part 1 of program for Okidata printers
CATLIB/OKE   11 PROGRAM    P   Part 2 of program for Okidata printers
CATLIB/OKF   28 PROGRAM    P   Part 3 of program for Okidata printers
CATLIB/XXX   33 PROGRAM    P   Part 1 of program for your customized printer
CATLIB/XXY   11 PROGRAM    P   Part 2 of program for your customized printer
CATLIB/XXZ   28 PROGRAM    P   Part 3 of program for your customized printer
LOAD          6 PROGRAM    P   Extended Basic Load (Provided by Rag Software)
OTTAWA*UG    19 PROGRAM    P   Text file of how Ottawa Users Group uses CATLIB
```

CATLIB is a Trialware program by:

Marty Kroll Jr.
218 Kaplan Avenue
Pittsburgh, PA 15227

Disk 85. Contents of file CATLIB/DOC

CATALOGING LIBRARY

Version 1.5 06/20/87

A Trialware program by Marty Kroll Jr.

NEW FEATURES OF VERSION 1.5

1. Global string search routines for disk and file names.
2. If disk to be added is already catalogued, you can type in replacement name for use in Catlib.
3. Merging of 2 or more smaller data files is now possible.
4. Faster sorting routines.
5. Catlib is now in program image format, providing faster loading and conserving disk space.
6. Includes an Extended Basic loader (compliments of RAG software).
7. Improved keyboard entry routines.
8. Ability to produce printouts in either 6 or 8 lines/inch format.
9. A much improved documentation file.

FEATURES OF CATLIB

1. Catalogs up to 123 disks & 900 files on each set of data files.
2. Saves data for later listings, additions, or deletions.
3. Merge smaller data files into memory and resave as larger data file.
4. Fast search routines. Search for either full file or disk name, or search for a string within the file or disk name.
5. Works on single or multiple disk systems.
6. For single disk systems there is no need to continually switch disks until all deletions and additions are made.
7. For multiple disk systems you can catalog a disk from any drive.
8. When adding disks, catalog is listed on the screen. You have the option of adding the disk or not.
9. When adding disks, you are informed if the diskname is already on file. If it is, you can either replace the old listing with the new (in the case of an update, etc.), or you can enter a temporary name for the disk to be used in the listing (As in case of a backup disk, etc).
10. Catalogs most all "funny sectored" disks.
11. Eliminates all non-printable characters from file and disk names, replacing them with a period, since no legal name contains a period. This eliminates sending unwanted control codes to your printer.
12. Print a standard format catalog of any disk on file in your library, including the funny sectored disks.

13. Outputs the following to screen or printer:
    - Summary of disks
    - Complete listing of files
    - Conventional catalog listing of any disk(s) on file.
14. Output complete catalog of all disks, disk by disk, either for all disks or selectively, to the printer.
15. Chose 1,2, or 3 columns for printer outputs.

SYSTEM REQUIREMENTS

CATLIB requires:

1. 99/4A System
2. Editor/Assembler, Extended Basic, or TI-Writer Cartridge.
3. Minimum one disk drive and controller
4. Memory expansion
5. * Printer recommended

FILES ON THE CATLIB DISK

The following files should be found on each distributed copy of the CATLIB disk:

```
CATLIB/CAT    8 DIS/VAR 80 P   This catalog listing of files on the disk
CATLIB/DOC  105 DIS/VAR 80 P  Documentation (E/A or TIW editor prints 12 pages)
CATLIB/EPS   33 PROGRAM    P   Part 1 of program for Gemini/Epson printers
CATLIB/EPT   11 PROGRAM    P   Part 2 of program for Gemini/Epson printers
CATLIB/EPU   28 PROGRAM    P   Part 3 of program for Gemini/Epson printers
CATLIB/OKD   33 PROGRAM    P   Part 1 of program for Okidata printers
CATLIB/OKE   11 PROGRAM    P   Part 2 of program for Okidata printers
CATLIB/OKF   28 PROGRAM    P   Part 3 of program for Okidata printers
CATLIB/XXX   33 PROGRAM    P   Part 1 of program for your customized printer
CATLIB/XXY   11 PROGRAM    P   Part 2 of program for your customized printer
CATLIB/XXZ   28 PROGRAM    P   Part 3 of program for your customized printer
LOAD          6 PROGRAM    P   Extended Basic Load (Provided by Rag Software)
OTTAWA*UG    19 PROGRAM    P   Text file of how Ottawa Users Group uses CATLIB
```

LOADING THE PROGRAM

From Editor/Assembler:
1. Place CATLIB disk in drive #1
2. Select option 5 (Run Program File)
3. Enter disk file name (CATLIB/EPS or CATLIB/OKD or CATLIB/XXX)

From Extended Basic:
1. Place CATLIB disk in drive #1
2. Select Extended Basic
3. "Load" program auto loads and runs
4. Select you printer from the menu.

From TI-Writer:

1. Place CATLIB disk in drive #1
2. Select option #3
3. Enter disk file name (CATLIB/EPS or CATLIB/OKD or CATLIB/XXX)

RUNNING THE CATLIB PROGRAM

After viewing title screen, press any key to continue with Catlib.

You will now be asked to enter the name for the files to read and/or store the file and disk data. Enter a 2-digit decimal number in the range of 00 to 99. You will enter the number for the filedata name, and the same number will be included in the diskdata name.

The first time you select a particular set of data files, 2 files will be opened on DSK1. Thereafter, each time you select the same file names for data, the 2 files on DSK1 will be read into memory.

Note: This selection is made upon entry into the program, and can also be made thru menu selection anytime while running the program.

Note: To get the maximum number of data files on a disk, either use a blank data disk, or make a duplicate master disk & delete all files not needed to run the program, such as CATLIB/DOC, and printer versions of Catlib you will not be using.

PLEASE: Before deleting any files, please copy entire disk, and use this complete version to pass on to others. Thank you!

HINT: Divide your disk library into categories, such as games, utilities, etc. Use one set of data files for each category, such as 01 for games, 02 for utilities. These numbers and categories can then be included when entering the date, giving a small meaningful title to your printout, such as: 01*GAMES 12-15-85

These smaller files can later be merged into a larger file, as shown later in the docs. REMEMBER: Do not get rid of the smaller files, They may still be useful for printouts and listings of individual categories of you disk library.

MAIN MENU FUNCTIONS

ADDING DISKS - MENU A

After listing disk & contents on screen choose to add the disk or not. Y or N If you pick Y, program searches for identical diskname already on catalog. If match is found you can:

> Replace old listing with new updated version, by pressing enter when prompted for a temporary name.
>
> Or give new listing temporary name by keying in the name you choose.
>
> Another search is made for a match to the temporary name. If it is found the above procedure is repeated.

Hint: If you attempt to add a disk that will take the totals over the maximum limit you will be given the catalog filled message. If you have deleted some disks from the catalog and have not yet sorted them, select option "M" to sort. This may free some room for more additions.

Hint: If you would like a disk entered into the catalog with a listed name different from the actual disk name, catalog the disk twice. On the second addition, you will be given the opportunity to enter a temporary disk name. Do this, and then delete the original disk name with menu option "B".

DELETE DISKS - MENU B

You can delete a disk from the listing. You will be asked for diskname. Type diskname only, NOT DSK1. If disk is in catalog you will be notified of its deletion. If it is not in catalog, you will be told this also.

LIST DISK SUMMARY - MENU C

Lists on screen, in alphabetical order, each disk, along with sectors used and sectors available, and number of programs on the disk.

LIST ALL PROGRAMS - MENU D

Lists on screen all files in the catalog, along with file type, size, protection status and diskname.

SEARCH FOR AND LIST A DISK - MENU E

Upon selection, you will be prompted to enter the string to search for.

You are next prompted as to whether you want a "F"ull name match or a "G"lobal string search. "F"ull name match is the default.

Selecting Full name match only matches the diskname that is the exact match of the string entered. In selecting Global string match a search is made for the string in any part of the disk name.

When a match is found, the full disk name is displayed on the screen. You are asked if this is the disk you are looking for. Entering "Y" will display the disk with all its files. Entering "N" will continue the search for another match.

SEARCH FOR AND LIST A FILE - MENU F

Upon selection, you will be prompted to enter the string to search for.

You are next prompted as to whether you want a "F"ull name match or a "G"lobal string search. "F"ull name match is the default.

Selecting Full name match only matches those filenames that are the exact match of the string entered. In selecting Global string match, a search is made for the string in any part of the file name.

All files that match the string are scrolled on the screen, along with the file type, size, protection status, and diskname.

PRINT CATALOG DISK BY DISK - MENU G

After picking this menu choice, you have the option of printing all disks or selective disks. If you choose selective, the disknames are displayed on the screen, one at a time, and you are prompted as to whether you want to print it or not.

Whether you choose printing all disks, or selective disks, the date and page number are centered on each page. The diskname, sectors used and free are printed next. Then the programs on the disk are printed, in the number of columns you selected with the printing option menu. If you previously selected 3 columns, the printout will actually be made in four columns, to balance out the paper.

Before another disk is printed, the program determines whether the listing will fit on the page. If not, the paper is advanced, then the disk and its contents are printed. This eliminates a disk and contents being split between 2 pages.

PRINT DISK SUMMARY - MENU H

Prints to printer, using the default device and format or that selected by changing the printer options. Prints Date/title, page number, followed by all the disks, number of sectors used and available, and number of programs on the disk.

PRINT ALL PROGRAMS - MENU I

Selecting this option prints using the default device and format, or that selected by changing the printer options. Prints all programs, 50 per column per page using 1/6 inch spacing, or 70 per column per page using 1/8 inch spacing. The program name is printed, along with the file type, size, protection status, and disk name.

SEARCH FOR AND PRINT A DISK - MENU J

This is the same as Search for and list a disk, except that output is to printer or disk instead of screen.

CHANGE FOREGROUND/BACKGROUND COLORS - MENU K

This option allows you to change the foreground and background colors of the screen. For your convenience, the same colors cannot be selected for both foreground and background. Also, transparent is not permitted for either choice.

CHANGE PRINTER OPTIONS - MENU L

You are first prompted to enter the output device name. This can be either a printer or a disk file.

Next you are asked if you want to use condensed print (or any control codes). Use this routine to print control codes (ex. condensed print for 3 output columns). If you want to send control codes, enter "Y" when asked if using condensed print. Then enter up to 4 HEX control codes. If you don't need to use all 4 values, just enter >20 for any entries not required.

Next enter up to 4 hex codes to return the printer to normal printing, again entering >20 for values that are not required on your printer.

Next you are asked if you want to printout with 8 lines or 6 lines per inch. If you enter "6" no further information will be requested. Entering "8" will next bring up the prompt to enter up to 4 HEX control codes to set your printer up for 1/8 inch line spacing . If you don't need to use all 4 values, just enter >20 for any entries not required.

Next enter up to 4 hex codes to return the printer to normal printing, (1/6 inch line spacing), again entering >20 for values that are not required on your printer.

Next you are asked to enter the number of columns for your output. Enter a number between one and three. Note that three columns are not accepted unless you have chosen to use condensed print. *Note*: Entering 3 columns and condensed print will cause the "Print all Disks" output to be in 4 instead of 3 columns, to balance out the paper.

Next, you can enter a date/title for your printouts.

SORT AND SAVE DATA - MENU M

This routine alphabetizes all files and disks. Then all data is saved to disk.

You are first reminded to put your data disk in drive number 1. Sorting then begins, after which data is written to disk.

Sorting and saving may take from 2 to 7 minutes total.

An audible chime will notify you when this process is complete. Then press any key to continue.

Note: If, while writing the data to disk, you get an error, as when a disk gets filled up, or if no disk is in drive #1, do not worry. This error is not fatal. Just press **ENTER**, and upon returning to the main menu, select this option once again. Place a disk with sufficient space left in drive #1, press **ENTER**, and the sorting and saving will begin again.

Note: To find out how much room is required on the disk for the save routine, just total the number of files and disks in the catalog, and divide by 10. Adding 2 to your answer will give you the approximate number of sectors required to save your data.

MERGE DATA FILES INTO MEMORY - MENU N

This option allows you merge smaller data files into memory.

You are first prompted to select the name of the file to merge. This file is then merged into memory. The process can be repeated until the maximum number of files or disks has been reached.

Note: If you try to merge a set of files that will cause the total files or disks to exceed the maximum, the load will be aborted, an error message will be displayed, and you will be returned to the main menu.

Hint: After merging your files, select this option one more time. Choose a set of data files that have not been used yet. The program will attempt to merge these files, but since they are empty, no loading takes place. The reason for doing this is that, once the files in memory are sorted, they are saved to disk using the names of the last files selected. By choosing an empty set of files as your last choice, the new data will be resaved, while keeping the original smaller files intact.

Hint: When 2 or more files are merged, the possibility exists that 2 disks with the identical name may be listed. Catlib is not able to determine which files belong to which disk.

To prevent any problems that may arise from this situation: After merging files, either list to screen or printer a disk summary of the catalog. Scan the list for identical disk names. If found, select option "B" and delete the disk name. This will delete all disks with the name, as well as all files.

You can then go back and re-enter the duplicate disks, this time having the opportunity to give one or more of the disks a temporary name to be used in the Catlib program.

LOAD ANOTHER SET OF DATA FILES - MENU O

Use this option to start another session of cataloging or updating a file. With this option, before the files are loaded, all data in memory is completely erased. DO NOT use this routine to merge files into memory, as memory is completely empty.

TERMINATE PROGRAM - MENU P

Selecting this will return you to the main title screen.

ODDS AND ENDS

HALTING/PAUSING OUTPUT

Pressing function 4 will halt all output, both to screen or printer.

Pressing any other key will temporary pause the screen scroll. To continue screen scrolling, just press another key.

CHANGING DEFAULTS

Use of a disk sector editor, such as Diskfixer, Disko, or Disk+Aid is required to make permanent changes to the CATLIB defaults.

First, determine which set of Catlib files you are going to change, depending on the printer you have. Then determine the third file of that set. This file should be one of the following: CATLIB/EPU or CATLIB/OKF or CATLIB/XXZ.

Using the sector editor, find the first data sector for the above file. The bytes of the file should resemble the following:

```
    | 00   02   04   06   08   0A   0C   0E      0123456789ABCDEF
----|---------------------------------------   ----------------
00  | 0000 1A34 2000 0717 07F6 0003 FFFF 0F20    \\\4 \\\\\\\\\\
10  | 2020 3E30 4620 3E32 3020 3E32 3020 3E32      >0F >20 >20 >2
20  | 3020 1220 2020 3E31 3220 3E32 3020 3E32    0 \   >12 >20 >2
30  | 3020 3E32 3020 0000 1B30 2020 3E31 4220    0 >20 \\\0  >1B
40  | 3E33 3020 3E32 3020 3E32 3020 1B32 2020    >30 >20 >20 \2
50  | 3E31 4220 3E33 3220 3E32 3020 3E32 3020    >1B >32 >20 >20
60  | 0032 0012 1060 5050 0000 0003 5049 4F20    \2\\\`PP\\\\PIO
70  | 2020 2020 2020 2020 2020 2020 2020 2020
80  | 2020 2020 2020 2020 2020 2020 2020 2020
90  | 2020 2020 0000 0000 0000 0000 0000 0000        \\\\\\\\\\\\
A0  | 0000 0000 0000 0000 0000 0000 0000 0000    \\\\\\\\\\\\\\\\
B0  | 0000 0000 208E 20B2 06A0 2108 D82D 0002    \\\\ \ \\\!\\-\\
C0  | 8C00 0380 208E 20C2 06A0 2108 D831 8C00    \\\\ \ \\\!\\1\\
D0  | 0602 16FC 0380 208E 20D4 06A0 210E DB60    \\\\\\ \ \\\!\\`
E0  | 8800 0002 0380 208E 20E4 06A0 210E DC60    \\\\\\ \ \\\!\\`
F0  | 8800 0602 16FC 0380 208E 20F6 C05D D82D    \\\\\\\\ \ \\]\-
```

These bytes on this sector contain the following information, which can be changed with the sector editor:

FOREGROUND/BACKGROUND COLORS:
Byte 0007
Default = 17 = black on cyan

FOREGROUND/BACKGROUND COLORS OF ERROR SUBROUTINE:
Byte 0009
Default = F6 = white on dark red

NUMBER OF COLUMNS FOR PRINTOUT
Byte 000B = 03
Default = 3

FLAG FOR CONDENSED PRINT:
Byte 000C,000D
Default = FF FF = condensed print
00 00 = normal print

CONTROL CODES FOR CONDENSED PRINT:
Bytes 000E - 0011 (HEX CODES)
Epson = 0F 20 20 20 = CHR$(15) & 3 spaces
Okidata = 1D 20 20 20 = CHR$(29) & 3 spaces

Bytes 0012 - 0021 (ASCII REPRESENTATION OF HEX CODES, INCLUDING SPACES AND HEXADECIMAL SIGN [>])
Epson = 3E 30 46 20 3E 32 30 20 = '>0F >20 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '
Okidata = 3E 31 44 20 3E 32 30 20 = '>1D >20 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '

CONTROL CODES FOR NORMAL PRINT:
Bytes 0022 - 0025 (HEX CODES)
Epson = 12 20 20 20 = CHR$(18) & 3 spaces
Okidata = 1E 20 20 20 = CHR$(30) & 3 spaces

Bytes 0026 - 0035 (ASCII REPRESENTATION OF HEX CODES, INCLUDING SPACES AND HEXADECIMAL SIGN [>])
Epson = 3E 31 32 20 3E 32 30 20 = '>12 >20 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '
Okidata = 3E 31 45 20 3E 32 30 20 = '>1E >20 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '

FLAG FOR LINE SPACING:
Byte 0036,0037
Default = 00 00 = 6 Lines per inch
FF FF = 8 Lines per inch

CONTROL CODES FOR 1/8 INCH LINE SPACING:
Bytes 0038 - 003B (HEX CODES)
Epson = 1B 30 20 20 = CHR$(27) & CHR$(48) & 2 spaces
Okidata = 1B 38 20 20 = CHR$(27) & CHR$(56) & 2 spaces

Bytes 003C - 004B (ASCII REPRESENTATION OF HEX CODES, INCLUDING SPACES AND HEXADECIMAL SIGN [>])
Epson = 3E 31 42 20 3E 33 30 20 = '>1B >30 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '
Okidata = 3E 31 42 20 3E 33 38 20 = '>1B >38 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '

CONTROL CODES FOR 1/6 INCH LINE SPACING:
Bytes 004C - 004F (HEX CODES)
Epson = 1B 32 20 20 = CHR$(27) & CHR$(50) & 2 spaces
Okidata = 1B 36 20 20 = CHR$(30) & CHR$(54) & 2 spaces

Bytes 0050 - 005F (ASCII REPRESENTATION OF HEX CODES, INCLUDING SPACES AND HEXADECIMAL SIGN [>])
Epson = 3E 31 42 20 3E 33 32 20 = '>1B >32 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '
Okidata = 3E 31 42 20 3E 33 36 20 = '>1B >36 '
3E 32 30 20 3E 32 30 20 = '>20 >20 '

OUTPUT DEVICE
Bytes 006B (NAME LENGTH)
Default = 03
Bytes 006C - 0093 (DEVICE NAME)
Default = 50 49 4F 20 20 20 20 (etc)
= 'PIO ' etc (40 chars maximum)

COMING SOON!!

CATLIB COMPANION is nearing completion. This program will enable you to use your CATLIB files, and add extra comments to them, such as language, modules required, plus additional comments about printer and hardware requirements, source of program, etc. When CATLIB files are updated, CATLIB COMPANION comment files can be automatically updated as well.

Of course, various search routines, as well as a variety of printer outputs will be available in CATLIB COMPANION.

This program will be very useful to both the individual as well as the User Group, for having on disk all needed information about the disks and files in their library. It will be released as Trialware.

Any comments or suggestions will be welcome. If you Users Group uses CATLIB to organize their library, why not let me know how, and I will pass it along to others.

Disk 86. XB Tools

Version: *Author*: Many
Requires: *Language*: *Updated*: 08/17/87

A collection of utility routines for the Extended BASIC programmer. Includes routines to change variable names, compress and uncompress program lines, analyze programs, debugging aids, and much more. Includes works by Barry Traver, Jim Swedlow, Mike Dodd, Greg Wonderly and others.

```
dskdir. v2.0. 12-dec-96
Disk name               = XB/TOOLS
Sectors total           = 360
Sectors used            = 333
Sectors available       = 25
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       4 DIS/VAR 80   >022 003
002 >003 ASCHART      20 PROGRAM      >025 019
003 >004 CHGE/TO/0     4 PROGRAM    Y >038 003
004 >005 COMPRESS     27 PROGRAM      >03b 026
005 >006 DATAPRESS     7 PROGRAM      >055 006
006 >007 DISPLAY/AT   31 PROGRAM    Y >05b 030
007 >008 FP           14 PROGRAM      >079 013
008 >009 FP/DOCS      19 DIS/VAR 80   >086 018
009 >00a LINEMOVE     15 PROGRAM      >098 014
010 >00b MENGEN/UTL   28 PROGRAM    Y >0a6 027
011 >00c MERGE/READ   15 PROGRAM    Y >0c1 014
012 >00d MERGEDITOR    7 PROGRAM    Y >0cf 006
013 >00e MERGETABLE    7 DIS/VAR 80   >0d5 006
014 >00f MUSIC         2 DIS/VAR163   >0db 001
015 >010 MUSICCODER    8 PROGRAM    Y >0dc 007
016 >011 NAMECHANGE    9 PROGRAM      >0e3 008
017 >012 PRINT*TEST    6 PROGRAM      >0eb 005
018 >013 PRINTER       8 PROGRAM      >0f0 007
019 >014 PROG/CHKR    12 PROGRAM    Y >0f7 011
020 >015 PROGRM/MKR   17 PROGRAM    Y >102 016
021 >016 REMDIVIDER    4 PROGRAM    Y >112 003
022 >017 SPRITCODER    7 PROGRAM    Y >115 006
023 >018 SPRITE/AID   32 PROGRAM    Y >11b 031
024 >019 TOKEN/READ   22 DIS/VAR163 Y >13a 021
025 >01a TRACE         5 PROGRAM      >14f 004
026 >01b UNCOMBINER    3 PROGRAM    Y >153 002
```

Disk 86. Contents of file -README

Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

Public Domain Disk #86

This disk contains programs by 5 different authors. There is very little documentation on the programs. In most cases you must list the program and read the comments to get the details. Below is a list of the files on the disk, arranged by author.

Jim Swedlow
- ASCHART
- COMPRESS
- DATAPRESS
- LINEMOVE
- NAMECHANGE
- PRINT*TEST
- PRINTER

Barry Traver
- CHGE/TO/0
- DISPLAY/AT
- MERGE/READ
- MERGEDITOR
- MERGETABLE
- MUSIC
- MUSICCODER
- PROG/CHKR
- PROGRM/MKR
- REMDIVIDER
- SPRITCODER
- SPRITE/AID
- TOKEN/READ
- UNCOMBINER

Greg Wonderly
- MENGEN/UTL

Mike Dodd
- TRACE

Ulrich Dillge
FP
FP/DOC

Disk 86. Contents of file FP/DOCS

LISTING-FORMATTER

1. REQUIREMENTS : Extended Basic, SS/SD Disk System, TI-Impact Printer (EPSON MX-80)

2. GENERAL PROGRAM DISCUSSION

The purpose of the 'LISTING-FORMATTER' (FP) is to generate a program listing that is easier to read than the one generated by TI-Extended Basic. Here are some of the faults that I found with the Extended Basic command 'LIST' and the program listing:

a. Automatic 'Wrap-around' if a program line exceeds 80 characters. The extension of the program line starts in column #1 on the next line. There is no clear distinction between two consecutive program lines.

b. The left margin is column #1. This eliminates punching holes for three ring binders and keeping neat files of programs.

c. Every program listing is printed in the 80 characters/line (normal print) mode. If condensed print was available it could save a considerable amount of paper.

d. Program name, disk name and date would have to be included in the program if they are desired in your listing. They take up unnecessary memory space.

e. The pages are neither numbered nor labeled with a name. Have you ever tried to put the pages back into the correct order ?

The 'LISTING-FORMATTER' eliminates these faults. 'FP' allows the user to set left and right margins. You may use the defaults of 4 and 80 or type in the desired margins. If you choose a right margin of greater than 80 you by definition select condensed print. The program will not allow you to waste a lot of paper by selecting absurd margins.

Left Margin :	1 to 10
Right Margin (normal):	70 to 80
Right Margin (condensed):	122 to 132
Page length :	60 lines

You must turn off 'Automatic perforation skip-over.

'FP' does not have to reside on the same diskette as your program that you want a listing of. After loading 'FP' you may remove the diskette from the drive.

As with any program there are always trade-offs and disadvantages. 'FP' is no exception. Here are some of them that will help 'FP' to run without faults :

a. The program should be RESequenced' before printing it through 'FP'. Using the line numbers 1 - 9 should be avoided and the line number increment should be 10.

b. The last program line should be 'END'. This serves two purposes. You can very easy determine if your entire program was printed and 'FP' likes to finish with a program line less than 80 characters long. This 'bug' will be eliminated in the next version.

c. The program must be 'LISTed' to disk with a DIFFERENT NAME than the program file itself. REMEMBER, TI-Extended Basic does not check for existing file names, it will unconditionally use the file name you specified. You can loose the entire program if you DO NOT change the file name.

3. HOW TO USE 'FORM-PRINT'

3.1 Load program into memory :

```
OLD DSKx.yyyyy
```

3.2 Insert as last line : 32000 END

3.3 RESequence the program : RES

3.4 Save it on disk using the LIST command : LIST "DSKx.zzzzz"

3.5 Load and run 'FP' : RUN "DSKx.FP"

3.6 Type in your response to the questions, or just hit **ENTER**.

4. Since 'FP' was written to be used with the TI-IMPACT Printer (EPSON MX-80), you may want to make changes to accommodate your printer :

```
170 DISPLAY AT(14,2):"OUTPUT   :
RS232.BA=4800" *** Displays
printer default ***

200 IF K$="" THEN
K$="RS232.BA=4800" *** Sets
printer default to RS232 at 4800
baud

520 CHR$(27);"A";CHR$(12) *** Set
line spacing to 12/72 inch
```

```
520 CHR$(27);"E"          ***
Turn 'ON' emphasized print

520 CHR$(27);"G"          ***
Turn 'ON' double printing

530 CHR$(14)              ***
Turn 'ON' enlarged printing

530 CHR$(20)              ***
Turn 'OFF' enlarged printing

560 CHR$(27);"H"          ***
Turn 'OFF' double printing

560 CHR$(27);"F"          ***
Turn 'OFF' emphasized printing

570 CHR$(15)              ***
Turn 'ON' condensed printing

680 CHR$(18)              ***
Turn 'OFF' condensed printing
```

Disk 86. Contents of file MERGETABLE

```
129 ELSE
130 ::
131 !
132 IF
133 GO
134 GOTO
135 GOSUB
136 RETURN
137 DEF
138 DIM
139 END
140 FOR
141 LET
142 BREAK
143 UNBREAK
144 TRACE
145 UNTRACE
146 INPUT
147 DATA
148 RESTORE
149 RANDOMIZE
150 NEXT
151 READ
152 STOP
153 DELETE
154 REM
155 ON
156 PRINT
157 CALL
158 OPTION
159 OPEN
160 CLOSE
161 SUB
162 DISPLAY
163 IMAGE
164 ACCEPT
165 ERROR
166 WARNING
167 SUBEXIT
168 SUBEND
169 RUN
170 LINPUT
171 OuNj^/w7(Nj}V\w}wN(<}7?n<K/;_u(~^g}<.m
172 Fr=p@R,4R,R
173 P<Q=f4!4PV3sHKS4@B
174  HD&,EF`VB3D.Td
175 od=.0a>gg\g
176 THEN
```

```
177 TO
178 STEP
179 ,
180 ;
181 :
182 )
183 (
184 &
185 5\JZR)0
186 OR
187 AND
188 XOR
189 NOT
190 =
191 <
192 >
193 +
194 -
195 *
196 /
197 ^
198 q(Vf[JB0Dq
199 "" 1.MERGETABLE
200 1.MERGETABLE
201 49 .MERGETABLE
202 EOF
203 ABS
204 ATN
205 COS
206 EXP
207 INT
208 LOG
209 SGN
210 SIN
211 SQR
212 TAN
213 LEN
214 CHR$
215 RND
216 SEG$
217 POS
218 VAL
219 STR$
220 ASC
221 PI
222 REC
223 MAX
224 MIN
225 RPT$
226 "Z M0>sM>"
227 LEn$MqN`nem
```

```
228 DmG uMM?
229 TTM|ZJILJ<KJM|=<iOT`iL=0I;TV`Ie`i
230  =_`^//<_`
231 b<\J\0UF\9V
232 NUMERIC
233 DIGIT
234 UALPHA
235 SIZE
236 ALL
237 USING
238 BEEP
239 ERASE
240 AT
241 BASE
242 J\AUF\Fu&\0F\V
243 VARIABLE
244 RELATIVE
245 INTERNAL
246 SEQUENTIAL
247 OUTPUT
248 UPDATE
249 APPEND
250 FIXED
251 PERMANENT
252 TAB
253 #
254 VALIDATE
```

Disk 87. Disk Utilities

Version: 4.0a *Author*: John Birdwell
Requires: XB, EA *Language*: AL *Updated*: 1/05/88

This fairware program is one of the most comprehensive disk sector editor programs available. It includes the ability to sector edit files, a disk manager, the ability to add comments to a file description, and many more powerful, useful, and relatively easy to use features.

```
dskdir. v2.0. 12-dec-96
Disk name               = BIRDWELL
Sectors total           = 360
Sectors used            = 281
Sectors available       = 77
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHARA1        5 PROGRAM      >022 004
002 >009 DSKU/DOC     96 DIS/VAR 80   >0c3 095
003 >00b DSKU/NEW     14 DIS/VAR 80   >124 013
004 >003 DSKU/O-SC    59 DIS/FIX 80   >026 058
005 >004 DSKU1        33 PROGRAM      >060 032
006 >005 DSKU1/LOAD    5 DIS/FIX 80   >080 004
007 >006 DSKU2        16 PROGRAM      >084 015
008 >007 DSKUD1       33 PROGRAM      >093 032
009 >008 DSKUD2       17 PROGRAM      >0b3 016
010 >00a LOAD          3 PROGRAM      >122 002
```

Disk 87. Contents of file DSKU/DOC

DISK UTILITITES
VERSION 3.3
by JOHN BIRDWELL
7052 Springhill Circle
Eden Praire, MN 55344

This program is made available as FAIRWARE. If you use it, please send $10.00 to me for my effort.

In addition to the program image version, it is also available for the SUPER-CART, but only as V3.0 and due to the memory restrictions of the SC some of the capabilities listed below could not be included.

NOTE: Added features release 3.3.

- A file copy ability has been added.
- Allows you to change the name of the file when you copy it.
- Allows you to override file protect of the destination file.
- Checks destination for space availability before starting copy.
- Moves comment line to destination file.
- Improvements to the Sector Editor.
- Current mode of operation is displayed in the top right corner of screen.
- Flashing cursor.
- As changes are made they are displayed in inverse video. This should prevent you from accidentally changing data. NOTE: Switching from HEX to ASCII or ASCII to HEX will remove inverse characters.
- Fragmented files are indicated by an asterisk (*) after the file size in the Directory/Comments option.
- Corrects a bug in file operations for files over 256 in length.
- Includes a demonstration version of this program that is the only one which I would like to see posted to any boards. This is for those who would like to see this FAIRWARE program but not pay for its use. Your cooperation with this is appreciated.

NOTE : Added features release 3.2-2.

- Find String length increased to 16 characters
- Spaces can now be included in ASCII Find String
- Current byte location displayed for Find String and Edit

A. REQUIREMENTS

TI-99/4A
32K Memory Expansion
1 Diskette Drive (minimum)
Extended Basic, TI-Writer or Editor/Assembler
CHARA1 file on DSK1 for true lower case

B. LOADING THE PROGRAM

From TI-Writer :
1. Place the disk in drive 1
2. Select option 3
3. Enter the name DSKU1
(*Note*: DSKU1 and DSKU2 files must be on the disk)

From EXTENDED BASIC :
1. Place the disk in drive 1
2. Select Extended Basic
(*Note*: LOAD, DSKU1/LOAD, DSKU1 & DSKU2 must be on the disk)

From Editor/Assembler :
1. Select option 5
2. Enter DSKn.DSKU1 (n=Drive number)
(*Note*: DSKU1 and DSKU2 files must be on the disk)

C. PURPOSE OF DSKU

These utilities provide the user with the means to study how data is stored on disk. Experienced users will find them useful for changing, i.e., editing, the data to suit their purposes.

D. MAIN MENU SELECTIONS

1 Compare Disks
Compare two diskettes sector by sector. If two sectors do not match they will be sent to the selected print device.

2 Print Sectors
Print selected single or multiple sectors.

3 Sector Editor
Edit a sector in ASCII or hex. Write edited sector back to same or another sector/disk.

4 Find String
Locate a string (up to 16 bytes, ASCII or hex). If found it will be displayed and SECTOR EDITOR can be entered.

5 Disk Report
List disk contents and detailed file information to output device.

6 Directory/Comments
Same as Disk Report, but listed to the display (Not available on SUPER-CART version).

7 File Utilities
** Sub menu - see below

8 Printer Setup
Set output device (printer or disk), disable Form Feed if desired, select standard (pica) or condensed modes for print utilities.

9 Screen Colors
Change text and background colors

** FILE UTILITIES

1 File Compare
Compare two files with same or different names located on same or separate disks. Sector which do not match are sent to output device.

2 File Print
Print all sectors of a file.

3 File Edit
Edit a file, the program determines beginning and ending sectors.

4 Find String
Locate a string (up to 16 bytes, ASCII or hex) within a file. If found the editor may be entered.

5 File Report
Display detailed file information.
(*Note*: This is the only FILE UTILITY available with the SUPER-CART version).

6 File Copy
Copies a single file. Destination can carry same or a different name. Comment line is carried to the destination.

All menu selections are described in detail on the following pages.

E. KEY ASSIGNMENTS

(*Note*: All 1-character entries do not require **ENTER** to be pressed)

1. ALL MODES EXCEPT EDITOR

FUNCTION
1 - Delete character
2 - Insert character
3 - Erase field
4 - Stop printing and/or return to Menu
8 - Return to currently selected function
+ - Quit, return to TI title screen
D - Move cursor right (arrow)
S - Move cursor left (arrow)

<ENTER> - Terminate input

CONTROL
P - Print copy of screen (screen dump)

2. SECTOR/FILE EDITOR

CONTROL
N - Read (N)ext sector (or use **ENTER**)
B - Go (B)ack one sector
W - (W)rite sector to disk (same or new location)
R - (R)ead a new drive/sector
P - (P)rint copy of screen (screen dump)
D - (D)one, return to beginning of current selection
8 - Same as 'D' above
A - Shift to ASCII display
H - Shift to HEX display

FUNCTION
4 - Return to main menu
D - Move cursor right (arrow)
S - Move cursor left (arrow)
E - Move cursor up (arrow)
X - Move cursor down (arrow)

F. COMPARE DISKS

This option will compare the contents of two diskettes over a selected range of sectors. Each pair of non-matching sectors is sent to the output device specified by option 8 - Printer Setup. Printed sectors are identified (drive and sector number).

G. PRINT SECTORS

Selected sector(s) is/are sent to the output device. Hex display is on the left, ASCII on the right. (See O - USER NOTES for correct settings for output device name length byte to enable output to a serial printer or to disk.)

H. SECTOR EDITOR

Shown below is a typical example of the display obtained with the SECTOR EDITOR in its normal hex mode:

```
Sector Editor EDIT
======================================
0100 0020 2020 2020 2020 2039 6000 42AA
0142 0000 4200 0042 600C 4200 0042 0000
4200 0042 6022 4211 4442 4953 424B 2042
5554 4249 4C42 4954 4259 2042 5633 422E
3042 0460 4265 AC42 0016 4211 0046 2020
3960 2A42 8484 4200 0042 0003 4250 4942
4F20 4220 2042 2020 4220 2042 2020 4220
2042 2020 4220 2042 2020 4220 2042 2020
4220 2042 2020 421D 0042 1E00 420C F542
0000 4205 0D42 07FA 42FF 0046 2020 2020
3960 5A42 0800 4200 0B42 4453 424B 3142
2E43 4248 4142 5241 4231 0042 6740 4268
3642 6870 426B AA42 6E5A 4270 6042 72C6
4273 5042 7D9C 4208 0042 6AFA 4209 0042
6AC0 420A 0042 6A96 420B 0046 2020 2020
4E44 204F 4620 4649 4C45 FF00 0000 0000

Drive 2 Sector # 23 Byte 255 >FF
```

It can be changed to an ASCII display by pressing CTRL-A, returned to hex by CTRL-H. All arrow keys are active and can be used to move the cursor to the desired byte. The current cursor position within the sector being edited is indicated in both decimal and hexadecimal. This is (Byte 255 >FF) in the example. When editing is complete, CTRL-W(rite) will first reverse text/background colors as a warning that the sector is about to be written to the disk. It may be written back to the same location (shown at the bottom of the screen) or to a different location if desired. The write can be aborted by using FUNCT-4 or FUNCT-8. If a write to disk is made, write to disk is made, the sector bitmap will be updated accordingly.

I. FIND STRING

A string of up to 16 bytes in length (ASCII or hex) can be located anywhere on the diskette in the specified drive and over any given range of sectors. The sector with the first occurrence of it is displayed with the string highlighted. On the bottom of the screen the options E(dit), C(ontinue) and Q(uit) are displayed. E will place the program in the SECTOR EDITOR with all the features explained above. C will continue the search with the next sector while Q will return to the beginning of Find String.

J. DISK REPORT

This selection prints out a list of pertinent disk information not normally available with a disk cataloger. While the basic information, such as disk name, total and available sectors, file names, types, sizes and protection are shown as usual, the disk report also includes the location of the files (including fractured files) and comments if any (see DIRECTORY/COMMENTS).

Here is a typical example of a disk report:

```
Diskname = DSKU/V3-0 Total Sectors = 718  Available Sectors = 68   Files = 19
------------------------------------------------------------------------------
                              Sector
Filename   File Type  Size Start End  Prot Comment
------------------------------------------------------------------------------
DSKU1      Program      33 0134 0153  Yes  Disk Utility Version 3.0 Program #1
DSKU2      Program       8 0154 0159  Yes  Disk Utility Version 3.0 Program #2
                           01A0 01A0
READ_ME    Dis/Var 80   39 015A 017F  No   Directions For Using Disk Utilities
UTIL-EQU1  Dis/Var 80   33 0022 0040  Yes  Equates For Program Image
                           02B6 02B6
UTIL-EQUSC Dis/Var 80   30 01A1 01BC  Yes
                           02B7 02B7
UTIL-MAIN  Dis/Var 80   13 0041 004C  Yes  Main Routine
UTIL-SRC   Dis/Var 80    2 0132 0132  Yes  Copy Files For Program Image
UTIL-SRCSC Dis/Var 80    2 0133 0133  Yes  Copy Files For Super Cart.
UTIL/O-SC  Dis/Fix 80   59 0208 0209  Yes  Super Cart Object Code
                           027E 02B5
UTIL1      Dis/Var 80   71 004D 0092  Yes  Compare,Print Disk,Edit,Find,Report
UTIL1SC    Dis/Var 80   68 01CF 01DC  Yes  Comp.,Print,Edit,Find,Report
                           020A 023E
UTIL2      Dis/Var 80   49 0093 00C1  Yes  File Comp.,Print,Edit,Find,Rpt,Subs
                           019E 019E
UTIL2SC    Dis/Var 80   17 01BF 01CE  Yes  File Report, File Subs
UTIL3      Dis/Var 80   11 00C2 00CB  Yes  Printer Options, Screen Colors
UTIL3SC    Dis/Var 80    9 0276 027D  Yes  Printer Options, Screen Colors
UTIL4      Dis/Var 80   58 00CC 0104  Yes  Main Sub-Routines
UTIL4SC    Dis/Var 80   56 023F 0275  Yes  Main Sub-Routines
UTIL5      Dis/Var 80   46 0105 0131  Yes  Getkeys,System Support Subs,Buffers
UTIL5SC    Dis/Var 80   46 01BD 01BE  Yes  Getkeys,System Support Subs,Buffers
```

A note regarding 'Comments':

This program uses the last 34 (>22) bytes of the file descriptor record for the comment line. Normally, bytes 28 (>1C) through 255 (>FF) are reserved for the data chain pointer blocks. A file would have to be very badly fractured to need the entire 227 (>E3) bytes for this purpose, so this should not ever interfere with normal operations. However, standard disk managers do not read beyond the data pointers and thus you may lose the comments if such a disk manager is used to copy files/disks. Only a sector type disk copier will transfer the entire file descriptor record including your comments or use the file copy feature of this program.

K. DIRECTORY/COMMENTS

Except for output being directed to the display screen and the ability to add or change comments, this option is essentially the same as DISK REPORT. But since only five files and their comment lines can be shown at one time, a current/total number of pages is displayed in the upper right hand corner of the screen. Files are numbered 1 to 5 on each page. Entering a number will provide the cue for adding/changing the comment line, <Enter> will advance to the next page. An example of a typical DIRECTORY/COMMENTS screen is shown below:

```
Direct/Comment 1/4
======================================
DSKU/V3-1 Tot=718 Avail=49 Files=19
Filename Size File Type Prot
======================================
1 DSKU1 33 Program No
Disk Utility Version 3.1 Program #1

2 DSKU2 10 Program No
Disk Utility Version 3.1 Program #2

3 READ_ME 63* Dis/Var 80 No
Directions For Using Disk Utilities

4 UTIL-EQU1 33 Dis/Var 80 No
Equates For Program Image

5 UTIL-EQUSC 30 Dis/Var 80 Yes
Equates For Super Cart Version 3.0

Enter to continue or 1-5 to update
```

(This option was added at the request of Joe White, K-TOWN UG)

L. FILE UTILITIES

When this function is selected a sub menu with the following options is displayed:

1 - File Compare
2 - File Print
3 - File Edit
4 - Find String
5 - File Report (only one available with SUPER-CART)
6 - File Copy

The basic difference between these options and the ones from the main menu is that all operations are based on file name. The user does not have to enter the beginning and ending sectors of the file to be edited. The first and last sectors of a file are indicated by an SOF or EOF displayed just above the sector number. A warning 'honk' and a message are issued if the user attempts to go beyond these sectors. If the user opts to exceed the file's boundary the program automatically reverts to the SECTOR EDITOR.

M. PRINTER SET-UP

Default is PIO, and, also by default, any printer output is followed by a form feed. Printer defaults for disk report, print sector and screen dump are shown and can be changed. Settings will remain in effect as long as the program is running or until this option is used to change them. Output can be directed to a serial printer or disk drive. Before DSKU will accept a device name of more than 3 characters, the length byte must be changed (see O. USER NOTES below).

N. SCREEN COLORS

Allows changing of the default text and screen colors. The changes will remain in effect as long as the program is running. To install permanent changes, see USER NOTES below.

O. USER NOTES

These notes are furnished for those who wish to modify the program, that is, set up defaults to meet their needs.

1. Output to disk

If output of any of the print options is routed to disk (by changing the printer description via PRINTER SET-UP) the 3 print options should first be set to 'standard'. This will create a D/V 80 file which is compatible with TI-Writer. (If left in compressed mode a D/V 132 file results.) If a file with the given name already exists the new output will be appended to it, otherwise a new file will be generated. The line of these files contain control characters (for TI-Writer) and should be deleted. A possible use for this feature is to maintain a master disk catalogue which is accessible from TI-Writer.

3. Default Parameters

All defaults are located in the first sector of file DSKU1. They are explained below and shown in both program image and SUPER-CART code. For clarity the bytes not involved are indicated by '-' .

a. >03 - length of output device name
b. >5049 4F - output device name (PIO)
c. >20 - all of them up to end of 3rd line may be used for output device name provided >03 above is set accordingly
d. >1D - compressed mode (OKIDATA), use >0F for Epson
e. >1E - cancel compressed mode (OKIDATA), use >12 for Epson
f. 0C - form feed, use >00 to omit form feed
g. >F5 - color byte, F=text, 5=screen color
h. >0101 00 - control print mode for disk report, screen dump and print sector in that order, use >00 for compressed,
>01 for normal pica. (Not avaiable from SUPER-CART.)

4. Default Locations

a. Program Image sector display

```
File Editor
========================================
---- ---- ---- ---- ---- ---- ---- ----
---- --03 5049 4F20 2020 2020 2020 2020
2020 2020 2020 2020 2020 2020 2020 2020
1D00 1E00 OCF5 0101 00-- ---- ---- ----
etc etc etc
```

b. SUPER-CART sector display

```
File Editor
========================================
---- ---- ---- ---- ---- ---- ---- ----
---- ---- ---- ---- ---- ---- ---- ----
---- ---- ---- ---- ---- ---- ---- ----
---- ---- ---- ---- ---- ---- ---- ----
---- ---- ---- ---- ---- ---- ---- ----
---- ---- ---- ---- ---- --03 --50 49--
4F20 --20 20-- 2020 --20 20-- 2020 --20
20-- 2020 --20 20-- 2020 --20 20-- 2020
--20 20-- 2020 --1D ---- 1E-- --0C F5--
etc etc etc
```

P. CONCLUSION

The foregoing should provide sufficient information for using this program. I would suggest that any editing and writing be done on a backup disk. Once the original data on a disk has been overwritten it can not be recovered. I would appreciate hearing from users who discover any bugs in the program or who might have worthwhile suggestions for improvements.

Disk 87. Contents of file DSKU/NEW

DISK UTILITITES

VERSION 3.3
by JOHN BIRDWELL
7052 Springhill Circle
Eden Praire, MN 55344

This program is made available as FAIRWARE. If you use it, please send $10.00 to me for my effort.

In addition to the program image version, it is also available for the SUPER-CART, but only as V3.0 and due to the memory restrictions of the SC some of the capabilities listed below could not be included.

NOTE: Added features release 3.3.

- A file copy ability has been added.
- Allows you to change the name of the file when you copy it.
- Allows you to override file protect of the destination file.
- Checks destination for space availability before starting copy.
- Moves comment line to destination file.
- Improvements to the Sector Editor.
- Current mode of operation is displayed in the top right corner of screen.
- Flashing cursor.
- As changes are made they are displayed in inverse video. This should prevent you from accidentally changing data. NOTE: Switching from HEX to ASCII or ASCII to HEX will remove inverse characters.
- Fragmented files are indicated by an asterisk (*) after the file size in the Directory/Comments option.
- Corrects a bug in file operations for files over 256 in length.
- Includes a demonstration version of this program that is the only one which I would like to see posted to any boards. This is for those who would like to see this FAIRWARE program but not pay for its use. Your cooperation with this is appreciated.

NOTE : Added features release 3.2-2.

- Find String length increased to 16 characters
- Spaces can now be included in ASCII Find String
- Current byte location displayed for Find String and Edit

PRINT DSKU_DOCS FOR COMPLETE DOCUMENTATION

Disk 88. c Libraries

Version:
Author: Tom Bentley
Requires: c99
Language: c99
Updated: 01/05/88

A collection of library routines for c99 including dynamic memory allocation, catalog disk, load CHARA1, floating point math, and a comprehensive set of file access utilities. With source code.

```
dskdir. v2.0. 12-dec-96
Disk name               = BENTLEY
Sectors total           = 360
Sectors used            = 291
Sectors available       = 67
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 ALLOCC        6 DIS/VAR 80   >022 005
002 >003 ALLOCI        3 DIS/VAR 80   >027 002
003 >004 CDIRC        12 DIS/VAR 80   >029 011
004 >005 CHRLOAD       5 DIS/FIX 80   >034 004
005 >006 CHRLOADC      8 DIS/VAR 80   >038 007
006 >007 CTYPEC       30 DIS/VAR 80   >03f 029
007 >008 DIR;C        12 DIS/VAR 80   >05c 011
008 >009 DIR;O        16 DIS/FIX 80   >067 015
009 >00a FLOAT        10 DIS/FIX 80   >076 009
010 >00b FLOATC       31 DIS/VAR 80   >07f 030
011 >00c FLOATDOC     30 DIS/VAR 80   >09d 029
012 >00d FLOATI        2 DIS/VAR 80   >0ba 001
013 >00e TCIO         22 DIS/FIX 80   >0bb 021
014 >00f TCIOC        17 DIS/VAR 80   >0d0 016
015 >010 TCIODOC      28 DIS/VAR 80   >0e0 027
016 >011 TCIOI         6 DIS/VAR 80   >0fb 005
017 >012 TEST         28 PROGRAM      >100 027
018 >013 TEST;C        8 DIS/VAR 80   >11b 007
019 >014 TEST;O       17 DIS/FIX 80   >122 016
```

Disk 88. Contents of file ALLOCC

```
/*
** memory storage allocation routines.
**
** the memory allocation routines 'alloc' and 'free' are used
** to allocate arbitrary memory and to release it when done.
** the include file 'alloci' must be included in your
** global declaration area, see this file for information
** how to reserve available memory.
**
*/

/*
**
** c=alloc(n)
**
** Returns next available address or 0 if there is no
** more room.
**
*/

alloc(n)
    int n;
  {
    if(!allocfirst){
        ++allocfirst;
        allocp = allocbuf;
        }
    if(is_odd(n))
        ++n; /* make sure it's even */
    if(allocp + n <= allocbuf + ALLOCBYTES){
        allocp = allocp + n;
        return(allocp - n);
        }
      else
        return(0);
  }

/*
**
** free(p)
**
** free storage allocated by alloc, the area
** must be free'd in the same order as they
** were allocated except in reverse.
**
*/

free(p)
```

```
    char *p;
  {
    if(p >= allocbuf & allocbuf + ALLOCBYTES)
        allocp = p;
  }

/*
**
** is_odd(n)
**
** Returns 1 if the value contained in x is odd, otherwise
** returns the value 0.
**
*/

is_odd(n)
    int n;
  {
    return(n % 2);
  }
```

Disk 88. Contents of file ALLOCI

```
/*
**
** the following defines and definitions are used with the
** storage allocator.
**
**/

#define ALLOCBYTES 256 /* size of available space in bytes (should be even) */

int  allocfirst = 0;       /* is free position set */
char allocbuf[ALLOCBYTES]; /* storage for alloc    */
char *allocp;              /* next free position   */
```

Disk 88. Contents of file CDIRC

```
/*
** Read a directory from disk
**
** Author: Tom Bentley
**         207 - 324 Cambridge ST. N.
**         Ottawa, Ont.
**         Canada  K1R 7B5
**
** Date:  May 5, 1986
**
** required libraries:
** - tcio
** - float
** - printf
** - csup
*/

#include "dsk1.tcioi"
#include "dsk1.floati"
#include "dsk1.printfio"

char buff[255];
char typs[40] = {"Dis/Fix","Dis/Var","Int/Fix","Int/Var","Program"};
char dir[6] = {"DSKx."};

main()
  {
    int fp,rec,eof,size,typ,b_ptr;
    char str[20],drive,prot,i;
    float f1[FLOATLEN],f2[FLOATLEN],res[FLOATLEN];

    rec = 0;
    puts("Drive Number ?");
    drive = getchar();
    dir[3] = drive;
    fp=topen(dir,INPUT+RELATIVE+INTERNAL+FIXED,0);
    if(fp<1)
      printf("I/O Error = %d\n",fp);
    else {
      /* get disk name and stats */
      putchar(12); /* clear screen */
      tread(buff,RELSEQ,fp,&size);
      b_ptr = buff;
      get_name(&b_ptr,str);
      printf("Directory= %s   Diskname= %s\n",dir,str);
      get_num(&b_ptr,str,f1); /* dummy record type */
      get_num(&b_ptr,str,f1); /* total sectors on disk */
      get_num(&b_ptr,str,f2); /* available sectors */
```

```
      printf("Available= %s ",str);
      fexp(f1,"-",f2,res);
      printf("Used= %s\n\n",ftos(res,str,0,0,0));
      puts(" Filename  Size    Type    P\n");
      puts("---------- ---- ---------- -\n");
      while(1) {
        poll(1);
        eof=tread(buff,RELSEQ,fp,&size);
        if(eof) break;
        b_ptr = buff;
        if(!get_name(&b_ptr,str)) break;
        printf("%-10s ",str);  /* file name */
        get_num(&b_ptr,str,f1); /* file type */
        typ=ftoi(f1);
        if(typ < 0){
          prot = 'Y';
          typ = -typ;
          }
        else prot = ' ';
        --typ;
        get_num(&b_ptr,str,f1); /* sectors allocated for file */
        printf("%4s %-7s",str,&typs[typ*8]);
        get_num(&b_ptr,str,f1); /* bytes per record */
        if(typ != 4)
          printf("%3s",str);
        else
          printf("   ");
        printf(" %c\n",prot);
        }
      }
    tclose(fp);
    }

/*
** get name of disk or file
*/

get_name(buff,t)
  int *buff;
  char *t;
  {
     char *b;
     int  j,siz;

     b = *buff;
     j=siz=*b++;
     *buff = *buff + j + 1;
     while(j--)
       *t++ = *b++;
     *t = '\000';  /* null terminate */
```

```
     return(siz);
  }

get_num(buff,t,f)
  int  *buff;
  char *t;
  float *f;
  {
    char *b;

    b = *buff;
    ++b;
    fcpy(b,f);
    ftos(b,t,0,0,0);
    *buff = *buff + 9;
  }
```

Disk 88. Contents of file CHRLOADC

```
/***
 ***  char_load is used to load in a
 ***  character definition file.
 ***
 ***  By Tom Bentley
 ***     P.O. Box 346
 ***     Osgoode, Ont.
 ***     Canada, K0A 2W0
 ***/

entry char_load;

#asm
       REF  VMBW,DSRLNK

PAB    EQU  8192
CHRPAB DATA >0500,>07FA,>0000,>0800
CHRLEN DATA >000B
CHRNAM TEXT 'DSK1.CHARA1               '
       EVEN
#endasm

/*
 * char_load(file)
 *
 * char_load load the specified character
 * definition file.
 *
 * to use specify: extern char_load();

 * example.  char_load("DSK1.CHARA1")
 *
 *           char_load("")  'loads DSK1.CHARA1 by default'
 *
 */

char_load(file)
    char *file;
  {

#asm
       CLR  4            LENGTH OF FILE
       MOV  @2(14),0     ADDRESS OF FILE
       MOVB *0,*0        IS IT A NULL?
       JEQ  CLOAD        THEN USE DEFAULT
       LI   1,CHRNAM     GET ADDRESS OF WHERE TO PUT FILE NAME
CMOV   MOVB *0+,*1+      MOVE IT
       JEQ  CNXT         FOUND A NULL
```

```
       INC  4             INC SIZE OF FILE NAME
       JMP  CMOV          MORE TO MOVE
CNXT   MOV  4,@CHRLEN     SET UP FILE LENGTH
CLOAD  LI   0,PAB         VDP LOCATION TO MOVE CHRPAB
       LI   1,CHRPAB      CHARACTER PAB
       LI   2,10          PAB LENGTH
       A    @CHRLEN,2     ADD LENGTH OF FILE NAME
       BLWP @VMBW         WRITE TO VDP
       LI   0,PAB+9       POINTER TO NAME LENGTH
       MOV  0,@>8356      DEVICE POINTER
       BLWP @DSRLNK       READ THE FILE
       DATA 8
       CLR  @>837C        CLEAR STATUS
#endasm

  }
```

Disk 88. Contents of file CTYPEC

```
/*
 * Written by: Tom Bentley
 *             P.O. Box 346
 *             Osgoode, Ont
 *             K0A 2W0
 */


/*
** abs -- returns absolute value of nbr
*/
abs(nbr) int nbr; {
  if(nbr<0) return -nbr;
  else return nbr;
  }


/*
** return pointer to the first occurrence of c in s
*/
indexp(s, c) char *s, c; {
  while(*s) if(*s++ == c) return --s;
  return 0;
  }


/*
** isalnum -- true if argument is valid ASCII alphabetic or digit
*/
isalnum(c) char c; {
  if((c >= 'a' & c <= 'z') | (c >= 'A' && c <= 'Z') |
      (c >= '0' & c <= '9')) return 1;
  else return 0;
  }


/*
** isalpha -- true if argument is valid ASCII alphabetic
*/
isalpha(c) char c; {
  if((c >= 'a' & c <= 'z') | (c >= 'A' & c <= 'Z')) return 1;
  else return 0;
  }


/*
** isascii -- true if argument is valid ASCII character
*/
```

```
isascii(c) char c; {
  if(c <= 127) return 1;
  else return 0;
  }


/*
** iscntrl -- true if argument is valid ASCII control character
*/
iscntrl(c) char c; {
  if(c < ' ' | c  == 127) return 1;
  else return 0;
  }


/*
** isdigit -- true if argument is valid ASCII digit
*/
isdigit(c) char c; {
  if(c >= '0' & c <= '9') return 1;
  else return 0;
  }


/*
** islower -- true if argument is valid ASCII lower case alphabetic
*/
islower(c) char c; {
  if(c >= 'a' & c <= 'z') return 1;
  else return 0;
  }


/*
** isprint -- true if argument is valid ASCII graphic
*/
isprint(c) char c; {
  if(c >= ' ' & c <= 126) return 1;
  else return 0;
  }


/*
** ispunct -- true if argument is neither ASCII control or alphabetic
**
** requires - isalpha() and iscntrl()
*/
ispunct(c) char c; {
  if(isalpha(c) | iscntrl(c)) return 0;
  else return 1;
  }
```

```
/*
** isspace -- true if argument is ASCII space, tab, carriage return,
**            newline (line feed), or form feed  (ie, "white space")
*/
isspace(c) char c; {
  if(c == ' ' | c == '\t' | c == '\r' | c == '\n' | c == '\f') return 1;
  else return 0;
  }


/*
** isupper -- true if argument is valid ASCII upper case alphabetic
*/
isupper(c) char c; {
  if(c >= 'A' & c <= 'Z') return 1;
  else return 0;
  }


/*
** return pointer to the last occurrence of c in s
*/
rindex(s, c) char *s, c; {
  char *rindex;
  rindex = 0;
  while(*s) {
    if(*s == c) rindex = s;
    s++;
    }
  return rindex;
  }


/*
** if character is upper case, convert to lower case
*/
tolower(c) char c; {
  if(c >= 'A' & c <= 'Z') return (c + ('a' - 'A'));
  return c;
  }


/*
** if character is lower case, convert to upper case
*/
toupper(c) char c; {
  if(c >= 'a' & c <= 'a') return (c + ('a' + 'A'));
  return c;
  }
```

```
/* name: index(string,substring)
 * function: Returns the starting
 * location of the substring in
 * the string, or the value -1 if
 * the substring was not found.
 * ex. index("This is it","it");
 *       returns 8
 *     index("This is it","IT");
 *       returns -1
 */
index(string,substring)
  char string[],substring[];
  {
    int i,j,k;

    i=0;
    while(string[i]!=0) {
      j=i;
      k=0;
      while(substring[k]==string[j]) {
        if(substring[1+k]==0)
          return(i);
        else {
          j++;
          k++;
          }
        }
      i++;
      }
    return(-1);
    }

/*
 * String copy
 *
 *  strcpy(s,t) - copy t to s
 *
 */

strcpy(s,t)
  char *s,*t;
  {
    while(*s++ = *t++);
  }

/*
 * string compare
 *
 *  strcmp(s,t) - return <0 if s<t,
 *                return 0  if s==t,
```

```
 *                return >0 if s>t
 *
 */

strcmp(s,t)
  char s[],t[];
  {
    int i;

    i=0;
    while(s[i] == t[i])
      if(s[i++] == '\0')
        return(0);
    return(s[i] - t[i]);
  }

/*
 * string concatenate
 *
 *  strcat(s,t) - concatenate to to end of s
 *
 */

strcat(s,t)
  char *s,*t;
  {
    --s;
    while(*++s);
    while(*s++ = *t++);
  }

/*
 * string length
 *
 *  strlen(s) - length of string
 *
 */

strlen(s)
  char *s;
  {
    char *t;

    t=s-1;
    while(*++t);
    return(t-s);
  }

/* fill s with t for n times */
fillb(s,t,n)
```

```
  char *s,t;
  int   n;
  {
  while(n--)
    *s++=t;
  *s=0;
  }

/* insert t into s starting at n */
instr(s,t,n)
  char *s,*t;
  int  n;
  {
  int  l;
  l=0;
  while(++l<n)
    *++s;
  while((*t!= 0)&(*s!=0))
    *s++=*t++;
  }

/* simple conversion functions */
/*
** n=atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
/*
** itod(nbr,str,sz) -
**        convert nbr to signed decimal string of width sz
**        right justified, blank filled, result in str[].
**        sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
```

```
  while(sz) str[--sz]=' ';
}


/*
** itod2-- convert nbr to signed decimal string of width 7
**         (including null) right adjusted, blank filled
*/
itod2(nbr, str) int nbr; char str[]; {
  char sgn;
  int count;
  if(nbr<0) {nbr = -nbr; sgn='-';}
  else sgn=' ';
  str = str+6; /* find end of string */
  *str-- = 0; /* store ending null */
  count = 6; /* remaining characters */
  while(1) {
    *str-- = (nbr % 10 + '0');
    count--;
    if(!(nbr=nbr/10)) break;
    }
  *str-- = sgn;
  while(--count) *str-- = ' ';
  }

/*
** itou -- convert nbr to unsigned decimal string of width 7
**         (including null) right adjusted, blank filled
*/
itou(nbr, str) int nbr; char str[]; {
  int count, lowbit;
  str = str+6; /* find end of string */
  *str-- = 0; /* store ending null */
  count = 6; /* remaining characters */
  while(1) {
    lowbit = nbr & 1;
    nbr = (nbr >> 1) & 32767;  /* divide by 2 */
    *str-- = ((nbr % 5) << 1) + lowbit + '0';
    count--;
    if(!(nbr=nbr/5)) break;
    }
  while(count--) *str-- = ' ';
  }

/*
** itoo -- convert nbr to unsigned octal string of width 7
**         (including null) right adjusted, blank filled
*/
itoo(nbr, str) int nbr; char str[]; {
  int count;
```

```
  str = str+6; /* find end of string */
  *str-- = 0; /* store ending null */
  count = 6; /* remaining characters */
  while(1) {
    *str-- = (nbr & 7) + '0';
    count--;
    if(!(nbr = ((nbr >> 3) &  8191))) break;
    }
  while(count--) *str-- = ' ';
  }

/*
** itox -- convert nbr to unsigned hexadecimal string of width 7
**         (including null) right adjusted, blank filled
*/
itox(nbr, str) int nbr; char str[]; {
  int count, digit;
  str = str+6; /* find end of string */
  *str-- = 0; /* store ending null */
  count = 6; /* remaining characters */
  while(1) {
    if((digit=(nbr & 15)) < 10)
      *str-- = (digit + '0');
    else
      *str-- = (digit + ('A' - 10));
    count--;
    if(!(nbr = ((nbr >> 4) &  4095))) break;
    }
  while(count--) *str-- = ' ';
  }
```

Disk 88. Contents of file DIR;C

```
/*
** Read a directory from disk
**
** Athor: Tom Bentley
**        207 - 324 Cambridge ST. N.
**        Ottawa, Ont.
**        Canada  K1R 7B5
**
** Date:  May 5, 1986
**
** required libraries:
** - tcio
** - float
** - printf
** - csup
*/

#include "dsk2.tcioi"
#include "dsk2.floati"
#include "dsk1.prf"

char buff[255];
char typs[40] = {"Dis/Fix","Dis/Var","Int/Fix","Int/Var","Program"};
char dir[6] = {"DSKx."};

main()
  {
    int fp,rec,eof,size,typ,b_ptr;
    char str[20],drive,prot,i;
    float f1[FLOATLEN],f2[FLOATLEN],res[FLOATLEN];

    rec = 0;
    puts("Drive Number ?");
    drive = getchar();
    dir[3] = drive;
    fp=topen(dir,INPUT+RELATIVE+INTERNAL+FIXED,0);
    if(fp<1)
      printf("I/O Error = %d\n",fp);
    else {
      /* get disk name and stats */
      putchar(12); /* clear screen */
      tread(buff,RELSEQ,fp,&size);
      b_ptr = buff;
      get_name(&b_ptr,str);
      printf("Directory= %s   Diskname= %s\n",dir,str);
      get_num(&b_ptr,str,f1); /* dummy record type */
      get_num(&b_ptr,str,f1); /* total sectors on disk */
      get_num(&b_ptr,str,f2); /* available sectors */
```

```
      printf("Available= %s ",str);
      fexp(f1,"-",f2,res);
      printf("Used= %s\n\n",ftos(res,str,0,0,0));
      puts(" Filename  Size    Type    P\n");
      puts("---------- ---- ---------- -\n");
      while(1) {
        poll(1);
        eof=tread(buff,RELSEQ,fp,&size);
        if(eof) break;
        b_ptr = buff;
        if(!get_name(&b_ptr,str)) break;
        printf("%-10s ",str);  /* file name */
        get_num(&b_ptr,str,f1); /* file type */
        typ=ftoi(f1);
        if(typ < 0){
          prot = 'Y';
          typ = -typ;
          }
        else prot = ' ';
        --typ;
        get_num(&b_ptr,str,f1); /* sectors allocated for file */
        printf("%4s %-7s",str,&typs[typ*8]);
        get_num(&b_ptr,str,f1); /* bytes per record */
        if(typ != 4)
          printf("%3s",str);
        else
          printf("   ");
        printf(" %c\n",prot);
        }
      }
    tclose(fp);
    }

/*
** get name of disk or file
*/

get_name(buff,t)
  int *buff;
  char *t;
  {
     char *b;
     int  j,siz;

     b = *buff;
     j=siz=*b++;
     *buff = *buff + j + 1;
     while(j--)
       *t++ = *b++;
     *t = '\000';  /* null terminate */
     return(siz);
```

```
  }

get_num(buff,t,f)
  int  *buff;
  char *t;
  float *f;
  {
    char *b;

    b = *buff;
    ++b;
    fcpy(b,f);
    ftos(b,t,0,0,0);
    *buff = *buff + 9;
  }
```

Disk 88. Contents of file FLOATC

```
/*
 * FLOATING POINT ROUTINES  V1.1
 *
 * Written by :
 *
 *  Tom Bentley
 *  P.O. Box 346
 *  Osgoode, Ontario
 *  Canada  K0A 2W0
 *
 *  Date Changed: May 6, 1986
 *
 */

#asm
 REF C$GPLL,XMLLNK,VMBW
 DEF ITOF,FTOI,STOF,FTOS,FEXP,FCOM,FPGET,FPPUT,FINT,FCPY

FAC    EQU >834A
ARG    EQU >835C
STATUS EQU >837C
VSPTR  EQU >836E
VDPWRK EQU >24CA
EQ     DATA >2000
GT     DATA >4000
GTEQ   DATA >6000
ONE    DATA 1
SAVREG BSS 12
#endasm

#define FLOATLEN 8
#define float    char


float ftemp[FLOATLEN];
int   itemp;

fpget(s,f)
  char  *s;
  float *f;
  {
    gets(s);
    return(stof(s,f));
  }

fpput(f,s)
  float *f;
  char  *s;
```

```
  {
    puts(ftos(f,s,0,0,0));
  }

/*
 * name: itof - integer to floating point
 *
 * itof(i,f)
 *
 * i = integer value
 * f = float pointer
 * returns pointer to float number
 *
 */

itof(i,f)
  int i;
  float *f;
  {

#asm
       MOV  @4(14),@FAC  MOVE INTEGER TO FAC
       BLWP @XMLLNK
       DATA >2300
       LI   2,8
       MOV  @2(14),0     GET FLOAT POINTER
       LI   1,FAC
ITOFL  MOVB *1+,*0+
       DEC  2
       JNE  ITOFL
#endasm
    return(f);
    }

/*
 * ftoi - floating point to integer
 *
 * i = ftoi(f)
 *
 * i = integer value
 * f = float pointer
 *
 */

ftoi(f)
  float *f;
  {
#asm
       LI   2,8
       MOV  @2(14),0     GET FLOAT POINTER
```

```
       LI   1,FAC
FTOIL  MOVB *0+,*1+
       DEC  2
       JNE  FTOIL
       BLWP @XMLLNK
       DATA >1200
       MOV  @FAC,@ITEMP
#endasm
    return(itemp);
    }

/*
 * name: stof - string to floating point
 *
 * stof(s,f)
 *
 * s = string pointer
 * f = float pointer
 * returns pointer to float number
 *
 */

stof(s,f)
  char *s;
  float *f;
  {
#asm
       LI   0,VDPWRK     MOVE STRING NUMBER TO VDP
       MOV  @4(14),1
       CLR  2            USED FOR COUNTER
STOFL2 INC  2
       CB   *1+,2        IS IT A NULL?
       JEQ  STOFL1
       CI   2,21         UP TO 20 DIGITS
       JLT  STOFL2
       DEC  1
       MOVB 2,*1         NULL TERMINATE
STOFL1 MOV  @4(14),1
       BLWP @VMBW        WRITE TO VDP
       MOV  0,@FAC+12    ADDRESS OF STRING IN VDP
       BLWP @XMLLNK
       DATA >1000
       MOV  @2(14),0
       LI   1,FAC
       LI   2,8
STOFL  MOVB *1+,*0+
       DEC  2
       JNE  STOFL
#endasm
    return(f);
    }
```

```
/*
 * name: ftos - floating point to string
 *
 * ftos(f,s,mode,signif,decimal)
 *
 * s = string pointer
 * f = float pointer
 * mode = 0 - basic mode, 1 - fix mode
 *  if fixed mode then the following must
 *  be specified.
 *
 * signif  = number of significant digits
 * decimal = indicates the number of digits to
 *           the right of the decimal point
 *
 * returns pointer to string
 *
 */

ftos(f,s,mode,signif,decimal)
  float *f;
  char  *s;
  int   mode,signif,decimal;
  {
#asm
       MOV  @6(14),0     MODE
       SWPB 0
       MOVB 0,@FAC+11
       MOV  @4(14),0     SIGNIF
       SWPB 0
       MOVB 0,@FAC+12
       MOV  @2(14),0     DECIMAL
       SWPB 0
       MOVB 0,@FAC+13
       LI   2,8
       MOV  @10(14),0
       LI   1,FAC
FTOSL  MOVB *0+,*1+
       DEC  2
       JNE  FTOSL
       LI   3,>8320      SAVE C WORKSPACE
       LI   4,SAVREG
       LI   5,6
X1     MOV  *3+,*4+
       DEC  5
       JNE  X1
       CLR  @STATUS
       BL   @C$GPLL
       DATA >0014
       CLR  0
```

```
       MOVB @FAC+11,0
       SWPB 0
       AI   0,>8300
       CLR  2
       MOVB @FAC+12,2
       SWPB 2
       LI   3,SAVREG     RESTORE C WORKSPACE
       LI   4,>8320
       LI   5,6
X2     MOV  *3+,*4+
       DEC  5
       JNE  X2
       MOV  @8(14),1
FTOSL1 MOVB *0+,*1+
       DEC  2
       JNE  FTOSL1
       CLR  0
       MOVB 0,*1         TERMINATE STRING WITH NULL
#endasm
      return(s);
      }

/*
 * name: fexp - execute an expression
 *
 *  fexp(f1,op,f2,res)
 *
 *  f1 = floating point number
 *  f2 = second floating point number
 *  op = op code (*,/,+,/)
 *  res = result in floating point format
 *  returns pointer to float result
 *
 */

fexp(f1,op,f2,res)
  float *f1,*f2,*res;
  char  *op;
  {
#asm
       MOV  @6(14),0     GET OP CODE
       CLR  3
       MOVB *0,3
       LI   2,8          MOVE F1 TO ARG
       MOV  @8(14),0
       LI   1,ARG
FEXPL1 MOVB *0+,*1+
       DEC  2
       JNE  FEXPL1
       LI   2,8          MOVE F2 TO FAC
       MOV  @4(14),0
```

```
       LI   1,FAC
FEXPL2 MOVB *0+,*1+
       DEC  2
       JNE  FEXPL2
FADD   CI   3,>2B00      ADD OP CODE
       JNE  FSUB
       BLWP @XMLLNK
       DATA >0600
       JMP  FRES
FSUB   CI   3,>2D00      SUBTRACT OP CODE
       JNE  FMULT
       BLWP @XMLLNK
       DATA >0700
       JMP  FRES
FMULT  CI   3,>2A00      MULTIPLY OP CODE
       JNE  FDIV
       BLWP @XMLLNK
       DATA >0800
       JMP  FRES
FDIV   CI   3,>2F00      DIVIDE OP CODE
       JNE  FRES
       BLWP @XMLLNK
       DATA >0900
FRES   MOV  @2(14),0
       LI   1,FAC
       LI   2,8
FEXPL3 MOVB *1+,*0+
       DEC  2
       JNE  FEXPL3
#endasm
    return(res);
  }

/*
 * name: fint - greatest integer function
 *
 * fint(f1,f2)
 *
 *  f1 = floating point value
 *  f2 = int floating point value
 *  return address of f
 *
 */

fint(f1,f2)
  float *f1,*f2;
  {
#asm
       MOV  @4(14),0
       LI   1,FAC
```

```
       LI   2,8
FINTL1 MOVB *0+,*1+
       DEC  2
       JNE  FINTL1
       LI   3,>8320      SAVE C WORKSPACE
       LI   4,SAVREG
       LI   5,6
X3     MOV  *3+,*4+
       DEC  5
       JNE  X3
       CLR  @STATUS
       BL   @C$GPLL
       DATA >0022
       LI   3,SAVREG     RESTORE C WORKSPACE
       LI   4,>8320
       LI   5,6
X4     MOV  *3+,*4+
       DEC  5
       JNE  X4
       MOV  @2(14),0
       LI   1,FAC
       LI   2,8
FINTL2 MOVB *1+,*0+
       DEC  2
       JNE  FINTL2
#endasm
    return(f2);
  }

/*
 * name: fcom - logical compare
 *
 * fcom(f1,rel,f2)
 *
 * f1 = floating point value
 * f2 = floating point value
 * rel = relation (==,!=,<,<=,>,>=)
 *
 * returns 1 for true 0 for false
 *
 */

fcom(f1,rel,f2)
  float *f1,*f2;
  char  *rel;
  {
#asm
       MOV  @4(14),3     GET REL CODE
       MOV  @6(14),0     MOVE F1 TO ARG
       LI   1,ARG
       LI   2,8
```

```
FCOML1 MOVB *0+,*1+
       DEC  2
       JNE  FCOML1
       MOV  @2(14),0     MOVE F2 TO FAC
       LI   1,FAC
       LI   2,8
FCOML2 MOVB *0+,*1+
       DEC  2
       JNE  FCOML2
       CLR  @ITEMP
       BLWP @XMLLNK
       DATA >0A00
       CLR  5
       MOVB @STATUS,5
       CLR  4
       MOVB *3+,4
       SWPB 4
       MOVB *3,4
       SWPB 4
EQUAL  CI   4,>3D3D      ==
       JNE  NEQUAL
       COC  @EQ,5
       JNE  FCOMR1
       JMP  FCOMR9
NEQUAL CI   4,>213D      !=
       JNE  LTHAN
       COC  @EQ,5
       JEQ  FCOMR1
       JMP  FCOMR9
LTHAN  CI   4,>3C00      <
       JNE  GTHAN
       CZC  @GTEQ,5
       JNE  FCOMR1
       JMP  FCOMR9
GTHAN  CI   4,>3E00      >
       JNE  LTHNEQ
       COC  @GT,5
       JNE  FCOMR1
       JMP  FCOMR9
LTHNEQ CI   4,>3C3D      <=
       JNE  GTHNEQ
       COC  @GT,5
       JEQ  FCOMR1
       JMP  FCOMR9
GTHNEQ CI   4,>3E3D      >=
       JNE  FCOMR1
       COC  @GT,5
       JEQ  FCOMR9
       COC  @EQ,5
       JNE  FCOMR1
```

```
FCOMR9 MOV  @ONE,@ITEMP
FCOMR1 NOP
#endasm
    return(itemp);
  }

fcpy(f1,f2)
  float *f1,*f2;
  {
#asm
       MOV  @4(14),0
       MOV  @2(14),1
       LI   2,8
FCPYL  MOVB *0+,*1+
       DEC  2
       JNE  FCPYL
#endasm
    return(f2);
  }
```

Disk 88. Contents of file FLOATDOC

'C' Floating Point Library (by Tom Bentley 860201)

Tom Bentley
P.O. Box 346
Osgoode, Ont.
Canada K0A 2W0

The following is a description of the floating point library that may be used by Clint Pulley's c99 program.

Library and Include Files

a) FLOAT - The floating point library. This contains all of the floating point routines.

b) FLOATI - Include file containing the REF's and defines required by the floating point library.

How to define a FLOAT data type:

To define a float data type use 'float' as the type and FLOATLEN as the size of the data type. A float number on the TI is always stored as an 8 byte number, keep this in mind if you wish to manipulate it with your own routines.

Example.

```
/* define float type */
float number[FLOATLEN];
```

Functions in FLOAT:

- Prompt for floating point number

```
float f[FLOATLEN];
char  *c, s[input size];

c=fpget(s,f);
```

Prompts for a floating point string and converts is to a floating point number. s is a character array which receives the string, f is the float array which receives the floating point number and c is a character pointer that contains the pointer to the float array.

Example 1.

```
/* prompt for a float number */

puts("Enter a number: ");
fpget(s,f);
```

- Display floating point number to screen

```
float f[FLOATLEN];
char  s[display size];

fpput(f,s);
```

Display at floating point number to the screen. Locate(r,c) may be used to position to the start of where to display from. f is the float array that contains the floating point number and s is the char array that contains the string representation of the floating point number. s is available to you even after the call to fpput so you may use it for other purposes.

Example 1.

```
/* display floating point number */

puts("The number is: ");
fpput(f,s);
```

- Integer to floating point

```
int   i;
float f[FLOATLEN];
char  *c;

c=itof(i,f);
```

Converts an integer value to a floating point number. i is any integer value, f is the float array which receives the floating point number and c is a character pointer that contains the pointer to the float array.

Example 1.

```
/* convert integer to float */

i = 100;
itof(i,f);
puts("i contains ");
fpput(f,s);
```

- Floating point to integer

```
float f[FLOATLEN];
int   i;

i=ftoi(f);
```

Converts a floating point number to integer. f is the floating point array to be converted and i is the integer. The floating point array must contain a valid integer number in the range of +32767 and -32768.

Example 1.

```
/* convert float to integer */

i=ftoi(f);
puts("i contains ");
itod(i,s,4);
puts(s);
```

- String to floating point

```
float f[FLOATLEN];
char  *c, s[string size];

c=stof(s,f);
```

Converts a numeric string to a floating point number. s is a character array that contains the number to be converted, f is the float array and c is a character pointer that contains the pointer to the float array.

Example 1.

```
/* convert string to float */

puts("Enter a number: ");
gets(s);
stof(s,f);
```

- Floating point to string

```
float f[FLOATLEN];
char  *c, s[string size];
int   mode, signif, decimal;

c=ftos(f,s,mode,signif,decimal);
```

Converts a float array to a string array. f is a float array that contains the float number to be converted, s is a character array that contains the converted number, mode is a value that specifies the conversion mode (0 = basic mode, displayed as in basic, 1 = fix mode, if fixed mode then signif and decimal must be specified.), signif contains the number of significant digits, decimal indicates the number of digits to the right of the decimal point and c is a character pointer that contains the pointer to the converted string.

Example 1.

```
/* convert float to string */

/* use basic mode */
ftos(f,s,0,0,0);
puts("Value is ");
puts(s);

/* use fix mode, scientific notation */
ftos(f,s,1,5,2);
puts("\nValue is ");
puts(s);

/* if f = 2355 then s = 2.355E+03    */
/* if f =   25 then s = 2.5E+01      */
/* if f = 1234.56 then s = 1.2346E+03*/
```

- Execute float expression

```
float f1[FLOATLEN],f2[FLOATLEN],res[FLOATLEN];
char  *c, op[2];

c=fexp(f1,op,f2,res);
```

Executes and expression between f1 and f2 using the provided op code. f1 is a float array containing the first operand, f2 is a float array containing the second operand, op is a character array containing the op code (one of *,/,+,-), res is a float array that contains the result of the expression and c is a character pointer that points the result.

Example 1.

```
/* perform expression */

/* add two numbers */
fexp(f1,"+",f2,res);

/* subtract two numbers */
fexp(f1,"-",f2,res);

/* multiply two numbers */
fexp(f1,"*",f2,res);

/* divide two numbers */
fexp(f1,"/",f2,res);
```

- Compare float numbers

```
float f1[FLOATLEN], f2[FLOATLEN];
char  rel[3];
int   true;

true=fcom(f1,rel,f2);
```

Compares two floating point numbers using the relation supplied (one of ==,!=,<,<=,>,>=). f1 is a float array containing first number, f2 is a float array containing second number and rel is a character array containing the compare relation.

Example 1.

```
/* compare two float numbers */

/* are they equal (==) ? */
if(fcom(f1,"==",f2))
/* true */

/* are they not equal (!=) ? */
if(fcom(f1,"!=",f2))
/* true */

/* is f1 < f2 ? */
if(fcom(f1,"<",f2))
/* true */

/* is f1 <= f2 ? */
if(fcom(f1,"<=",f2))
/* true */
```

```
/* is f1 > f2 ? */
if(fcom(f1,">",f2))
/* true */

/* is f1 >= f2 ? */
if(fcom(f1,">=",f2))
/* true */
```

- Float greatest integer function

```
float f1[FLOATLEN], f2[FLOATLEN];
char  *c;

c=fint(f1,f2);
```

Takes a float value and returns it's greatest integer value. f1 is a float array containing a float value, f2 is a float array that will contain the greatest integer value and c is a character pointer that points to f2.

Example 1.

```
/* take the greatest integer value */

f= some float value
fint(f,f);
```

- Copy float number

```
float f1[FLOATLEN], f2[FLOATLEN];
char  *c;

c=fcpy(f1,f2);
```

Copy one float array to another float array, c is a character pointer to f2.

Example 1.

```
/* copy one float to another float */

fcpy(f1,f2);
```

Disk 88. Contents of file FLOATI

```
/* C Float Include file */

#asm
 REF ITOF,FTOI,STOF,FTOS,FEXP,FCOM,FPGET,FPPUT,FINT,FCPY
#endasm

#define float    char /* float data type         */
#define FLOATLEN 8    /* size of float data type */
```

Disk 88. Contents of file TCIOC

```
/*
 * Written by:
 *
 *  Tom Bentley
 *  P.O. Box 346
 *  Osgoode, Ont.
 *  Canada  K0A 2W0
 */

#asm
 REF DSRLNK,VMBW,VSBW,VSBR,VMBR,GRMRA,GRMWA
 DEF TOPEN,TCLOSE,TREAD,TWRITE
#endasm

#define PAB1      8192
#define PAB2      8498
#define PAB3      8804
#define PAB4      9110
#define PABLEN      50
#define NUMFIL       4
#define DSRCLR      31
#define WS       33536


char *status;
int  *wsp;
int  dsrerr,saveg;
int  filalloc[NUMFIL]={PAB1,PAB2,PAB3,PAB4};
int  filstk[NUMFIL]={0,0,0,0};
char pab[PABLEN];

topen(name,access,size)
  char *name;
  char access,size;
  {
  int  *pb,lp,fp;

  wsp = WS;    /* >8300 */


  /* set pab up for open */
  pab[0] = 0;     /* op code */
  pab[1] = access;
  /* data buffer address */
  fp=0;
  dsrerr=4; /* if file table is full */
  while(fp<NUMFIL-1) {
    if(filstk[fp]==0) {
```

```
      filstk[fp]=filalloc[fp];
      pb=&pab[2];
      *pb=PABLEN+filstk[fp++];
      dsrerr=0;
      break;
      }
    else
      fp++;
    }
  if(dsrerr==0) {
    pab[4] = size;  /* record length */
    pab[5] = size;  /* character count */
    pab[6] = 0;     /* record number */
    pab[7] = 0;
    pab[8] = 0;     /* screen offset */
    /* name length */
    pab[9] = tlen(name);
    /* device name... */
    lp=10;
    while(*name)
      pab[lp++]=*name++;
    if(linkdsr(fp)!=-1)
      fp=-dsrerr;
    }
  else fp=-dsrerr;
  return(fp);
  }

tclose(fp)
  int fp;
  {
  int eof;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp=WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=1;
  eof=linkdsr(fp);
  if(eof!=-1&eof!=7)
    return(-2);
  filstk[fp-1]=0;
  fp=0;
  }
```

```
tread(buff,rec,fp,size)
  char buff[];
  int  rec,fp,*size;
  {
  int  *pb,eof;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp = WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=2; /* read op code */
  pab[1]=pab[1]&DSRCLR;
  /* if fixed and relative */
  if((pab[1]&17)==1 & rec>=0) {
    pb=&pab[6];
    *pb=rec;
    }
  eof=linkdsr(fp);
  if(eof==5)
    return(-1);
  if(eof!=-1)
    return(-2);
  wsp=WS;
  pb=&pab[2];
  *wsp++=*pb;
  *wsp++=buff;
  *wsp=pab[4];
  #asm
   MOV  1,6
   BLWP @VMBR * MOVE DATA TO RAM
  #endasm
  wsp = WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR * MOVE PAB TO RAM
   MOV  1,4
   AI   4,5   * NUM OF CHAR READ
   CLR  5
   MOVB *4,5
   SWPB 5
   MOV  5,2   * SIZE OF BUFFER
   A    5,6
   MOVB 5,*6  * TERMINATE WITH NULL
  #endasm
```

```
  *size=*wsp;
  return(0);
  }

twrite(buff,rec,fp,size)
  char buff[];
  int  rec,fp,size;
  {
  int  *pb,fr;

  if(fp>NUMFIL|fp<1|filstk[fp-1]==0)
    return(-2);
  wsp=WS;
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=PABLEN;
  #asm
   BLWP @VMBR
  #endasm
  pab[0]=3; /* write op code */
  pab[1]=pab[1]&DSRCLR;
  /* if fixed and relative */
  fr=pab[1]&17;
  if(fr==1) {
    pb=&pab[6];
    *pb=rec;
    }
  if(size)
    pab[5]=size;
  else pab[5]=tlen(buff);
  wsp=WS;
  pb=&pab[2];
  *wsp++=*pb;
  *wsp++=buff;
  *wsp=pab[5];
  #asm
   BLWP @VMBW * MOVE BUFF TO VDP
  #endasm
  if(linkdsr(fp)!=-1)
    return(-2);

  return(0);
  }

linkdsr(fp)
  int fp;
  {
  wsp = WS;    /* >8300 */
  status = 33660; /* >837C */
```

```
  *status = 0;

  /* set up workspace registers */
  *wsp++=filstk[fp-1];
  *wsp++=pab;
  *wsp=10+pab[9];
  #asm
   BLWP @VMBW
   MOV  0,4
   AI   4,9
   MOV  4,@>8356 * DEVICE POINTER
   SETO @DSRERR
   MOVB @GRMRA,@SAVEG
   NOP
   MOVB @GRMRA,@SAVEG+1
   DEC  @SAVEG
   BLWP @DSRLNK
   DATA 8
   JEQ  DE
   JMP  DA
DE SRL  0,8
   MOV  0,@DSRERR
DA MOVB @SAVEG,@GRMWA
   NOP
   MOVB @SAVEG+1,@GRMWA
  #endasm
  return(dsrerr);
  }


tlen(s)
  char *s;
  {
  char *t;
  t=s-1;
  while(*++t);
  return(t-s);
  }
```

Disk 88. Contents of file TCIODOC

Enhanced I/O Library (by Tom Bentley 860201)

The following is a description of a new I/O library that may be used by c99 users to perform I/O functions on any file type on the TI. *Note*: This library was written using Clint Pulley's c99 program.

I. Library and Include Files

a) TCIO - The file input/output library. This contains the file tables and all file I/O functions.

b) TCIOI - Include file containing I/O definitions for TCIO file functions as well as REF directives. This file will not conflict with the I/O include files that come with the compiler.

Functions in TCIO:

- Open a file.

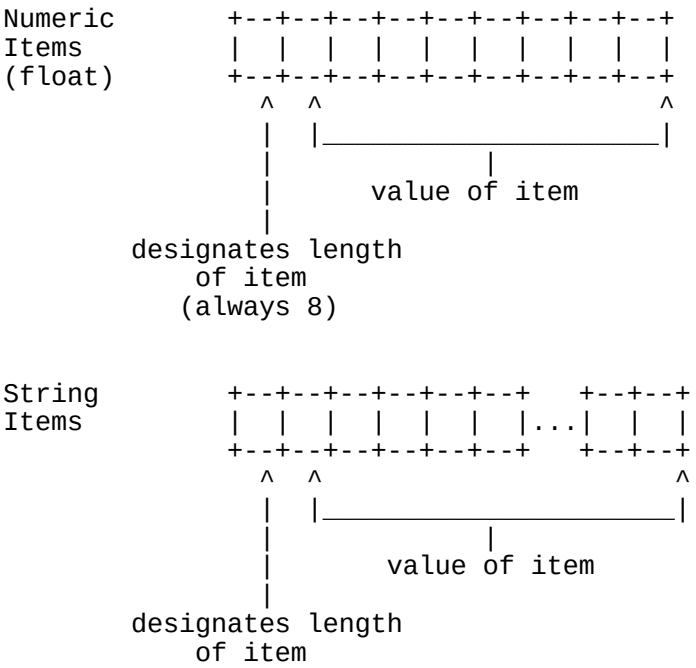
```
int filptr;
char name[],access,fsize;

filptr=topen(name,access,fsize);
```

Name contains a valid filename in lower or upper case, access contains all the access information for the file (see include file for access information) and fsize contains the size in bytes of the file. If the file is opened for input the fsize parm may be set to zero (undefined) to tell the system to determine the size of the file. If the open is successful filptr will contain a value greater than 0, otherwise it will contain the error code (see the include file for open errors).

DATA TYPE INFORMATION

DISPLAY-type data has the same form as data entered from the keyboard. If you wish to use this file between 'C' and Basic then the computer knows the length of each data item in a DISPLAY-type record by the comma separators placed between items (basic will locate the items automatically but in 'C' you have to locate the items yourself).

INTERNAL-type data has the following form:



The computer knows the length of each INTERNAL-type item by interpreting the one-position length indicator at the beginning of each item (basic will interpret the item automatically when inputting or printing but you must do the interpretation of items yourself when in 'C').

No validation of INTERNAL-type data-items is performed. All numeric items must be 9 positions long (8 digits plus one position which specifies the length) and must be valid representations of float-point number (see floating point library).

For more information refer to pages II-126 and II-132 of the TI-99/4A *User's Reference Guide*. Just keep in mind that Basic will do all of the translation effort while you must do the translation if in 'C'.

Example 1.

```
filptr=topen("DSK1.MYFILE",UPDATE+INTERNAL+RELATIVE+FIXED,100)
if(filptr<1) {
  /* error */
  }
```

Example 2. (open as an undefined file size)

```
filptr=topen("DSK1.MYFILE",INPUT+DISPLAY+SEQUENTIAL+FIXED,0)
if(filptr<1) {
  /* error */
  }
```

- Read a file.

```
int eof,rec,filptr,size ;
char buff[];

eof=tread(buff,rec,filptr,&size);
```

Reads the file specified by filptr. If eof returns -1 then it is a valid end of file condition, if it returns -2 then it is an error. Buff is a character buffer, this buffer should be one byte more than the size of the file, for example if you open the file with a size of 80 then the definition of buff should say buff[81]. If you've opened the file with an undefined size you should ensure the buffer size will be big enough to hold the longest record, to be safe use buff[256]. Rec is used to perform direct record reads, if the file is open with access relative and rec is -1 then the file will be read sequentially, otherwise the record specified in rec is read. Size will contain the actual number of bytes read from the disk.

Example 1.

```
/* Read the file sequentially */

while(1) {
  eof=tread(buff,RELSEQ,filptr,&size);
  if(eof==TEOF)
    break;
  else if(eof==TERR) {
    puts("File Error");
    break;
    }
  /* Display Buffer */
  puts(buff);
  }
```

Example 2.

```
/* Read using directed reads */
rec = 0;
while(1) {
  eof=tread(buff,rec++,filptr,&size);
  if(eof==TEOF)
    break;
  else if(eof==TERR) {
    puts("File Error");
```

```
    break;
    }
  /* Display Buffer */
  puts(buff);
  }
```

- Write to a file.

```
int eof,rec,filptr,size;
char buff[];

eof=twrite(buff,rec,filptr,size);
```

Writes to the file specified by filptr. If eof returns -2 then there is a file error. Buff is a character buffer. If the file is a relative file then rec will contain the record number of where to write the buffer to. If size is not equal to zero the write will perform a write of size bytes to the disk, otherwise it will write up to the null terminator of buff.

Example 1.

```
/* Write to sequential file */

while(there is data) {
  eof=twrite(buff,0,filptr,size);
  if(eof) {
    puts("File Error");
    break;
    }
  }
```

Example 2.

```
/* Write to relative file */

while(there is data) {
  rec = record to update
  eof=twrite(buff,rec,filptr,size);
  if(eof) {
    puts("File Error");
    break;
    }
  }
```

- Close a file.

```
int eof,filptr;

eof=tclose(filptr);
```

Closes the specified file, if an error occurs eof is set to -2.

Example 1.

```
/* Close a file */

eof=tclose(filptr);
if(eof)
  puts("File Error");
```

Disk 88. Contents of file TCIOI

```
/*****************************************/
/* Include file for TCIO file library     */
/*****************************************/

#asm
 REF    TOPEN,TCLOSE,TREAD,TWRITE
#endasm

/* File Type         */
#define SEQUENTIAL    0
#define RELATIVE      1

/* Record Type       */
#define FIXED         0
#define VARIABLE     16

/* Data Type         */
#define DISPLAY       0
#define INTERNAL      8

/* Mode of Operation */
#define UPDATE        0
#define OUTPUT        2
#define INPUT         4
#define APPEND        6

/* read relative file*/
/* sequentially. use */
/* this in place of  */
/* the record number */
#define RELSEQ       -1

/* Error & Eof Codes */
#define TERR         -2
#define TEOF         -1

/* Open Errors       */
/* for more info see */
/* page 299 in E/A.  */
#define BADDEV        0
#define PROTECTD     -1
#define BADATTR      -2
#define ILGALOP      -3
#define NOSPACE      -4
#define READEOF      -5
#define DEVERR       -6
#define NOFILE       -7
```

Disk 88. Contents of file TEST;C

```
/*******************************/
/* program: type               */
/* author : Tom Bentley        */
/* date   : Dec. 21, 1985      */
/*******************************/
/*******************************/
/* This program will TYPE any  */
/* display 80 files, either    */
/* fixed or variable, to your  */
/* screen. Press space bar to  */
/* hold the display.           */
/*******************************/

/* ensure these locations match your environment */
#include DSK2.CONIO
#include DSK3.TCIOI
#include dsk2.conv;c
main()
  {
    char filename[16],buffer[81],str[6];
    int  fp,eof,rec,rsiz;

    putchar(FF); /* home and clear screen */
    puts("  ************************************\n");
    puts(" *       T Y P E - A - F I L E       *\n");
    puts("************************************\n\n\n\n");
    puts("Type Filename >");
    gets(filename);

    /* lets try to open it as a variable length */
    /* file as it is more common.               */

    fp=topen(filename,INPUT+SEQUENTIAL+VARIABLE+DISPLAY,80);
    if(fp<1) /* can't open it, lets try fixed */
      fp=topen(filename,INPUT+SEQUENTIAL+FIXED+DISPLAY,80);
    if(fp<1) { /* i'll have to stop */
      puts("\n\n  bad filename!!\n\n");
      puts("file status = ");
      itod(fp,str,4);
      puts(str);
      puts("\n");
      exit(0);
      }
    putchar(FF);
    puts("To pause press <spacebar>.\n\n");
    eof=tread(buffer,0,fp,&rsiz);
    while(eof!=TEOF) {
      puts(buffer);
```

```
  puts("\n");
  poll(1);
  eof=tread(buffer,0,fp,&rsiz);
  }
tclose(fp);
}
```

Disk 89. Picasso

Version: 1.1 *Author*: Arto Heino
Requires: XB, EA *Language*: AL *Updated*: 01/05/88

A very useful graphics program out of Australia. Rivals some commercial products and includes font editor and drawing capabilities. Useful to anyone who uses graphics.

```
dskdir. v2.0. 12-dec-96
Disk name               = 0318-1-X
Sectors total           = 360
Sectors used            = 269
Sectors available       = 89
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHARS/O       7 DIS/FIX 80   >022 006
002 >003 DISKPRINT     5 PROGRAM      >028 004
003 >004 FONT-1        5 DIS/VAR 80   >02c 004
004 >005 FONT-10       5 DIS/VAR 80   >030 004
005 >006 FONT-2        5 DIS/VAR 80   >034 004
006 >007 FONT-3        5 DIS/VAR 80   >038 004
007 >008 FONT-4        5 DIS/VAR 80   >03c 004
008 >009 FONT-5        5 DIS/VAR 80   >040 004
009 >00a FONT-6        5 DIS/VAR 80   >044 004
010 >00b FONT-7        5 DIS/VAR 80   >048 004
011 >00c FONT-8        5 DIS/VAR 80   >04c 004
012 >00d FONT-9        5 DIS/VAR 80   >050 004
013 >00e LOAD          2 PROGRAM      >054 001
014 >00f MACDMP/O      9 DIS/FIX 80   >055 008
015 >010 P/LOADER      9 DIS/FIX 80   >05d 008
016 >011 PICASLOGO    57 PROGRAM      >065 056
017 >012 PICASOA      33 PROGRAM      >09d 032
018 >013 PICASOB      33 PROGRAM      >0bd 032
019 >014 PICASODOC    51 DIS/VAR 80   >0dd 050
020 >015 PICPAT/O      5 DIS/FIX 80   >10f 004
021 >016 XBFONTS       8 PROGRAM      >113 007
```

Disk 89. Contents of file PICASODOC

PICASSO PUBLISHER V1.1

(c) Arto Heino 1987

I hope this program fills the needs of those of us who like to create.

1. SYSTEM REQUIREMENTS

XB or EA or MM, 32K Memory, Disk System, Epson compatible printer and Joystick.

2. LOADING PROGRAM

Mini-Memory and Editor/Assembler:

```
OLD DSK1.LOAD
RUN
```

EXTENDED BASIC:

WILL AUTO-LOAD

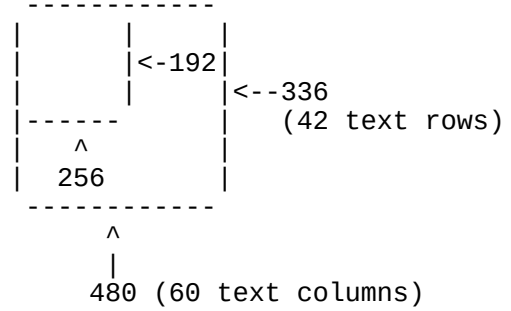
OR (MM, EA, XB)

```
CALL LOAD("DSK1.P/LOADER")
CALL LINK("PLOAD")
```

3. STARTING

Once the Title screen has loaded, press **FCTN 4** to clear the screen. Picasso Publisher defaults to the drawing mode, so you can start drawing straight away.

The viewing area of the screen is a small window onto a large 480 × 336 area, which can be scrolled by moving to the edge with the pointer. Your starting screen is at the top lefthand corner:



The large screen area is also where you load your text files. Each text char is 8 × 8. You have 42 lines of 60 columns of text input area, this will be important to remember when you prepare your text files for loading.

4. COMMANDS

B = BRUSHES
C = CIRCLE
D = DRAW
F = FILL WINDOW ACTIVE
G = GET FILE
I = INVERT WINDOW ACTIVE
K = REVERSE WINDOW ACTIVE
L = LINES
M = MIRROR WINDOW ACTIVE
N = TEXTURES
O = BOX
P = PRINTOUT
R = RAYS
S = SAVE FILE
T = TEXT INPUT
U = TOGGLE ON/OFF
W = WINDOW
Z = TOGGLE MOVE/COPY WINDOW ACTIVE
+ = JOYSPEED FAST
- = JOYSPEED SLOW
1 = SAVE FONT
2 = LOAD FONT
8 = UNDO
FCTN 4 = CLEAR SCREEN
FCTN = = FILE UTIL MENU
ENTER = FONT EDITOR
SPACE = ZOOM

5. COMMANDS in detail

B = BRUSHES

You can choose from 32 different brush shapes. After pressing 'B' move your joystick left or right to select which brush, when you have made your choice press the joystick button and you will return to where you were before pressing 'B'.

C = CIRCLE

Place your pointer at the center of where you want the circle then press the joystick button. You can now vary the size you want by moving the joystick away from the center. When you are satisfied press the joystick button to draw the circle. Press 'D' to abort.

D = DRAW

This will return you to the normal drawing mode from what ever function you were performing previously.

F = FILL

You must have the WINDOW mode active and an area framed. This will only fill with solid black.

G = GET FILE

This will give you a prompt to put in a file name. If you want to abort just press 'FCTN S'. The file type is DISPLAY VARIABLE 80, so be careful what you load in.

I = INVERT

You must have the WINDOW mode active and an area framed. This will turn the area enclosed upside down,

K = REVERSE

You must have the Window mode active and an area framed. This will turn OFF all dots that are on and turn ON all dots that are not.

L = LINES

After pressing 'L' choose the start position of line and press joystick button, now move the joystick until you have located the end position of the line, then press the joystick button. You exit this mode by pressing 'D'.

M = MIRROR

You must have the WINDOW mode active and an area framed. This will turn the area enclosed into the mirror image of itself.

N = TEXTURES

You can choose from 32 different textures. After pressing 'N' move your joystick left or right to which texture, when you have made your choice press the joystick button and you will return to where you were before pressing 'N'.

O = BOX

After pressing 'O' move your pointer to the position you want one of the BOX corners to be then press the joystick button. Moving the joystick you can expand the box into what ever size you want. To leave this mode press 'D'.

P = PRINTOUT

This will give you a prompt to put in the print DEVICE name. To abort just press **FCTN S**. Here is a list of the PRINTERS it will be compatible with:

EPSON FX80 graphics
BROTHER M1009
BROTHER M1109
IBM graphics
other EPSON compatible printers

R = RAYS

First position your pointer to where you want then press 'R'. Move the joystick around and press the joystick button when you want the rays to be drawn. Press 'D' to abort function.

S = SAVE FILE

This will give you a prompt to put in a file name. If you want to abort just press 'FCTN S'. The file type is DISPLAY VARIABLE 80, so be careful what name you use.

T = TEXT INPUT

You can enter text in either upper or lower case. You can pixel adjust where your text line by moving the joystick up or down. Function keys 'S D E X' are active. If you press enter the pointer will return to the left most column. To exit press fire button.

U = TOGGLE ON/OFF

This will toggle your pixel status either ON or OFF. Also works in ZOOM mode.

W = WINDOW

Position the pointer to the desired location then hold the joystick button while moving the joystick. A wire frame window will outline part of your picture. To abort function press 'D'.

Z = TOGGLE MOVE/COPY

Once you have enclosed an area with the window function, move the pointer inside the frame and press 'Z' to toggle from 'M' to 'C' on the little hand.

To MOVE or COPY a section hold the joystick button while moving the joystick to your new position. Release the joystick button then press it again to activate function. You can only MOVE/COPY in the visible screen area, if you go to the edge, the screen will scroll and your original frame will shift its position. To abort frame press the joystick button outside the window.

+ = JOYSPEED FAST

This will speed up the operation of the joystick if it has been slowed down by pressing '-'. WARNING: If you have pressed it too often the pointer will seem to stop, just press '-' till you can move it again.

- = JOYSPEED SLOW

This function is handy if you want to draw carefully. To speed up again just press '+'. You have 65000 increments to slow down with.

1 = SAVE FONT

This will give you a prompt to put in your file name. The format of the file is DISPLAY/VARIABLE 80.

2 = LOAD FONT

This will give you a prompt to put in your file name. The format of the file is DISPLAY/VARIABLE 80.

8 = UNDO

This will undo anything that has been done on your screen since SCROLLING, SAVING OR LOADING.

ENTER = FONT EDITOR

The screen will display the ASCII CHARACTERS in a magnified form. To turn pixels ON or OFF press the joystick button. To locate any char press that letter on the keyboard. To abort press enter. You can define your own texture pattern by choosing the last char.

SPACE = ZOOM

The screen will display the screen area in a magnified form. You can move all over the bitmap screen in this mode. Press 'U' to toggle the pixel status. To abort press 'D'.

FCTN 4 = CLEAR SCREEN

This will only clear the visible part of the screen.

FCTN + = FILE UTILITY MENU

1 = LOAD TI-Writer FILE
This will load 42 lines of 60 CHARS wide starting at your line option.

2 = LOAD GRAPHIC FILE
This will load a TI-ARTIST file with suffix '_P' or any graphic file with 24 SECTORS. The picture will be placed over whatever was in the displayed screen area. If you don't want the graphic you just loaded in press '8'(undo) to clear it.

3 = OVERLAY GRAPHIC
This will do the same as option 2 but will overlay the picture transparently over what was on the screen previously. You cannot undo this function.

4 = CATALOG DISK
This option will give you a choice of disk 1-4. The file information includes the file names and their sizes and the disk name and the amount used. Maximum files visible is 44. Press the space bar to return to file menu.

5 = RETURN TO PICASSO
This will return you to the drawing mode.

6 = EXIT PICASSO PUBLISHER
This will QUIT the program and return you to the TI TILE SCREEN. All data in memory will be lost.

6. PREPARING FILES FOR USE

When you want to create a particular type of page with text and graphics use the following method:

FIRST — Adjust your TI-Writer editor tab setting to 60 max chars. To make an A4 size page use 84 lines. Use the powerful editor functions to lay out your page. Remember if you are going to load a picture file later, do not put your picture overlapping line 42.

SECOND — If you want to use any GRAPHX files then you will have to convert them with either MAX-RLE or TI-ARTIST. Convert your files into TI-ARTIST format so you can use the '_P' file for loading into PICASSO.

THIRD — XB basic screens can be dumped using an assembly routine outlined in ENHANCED XB program.

FOURTH — Load PICASSO and CLEAR SCREEN now select your font choice to load then press **FCTN +** to select file utilities. Load your TEXT file starting at line 1. Select the area you want the graphic file to load use either Load or Overlay graphic option. You should have half an A4 page almost ready, SAVE your file before you go any further. Now load the second half of your Text file starting at line 43. Do the same procedure to load any graphic files. SAVE it. You should now have 2 files that contain your page. PRINT them or do some artistic adjustments at your leisure.

7. XB UTILITIES

I have included 2 extra programs on the disk:

DISKPRINT

This will print your PICASSO files from disk straight to the printer. If you don't have an EPSON type printer you can change the printer codes to suit.

XBFONTS

Contained in this program are some programming examples on how to

SAVE XB CHARS INTO PICASSO FONTS
LOAD PICASSO FONTS INTO XB
CREATE PICASSO FILE FROM XB

Also included on the disk is an assembly file called 'CHARS/O'. This file contains 3 links to use in XB.

```
CALL LINK("CHRSET",A$,B$,C$,D$,E$)
CALL LINK("CHAR",CHRNUM,STR$)
CALL LINK("CHRPAT",CHRNUM,LENG,RET$)
```

You can load your Font from PICASSO into XB just to see what it looks like.

MACDMP/O

If you have any BITMAC files you can use this utility to change the format. e.g.

```
CALL LOAD("DSK1.MACDMP/O")
CALL LINK("MACDMP","DSK1.MACFILE","DSK1.GRAPHIC")
```

The new file format can be now be used from your FILE OPTIONS MENU.

8. ENHANCED XB

I have added another call link so you can use screens created with it into PICASSO Load Graphic File option.

```
CALL LINK("ARTDMP","DSK?.NAME")
```

I am only asking $20 for it. If you want the program by Post add $2.50 postage & handling.

AUTHOR: Arto Heino
35/8 Guernsey Ave.,
MINTO, 2566,
N.S.W., AUSTRALIA

Disk 90. XB Games

Version:
Requires: XB
Author: Walt Howe, others
Language: XB
Updated: 01/05/88

Three "puzzle/thinking" type games. Chainlink is an addicting version of solitaire. Segregation and Perplex are well executed pattern matching games. "Hours of fun. . ."

```
dskdir. v2.0. 12-dec-96
Disk name               = BCSXBGAMES
Sectors total           = 360
Sectors used            = 272
Sectors available       = 86
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CHAIN/DOC    31 DIS/VAR 80   >022 030
002 >003 CHAINLINK5   42 PROGRAM      >040 041
003 >004 DEAL1         2 INT/FIX106   >069 001
004 >005 DEAL10        2 INT/FIX106   >06a 001
005 >006 DEAL2         2 INT/FIX106   >06b 001
006 >007 DEAL3         2 INT/FIX106   >06c 001
007 >008 DEAL4         2 INT/FIX106   >06d 001
008 >009 DEAL5         2 INT/FIX106   >06e 001
009 >00a DEAL6         2 INT/FIX106   >06f 001
010 >00b DEAL7         2 INT/FIX106   >070 001
011 >00c DEAL8         2 INT/FIX106   >071 001
012 >00d DEAL9         2 INT/FIX106   >072 001
013 >00e PERPLEX      21 PROGRAM      >073 020
014 >00f PERPLEX-LI   23 DIS/VAR 80   >087 022
015 >010 PERPLEXDOC   18 DIS/VAR 80   >09d 017
016 >011 SEGCHAR_D     4 INT/FIX 17   >0ae 003
017 >012 SEGINST1_X   37 PROGRAM      >0b1 036
018 >013 SEGINST2_X   33 PROGRAM      >0d5 032
019 >014 SEGREGATE    13 PROGRAM      >0f5 012
020 >015 SEGRGATE_X   30 PROGRAM      >101 029
```

Disk 90. Contents of file CHAIN/DOC

CHAINLINK MASTER SOLITAIRE, Version 5.0

CHAINLINK5 is a very challenging solitaire card game that is playable on the TI-99/4A or the Myarc 9640. It may seem difficult at first, but with good play, it can be solved one-third to one-half of the time. The program package includes 10 saved games that are guaranteed to be solvable, and the ability to randomly generate thousands more.

This program package is fairware. It may be freely copied and passed along to others, or placed on bulletin boards and information services. It may under no circumstances be sold for profit, except that legitimate non-commercial user groups may charge a reasonable fee for its copying and distribution not to exceed $3 except with the expressed permission of the author. The package may be distributed in only two forms: either as an archived (packed) file on telecommunications media or in unpacked form on a single disk. Whether distributed as a packed file or as separate files on a disk, it must contain all of the following files: *README (this file), CHAINLINK5, and the ten saved games with the filenames DEAL1, DEAL2, . . . through DEAL10. Additional saved deals may be added to the disk or pack, but no files may be removed.

The program may be tried without charge, but continued use over a period longer than two days or trying more than two of the saved games incurs an obligation to pay for it. Send $6 to Walter Howe, 43 S. Chelmsford Rd., Westford, MA 01886 if you decide to pay for it. Those who have previously paid for earlier versions of CHAINLINK are considered to have paid for this version, too.

CHAINLINK5 is written in TI Extended BASIC with embedded assembly code. It will also run with Mechatronic XBASIC, Myarc XB II, and DataBioTics Super Extended BASIC. It runs on the Myarc 9640 with any of the above XBASICs, but will not run in the first release of Advanced BASIC (a modification of XB II). When the final Advanced BASIC is released, either this version will run, or a modification will be written that will run.

Enhancements over earlier versions include several improvements in the playability of the game, the ability to save and load deals, and elimination of the problem that always gave the same deals if you autoloaded it by naming the program LOAD or by loading with the RUN command.

Complete rules for play are available through on-screen menus. CHAINLINK will not allow you to make illegal moves, so cheating, accidental or intentional, is difficult. If you do make a wrong (but legal) move and immediately regret it, there is an OOPS! key. Press Function 1 (F1 on the 9640) to take back a move.

PLAY OF THE GAME:

If convenient, it is a good idea to load the program before reading these rules. The same rules in a somewhat more abbreviated form are available on screen.

The object of the game is to play all 52 cards, one at a time, into four piles at the top of the screen, one for each suit. The deal consists of 13 columns of cards 4 cards deep. All cards are visible. All plays are made one card at a time from the bottom of one column of cards to the bottom of another. A card may only be played on the card one higher or one lower in the same suit, for example, the 4 of hearts may only be played directly on the 3 of hearts or the 5 of hearts at the bottom of another column. The cards rank in order from ace (low) to king (high). All bottom cards are eligible to be moved at any time. Due to screen limitations, no column may be longer than 10 cards in length; you should plan your play so that this does not prevent you from solving a deal. Blank columns may only be filled by kings. To move a card, you press the letter that appears at the top of the column containing the card you want to move, followed by the letter of the column you want to move it to.

The four top piles are started with the aces whenever they become available and top play proceeds with the 2, 3, 4, etc. in order. Any time a card is open that can be played to the top piles, pressing **ENTER** will make all possible such moves automatically.

At the beginning of play, you are given the option of starting the deal that appears on screen, dealing a new game, reading the rules, saving the current deal, or loading a saved deal. Whichever you select, you are prompted for follow-on actions as necessary.

During play, several Function keys are enabled to let you start the deal over again (F9), start a new deal (F8), take back a move (F1), save a deal to disk (F2), or load a saved deal (F3). Fctn = is not enabled; to QUIT, select F8 or F9 and follow the prompts.

If you have trouble solving a game, try any or all of the saved games named DEAL1 through DEAL10. You can learn a lot about the strategy of the game by trying these, knowing that they can be solved.

I hope you enjoy the game, but I must give you fair warning. Some people have become thoroughly addicted to CHAINLINK and waste many hours playing it. If you find yourself caught by the game, just remember that you are sharpening your powers of logic and reasoning through steady play.

Disk 90. Contents of file PERPLEX-LI

```
100 REM PERPLEX
110 REM TI-99/4A
120 REM EXTENDED BASIC
130 REM WESLEY R RICHARDSON
140 REM BLUEGRASS 99 COMPUTER SOCIETY,
    INC.
150 REM NOVEMBER, 1985
160 DIM A$(52)
170 CALL CLEAR
180 CALL SCREEN(6)
190 CALL CHARPAT(80,A$(0),81,A$(1),82,A
    $(2),83,A$(3),84,A$(4),85,A$(5),86,
    A$(6),87,A$(7),88,A$(8))
200 CALL CHAR(43,A$(0),33,A$(1),35,A$(2
    ),36,A$(3),37,A$(4),40,A$(5),41,A$(
    6),42,A$(7),45,A$(8))
210 CALL CHAR(80,"0",89,"0",97,"0",104,
    "0",113,"0",120,"0")
220 CALL COLOR(0,16,1,1,16,1,2,16,1,3,1
    6,1,4,16,1,5,16,1,6,16,1)
230 CALL COLOR(7,1,14,8,1,2,9,1,16,10,1
    ,3,11,1,12,12,1,10)
240 DISPLAY AT(2,11):"+E#+LE-"
250 M=0
260 CALL HCHAR(5,2,80,9)
270 CALL HCHAR(16,23,80,9)
280 CALL HCHAR(5,11,89,12)
290 CALL HCHAR(16,11,89,12)
300 CALL HCHAR(5,23,97,9)
310 CALL HCHAR(16,2,97,9)
320 CALL HCHAR(10,7,113,8)
330 CALL HCHAR(11,7,113,8)
340 CALL HCHAR(10,19,120,8)
350 CALL HCHAR(11,19,120,8)
360 CALL VCHAR(6,2,104,10)
370 CALL VCHAR(6,31,104,10)
380 A$(3)="P" :: A$(15)="P" :: A$(20)="
    P" :: A$(32)="P" :: A$(42)="P" :: A
    $(45)="P"
390 A$(2)="Y" :: A$(14)="Y" :: A$(19)="
    Y" :: A$(35)="Y" :: A$(48)="Y"
400 A$(6)="Y" :: A$(18)="Y" :: A$(23)="
    Y" :: A$(31)="Y" :: A$(41)="Y"
410 A$(5)="a" :: A$(17)="a" :: A$(22)="
    a" :: A$(34)="a" :: A$(44)="a" :: A
    $(47)="a"
420 A$(4)="h" :: A$(16)="h" :: A$(21)="
    h" :: A$(33)="h" :: A$(43)="h" :: A
```

```
    $(46)="h"
430 A$(7)="q" :: A$(8)="q" :: A$(9)="q"
     :: A$(10)="q" :: A$(11)="q" :: A$(
    12)="q"
440 A$(25)="q" :: A$(26)="q" :: A$(27)=
    "q" :: A$(28)="q" :: A$(29)="q" ::
    A$(30)="q"
450 A$(1)="x" :: A$(36)="x" :: A$(37)="
    x" :: A$(38)="x" :: A$(39)="x" :: A
    $(40)="x"
460 A$(13)="x" :: A$(24)="x" :: A$(49)=
    "x" :: A$(50)="x" :: A$(51)="x" ::
    A$(52)="x"
470 GOTO 1530
480 REM MAIN LOOP
490 DISPLAY AT(20,1):"  $=LEF% CC*    J
    =#IGH% CC*"
500 DISPLAY AT(21,1):"  D=LEF%  C*    K
    =#IGH%  C*"
510 DISPLAY AT(23,3):"! %O #E$%A#% O# E
    ND"
520 DISPLAY AT(18,11):"MO)E$";M
530 CALL KEY(0,K,S)
540 IF S=0 THEN 530
550 IF K=83 THEN 610
560 IF K=68 THEN 670
570 IF K=74 THEN 730
580 IF K=75 THEN 980
590 IF K=81 THEN 1640
600 GOTO 530
610 REM LEFT CCW
620 FOR I=6 TO 30 STEP 6
630 GOSUB 1370
640 A$(I-5)=A$(0)
650 NEXT I
660 GOTO 1510
670 REM LEFT CW
680 FOR I=1 TO 25 STEP 6
690 GOSUB 1230
700 A$(I+5)=A$(0)
710 NEXT I
720 GOTO 1510
730 REM RIGHT CCW
740 I=40
750 GOSUB 1410
760 A$(36)=A$(1)
770 A$(1)=A$(0)
780 I=35
790 GOSUB 1410
800 A$(31)=A$(7)
810 A$(7)=A$(0)
```

```
 820 I=44
 830 GOSUB 1450
 840 A$(41)=A$(30)
 850 A$(30)=A$(19)
 860 A$(19)=A$(0)
 870 I=48
 880 GOSUB 1450
 890 A$(45)=A$(18)
 900 A$(18)=A$(25)
 910 A$(25)=A$(0)
 920 I=52
 930 GOSUB 1450
 940 A$(49)=A$(24)
 950 A$(24)=A$(13)
 960 A$(13)=A$(0)
 970 GOTO 1510
 980 REM RIGHT CW
 990 I=36
1000 GOSUB 1270
1010 A$(40)=A$(1)
1020 A$(1)=A$(0)
1030 I=31
1040 GOSUB 1270
1050 A$(35)=A$(7)
1060 A$(7)=A$(0)
1070 I=41
1080 GOSUB 1310
1090 A$(44)=A$(19)
1100 A$(19)=A$(30)
1110 A$(30)=A$(0)
1120 I=45
1130 GOSUB 1310
1140 A$(48)=A$(25)
1150 A$(25)=A$(18)
1160 A$(18)=A$(0)
1170 I=49
1180 GOSUB 1310
1190 A$(52)=A$(13)
1200 A$(13)=A$(24)
1210 A$(24)=A$(0)
1220 GOTO 1510
1230 REM ROTATE 6 CW
1240 GOSUB 1270
1250 A$(I+4)=A$(I+5)
1260 RETURN
1270 REM ROTATE 5 CW
1280 GOSUB 1310
1290 A$(I+3)=A$(I+4)
1300 RETURN
1310 REM ROTATE 4 CW
```

```
1320 A$(0)=A$(I)
1330 A$(I)=A$(I+1)
1340 A$(I+1)=A$(I+2)
1350 A$(I+2)=A$(I+3)
1360 RETURN
1370 REM ROTATE 6 CCW
1380 GOSUB 1410
1390 A$(I-4)=A$(I-5)
1400 RETURN
1410 REM ROTATE 5 CCW
1420 GOSUB 1450
1430 A$(I-3)=A$(I-4)
1440 RETURN
1450 REM ROTATE 4 CCW
1460 A$(0)=A$(I)
1470 A$(I)=A$(I-1)
1480 A$(I-1)=A$(I-2)
1490 A$(I-2)=A$(I-3)
1500 RETURN
1510 REM DISPLAY NEW MOVE
1520 M=M+1
1530 DISPLAY AT(7,2):A$(21);A$(15);" ";A
     $(3);A$(3);" ";A$(20);A$(14);" ";A$
     (2);A$(2);" ";A$(19);A$(19)
1540 DISPLAY AT(7,17):A$(35);A$(35);" ";
     A$(48);A$(44);" ";A$(34);A$(34);" "
     ;A$(47);A$(43)
1550 DISPLAY AT(8,2):A$(21);A$(27);" ";A
     $(9);A$(9);" ";A$(26);A$(26);" ";A$
     (8);A$(8);" ";A$(25);A$(13)
1560 DISPLAY AT(8,17):A$(40);A$(40);" ";
     A$(52);A$(52);" ";A$(39);A$(39);" "
     ;A$(51);A$(43)
1570 DISPLAY AT(10,2):A$(4);A$(10);" qqq
     qqqqq ";A$(7);A$(1);" xxxxxxxx ";A$
     (38);A$(33)
1580 DISPLAY AT(11,2):A$(4);A$(10);" qqq
     qqqqq ";A$(7);A$(1);" xxxxxxxx ";A$
     (38);A$(33)
1590 DISPLAY AT(13,2):A$(16);A$(28);" ";
     A$(11);A$(11);" ";A$(29);A$(29);" "
     ;A$(12);A$(12);" ";A$(30);A$(24)
1600 DISPLAY AT(13,17):A$(36);A$(36);" "
     ;A$(49);A$(49);" ";A$(37);A$(37);"
     ";A$(50);A$(46)
1610 DISPLAY AT(14,2):A$(16);A$(22);" ";
     A$(5);A$(5);" ";A$(17);A$(23);" ";A
     $(6);A$(6);" ";A$(18);A$(18)
1620 DISPLAY AT(14,17):A$(31);A$(31);" "
     ;A$(41);A$(45);" ";A$(32);A$(32);"
     ";A$(42);A$(46)
```

```
1630 GOTO 480
1640 DISPLAY AT(23,3):"
       "
1650 DISPLAY AT(19,3):"1 %O #E$%A#%, 2 %
     O CON%IN(E,3 %O END"
1660 CALL KEY(0,K,S)
1670 IF S=0 THEN 1660
1680 IF K=49 THEN 240
1690 IF K=50 THEN 480
1700 IF K=51 THEN END
1710 GOTO 1660
```

Disk 90. Contents of file PERPLEXDOC

PERPLEX

by WESLEY R. RICHARDSON

BLUEGRASS 99 COMPUTER SOCIETY, INC.

PERPLEX is a two dimensional graphical puzzle in which the player first scrambles the pieces and then attempts to restore them to their initial positions. The program was written for the TI-99/4A computer with Extended Basic, and requires a color TV or monitor. For use with a black and white TV, see the Remarks section.

PERPLEX has two types of pieces, those with two colors, called BI pieces, and those with three colors, called TRI pieces. There are 11 BI pieces and 10 TRI pieces. BI pieces can only move to BI locations, and TRI pieces can only move to TRI locations. Pieces on the left side of the puzzle are moved counter-clockwise (CCW) by pressing S, and clockwise (CW) by pressing D. Pieces on the right side are moved counter-clockwise by pressing J, and clockwise by pressing K. At any time, pressing Q will allow the user to restart at the initial position, to continue making moves, or to end the game.

Although PERPLEX looks simple at first, a relatively few number of moves will scramble the pieces sufficiently to require 100 or more moves to restore the initial position.

PERPLEX is similar to several other puzzles in which pieces are moved in groups rather than individually. One way to solve these puzzles is to use operators to systematically put pieces in their correct positions. An operator is defined as a specific series of moves. Particularly useful operators are those which, after completed, move some of the pieces, but do not disturb those pieces already in place. The starting position can be used to test an operator to find where the pieces move for a given sequence. A set of operators which includes the following is sufficient to solve PERPLEX:

2 BI cycle
3 BI cycle
3 TRI cycle
3 TRI twist
2 pair TRI cycle

For example, a 3 TRI cycle operator will switch the locations of 3 TRI pieces, but after it is completed, will return all other pieces to their positions prior to the operator.

PERPLEX is frustrating and perplexing, but it will also be fun and rewarding as new operators are discovered, and as a method for restoring the pieces to their initial position is found.

REMARKS

The program uses a one dimensional string array to store a character which corresponds to each color in the pieces. The six characters are from six different color sets, and have the foreground set to transparent, and the background set to the color for that position. Movements are made by interchanging the locations of these characters, and then printing the array to the screen in the pattern of the puzzle.

To use the program with a black and white TV, the character background colors for color sets 7-12 may need to be changed so that those colors can be distinguished from each other. The character patterns for characters 80, 89, 97, 104, 113, and 120 may also be redefined to unique shapes in line 210, to identify them.

LISTING ANNOTATIONS

Line Nos.	*Description*
100-150	Program header
160-250	Initialize characters
160-370	Display initial pattern
380-470	Initialize array
480-600	Main input loop
610-660	Left CCW
670-720	Left CW
730-970	Right CCW
980-1220	Right CW
1230-1260	Rotate 6 pieces CW
1270-1300	Rotate 5 pieces CW
1310-1360	Rotate 4 pieces CW
1370-1400	Rotate 6 pieces CCW
1410-1440	Rotate 5 pieces CCW
1450-1500	Rotate 4 pieces CCW
1510-1630	Display new move
1640-1710	Restart, continue, end

DIRECTORY OF VARIABLES

Variables	*Functions*
A$()	Piece color parts
I	Loop counter
K	Input character
M	Number of moves
S	Input status

Disk 91. 9640 Technical Material

Version:
Author: Paul Charlton, others
Requires: 9640
Language: AL
Updated: 3/11/88

A set of software XOP specifications, sample source code, a linker, and other material pertaining to assembly code development under MDOS.

```
dskdir. v2.0. 12-dec-96
Disk name               = 9640/TECH1
Sectors total           = 360
Sectors used            = 340
Sectors available       = 18
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CMDSTR/SRC    7 DIS/VAR 80 Y >022 006
002 >003 DSR/DOC     101 DIS/VAR 80   >028 100
003 >004 KEY/DOC       4 DIS/VAR 80   >08c 003
004 >00a LINK         13 PROGRAM    Y >106 012
005 >00b LINK/DOC      4 DIS/VAR 80 Y >112 003
006 >005 MANAGE/DOC   22 DIS/VAR 80   >08f 021
007 >006 MATH/DOC     35 DIS/VAR 80   >0a4 034
008 >00c MORE          6 PROGRAM      >115 005
009 >00d MORE_DOC     25 DIS/VAR 80   >11a 024
010 >00e MORE_S       56 DIS/VAR 80   >132 054 >00f 001
011 >007 NOTES/DOC     8 DIS/VAR 80   >0c6 007
012 >008 UTIL/DOC     14 DIS/VAR 80   >0cd 013
013 >009 VID/DOC      45 DIS/VAR 80   >0da 044
```

Disk 92. 9640 Terminal Emulators

Version:
Requires: 9640
Author: Charlton, Schroeder, etc.
Language: AL
Updated: 3/1/88

Three terminal emulator programs for the 9640 - Fast-term and Mass-transfer in 80 columns under GPL, and NOT-My-term under MDOS by Jim Schroeder.

```
dskdir. v2.0. 12-dec-96
Disk name               = 9640/TE
Sectors total           = 360
Sectors used            = 200
Sectors available       = 158
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >009 AUTODIAL/T    9 DIS/VAR 80   >0ab 008
002 >008 CHARA1        5 PROGRAM      >0a7 004
003 >002 FTERM/HLP    17 DIS/VAR 80   >022 016
004 >003 FTG          33 PROGRAM      >032 032
005 >004 FTGDOC/TXT    9 DIS/VAR 80   >052 008
006 >007 FTH          31 PROGRAM      >089 030
007 >005 MASS80       33 PROGRAM      >05a 032
008 >006 MASS81       16 PROGRAM      >07a 015
009 >00c NOTMY/DOC    10 DIS/VAR 80   >0d6 009
010 >00a NOTMYTERM    34 PROGRAM      >0b3 033
011 >00b NOTMYTERN     3 PROGRAM      >0d4 002
```

Disk 93. 9640 PR Base

Version: 2.1 *Author*: William Warren and M. Dodd
Requires: 9640 *Language*: AL *Updated*: 3/11/88

Modified version of PR Base for use on 9640. Does not use first 2 disk sectors. Still requires original PR Base documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = PRBASE
Sectors total           = 360
Sectors used            = 249
Sectors available       = 109
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00c -README-      4 DIS/VAR 80   >0d9 003
002 >002 -READTHIS-   28 DIS/VAR 80 Y >022 027
003 >003 CHAR          6 PROGRAM    Y >03d 005
004 >004 CRT:1        33 PROGRAM    Y >042 032
005 >005 CRT:2        26 PROGRAM    Y >062 025
006 >006 LOAD         15 PROGRAM    Y >07b 014
007 >007 PRB:1        33 PROGRAM    Y >089 032
008 >008 PRB:2        31 PROGRAM    Y >0a9 030
009 >009 PRBCONV-DS    7 DIS/VAR 80 Y >0c7 006
010 >00a PRBCONV-SS    6 DIS/VAR 80 Y >0cd 005
011 >00d PRBUTL/2*2   52 INT/VAR254   >0dc 051
012 >00b UTIL1         8 PROGRAM    Y >0d2 007
```

Disk 94. 9640 Routines by J.P. Hoddie

Version:
Requires: 9640

Author: J. Peter Hoddie
Language: AL

Updated: 3/1/88

A collection of JPH programs for the 9640 including 2 specialized (fast) MY-Word loaders, a very unique clock program, Extended BASIC mouse support, and other things.

```
dskdir. v2.0. 12-dec-96
Disk name               = 9640/JPH
Sectors total           = 360
Sectors used            = 115
Sectors available       = 243
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >007 ED            6 PROGRAM      >054 005
002 >00a MEGASORT96   14 PROGRAM      >06a 013
003 >003 MOUSE         7 DIS/FIX 80   >02a 006
004 >004 MOUSE/DOC    21 DIS/VAR 80   >030 020
005 >005 MOUSEDEMO     8 PROGRAM      >044 007
006 >006 MYLOAD/DOC   10 DIS/VAR 80   >04b 009
007 >008 RE            6 PROGRAM      >059 005
008 >00c SORT96/DOC   34 DIS/VAR 80   >0aa 033
009 >002 TIME          9 PROGRAM    Y >022 008
```

Disk 95. p-Code Units by Anders Persson

Version: *Author*: Anders Persson
Requires: p-Code *Language*: Pascal *Updated*: 3/14/88

A collection of programs that run in the UCSD p-System environment, primarily units, allowing for scrolling parts of the screen, windows, a call key routine, and many other routines that simulate XB routines.

```
dskdir. v2.0. 12-dec-96
Disk name               = SIGCODE
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 PASCAL      358 DIS/FIX128   >022 326 >003 031

p-System first block  = 0 (2)
Duplicate directories = yes
p-System disk name    = SIGCODE
p-System block count  = 180
p-System file count   = 14
Date of last boot     = 20-Nov-87

File name -----  Blk  Date ----  Strt  End   Last  Type
SYSTEM.CHARAC    002  24-Feb-85  >00a  >00b  >200  data
SYSTEM.MISCINFO  001  01-Oct-85  >00c  >00c  >0c2  data
SYSTEM.PASCAL    007  22-Oct-85  >00d  >013  >200  data
SYSTEM.LIBRARY   077  04-Mar-87  >014  >060  >200  exec
EXTRASCR.CODE    009  08-Oct-86  >061  >069  >200  exec
DFORMAT.CODE     007  09-Mar-87  >06a  >070  >200  exec
WELCOME.CODE     003  04-Mar-87  >071  >073  >200  exec
DISKMAP.CODE     003  23-Jan-86  >074  >076  >200  exec
ERRORUNIT.CODE   005  05-Dec-85  >077  >07b  >200  exec
MINIMEMORY.CODE  003  06-Dec-85  >07c  >07e  >200  exec
DISASSEM.CODE    010  08-Jan-86  >07f  >088  >200  exec
MINITEST.CODE    003  11-Feb-86  >089  >08b  >200  exec
ERRORTEST.CODE   003  12-Feb-86  >08c  >08e  >200  exec
EXTRATEST.CODE   003  04-Mar-87  >08f  >091  >200  exec
```

Disk 96. p-Code Units — Source 1

Version:
Requires: p-Code

Author: Anders Persson
Language: Pascal

Updated: 3/14/88

First disk of source code and documentation for disk 95.

```
dskdir. v2.0. 12-dec-96
Disk name               = SIG-TX1
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 PASCAL      358 DIS/FIX128   >022 326 >003 031

p-System first block  = 0 (2)
Duplicate directories = yes
p-System disk name    = SIG-TX1
p-System block count  = 180
p-System file count   = 11
Date of last boot     = 20-Nov-87

File name -----  Blk  Date ----  Strt  End   Last  Type
EXTRAPAS.TEXT    024  08-Oct-86  >00a  >021  >200  text
EXTRAASM.TEXT    014  02-Jun-86  >022  >02f  >200  text
ERRORASM.TEXT    016  05-Dec-85  >030  >03f  >200  text
ERRORPAS.TEXT    006  05-Dec-85  >040  >045  >200  text
ERRORTEST.TEXT   006  12-Feb-86  >046  >04b  >200  text
EXTRATEST.TEXT   006  04-Mar-87  >04c  >051  >200  text
ERRORINFO.TEXT   010  04-Mar-87  >052  >05b  >200  text
EXTRAINFO.TEXT   020  20-Nov-87  >05c  >06f  >200  text
FORMATTER.TEXT   014  09-Mar-87  >070  >07d  >200  text
FORMASM.TEXT     012  09-Mar-87  >07e  >089  >200  text
FILEPRINT.TEXT   018  13-Nov-87  >08a  >09b  >200  text
```

Disk 96. Contents of file EXTRAPAS.TEXT

```
unit extrascreen;
  (* Additional screen in- and output routines.
     A-DATA 861003 *)

interface

type
  charset = set of char;

var
  (* Variables set up by the unit to contain *)
  digit,        (* '0'..'9' *)
  numeric,      (* digit and '-','+','.','E','e' *)
  ualpha,       (* 'A'..'Z' *)
  abortkeys,    (* none *)
  terminator:charset;   (* ENTER, up arrow, down arrow *)

function keyscan(keyboard:integer):integer;
  (* Reads the keyboard on the fly *)
function bufscan:integer;
  (* Reads the next char from the buffer, if any *)
function readint:integer;
  (* Reads an integer from the current location *)
function readreal:real;
  (* Reads a real from the current location *)
function validint(size:integer; valider:charset):integer;
  (* As readint, but with validation.
     Accepts characters in valider only.
     Size characters accepted, unless size=0. That means no limit *)
function validreal(size:integer; valider:charset):real;
  (* Readreal with validation *)
procedure validstring(var result:string; size:integer; valider:charset);
  (* String input with validation *)
function lastchar:char;
  (* Gives the last character already taken from the buffer. *)
function goodinput:boolean;
  (* Returns true if the last input was OK *)
function abortinput:boolean;
  (* Returns true if last input was aborted *)
function getchar(x,y:integer):char;
  (* Returns the character at the specified screen location.
     Reads 80 col screen. *)
procedure hchar(x,y,len:integer; ch:char);
  (* Like HCHAR in BASIC. 80 col screen *)
procedure vchar(x,y,len:integer; ch:char);
  (* Like VCHAR in BASIC. *)
procedure newline(left,right,top,bottom :integer);
  (* Performs a writeln within the given area. *)
```

```
procedure findxy(var x,y :integer);
  (* The opposite of gotoxy. Returns current print position. *)

implementation

uses
  misc;

var
  abortresult,          (* Internal global. True if last input was aborted *)
  inputresult:boolean;  (* Internal global. True if last conversion was OK *)


(* Assembly support *)
function kscan(keyboard:integer):integer;    external;
function bscan:integer;  external;
function getlast:char;   external;
procedure horchar(x,y,len:integer; ch:char); external;
procedure verchar(x,y,len:integer; ch:char); external;
procedure scroll(left,right,top,bottom :integer);       external;


function keyscan;
  (* Realtime keyboard scanning *)

begin
  keyscan := kscan(keyboard);
end; (* keyscan *)


function bufscan;
  (* Buffer scanning *)

begin
  bufscan := bscan;
end; (* bufscan *)


procedure findxy;
  (* Returns current print position. *)

  function peek(addr :integer) :integer;
    (* Returns value at addr *)

  type
    dual = record
      case boolean of
        true  :(int :integer);
        false :(ptr :^integer);
    end; (* dual *)
```

```
  var
    window :dual;

  begin
    window.int := addr;
    peek := window.ptr^;
  end; (* peek *)

begin (* findxy *)
  x := peek(11458);     (* Addr of current column *)
  y := peek(11456);     (* Current row *)
end; (* findxy *)


procedure newline;
  (* Partial scrolling *)

var
  x,
  y :integer;

begin
  findxy(x,y);
  if y<bottom then
    gotoxy(left,y+1)
  else
  begin
    scroll(left,right,top,bottom);
    gotoxy(left,bottom);
  end;
end; (* newline *)


function lastchar;
  (* Returns the last character already taken from the buffer *)

begin
  lastchar := getlast;
end; (* lastchar *)


procedure vchar;

begin
  verchar(x,y,len,ch);
end; (* vchar *)


procedure hchar;
```

```
begin
  horchar(x,y,len,ch);
end; (* hchar *)


function getchar;
  (* Like GCHAR in BASIC. Reads 80 col screen. *)

const
  screenbase = 8192;

type
  dualpoint = record
    case boolean of
      true :(int:integer);
      false:(ptr:^char);
    end;

var
  point:dualpoint;

begin  (* getchar *)
  point.int := y*80+x+screenbase;
  getchar := point.ptr^;
end; (* getchar *)


function str_int
  (* Converts a string to an integer. Returns 0 if conversion was impossible *)

  (instring:string      (* String with numeric equivalent *)
  ):integer;

const digits = '0123456789';

var
  firstdigit,
  lastdigit,
  i,
  accum   :integer;   (* Accumulator used during conversion. *)
  negative:boolean;   (* Sign flag. *)

begin (* str_int *)
  negative := false;
  firstdigit := break(instring,digits);

  if firstdigit>0 then  (* valid string *)
  begin
    (* Negative? *)
    if firstdigit>1 then
```

```
      negative := instring[firstdigit-1]='-';

    delete(instring,1,firstdigit-1);  (* Get rid of chars in front of number *)
    lastdigit := span(instring,digits);

    if lastdigit=0 then
      lastdigit := length(instring)
    else
      lastdigit := lastdigit-1;

    (* Max five digits allowed *)
    if lastdigit>5 then
      lastdigit := 5;

    (* Calculate value of input *)
    accum := 0;
    for i := 1 to lastdigit do
      accum := accum*10+ord(instring[i])-48;
    if negative then
      accum := -accum;
    str_int := accum;
    inputresult := true;
  end (* if-then *)
  else
  begin  (* Fatal input error. Conversion impossible. *)
    str_int := 0;
    inputresult := false;
  end; (* if-else *)
end; (* str_int *)


procedure valid_string;
  (* Reads a string with validation *)

var
  count   :integer;
  ch      :char;

begin
  if size=0 then
    size := 80;
  count := 0;
  abortresult := false;

  (*$R-*)
  repeat
    read(keyboard,ch);
    if getlast in valider then       (* If acceptable then echo *)
    begin
      if count<size then
        count := count+1;
```

```
      result[count] := ch;
      write(ch);
      if count=size then
        write(chr(8));
    end
    else if (ch=chr(136)) and (count>0) then
    begin
      if count<size then
        write(chr(8));
      write(' ');
      write(chr(8));
      count := count-1;         (* If not acceptable, check for left arrow *)
    end;
    if getlast in abortkeys then
      abortresult := true;
  until (getlast in terminator) or abortresult;
  result[0] := chr(count);
  (*$R+*)
end; (* valid_string *)


function valid_int;
  (* Reads integer with validation *)

var
  temp:string;

begin
  valid_string(temp,size,valider);
  valid_int := str_int(temp);
end; (* valid_int *)


function readint;
  (* Reads integers *)

var
  temp:string;

begin
  readint := valid_int(0,['-','+','0'..'9']); (* Take string first *)
end; (* readint *)


function str_real(instring:string):real;
  (* Converts a string to a real *)

type
  modetype = (started,lookfrac,lookexp);
```

```
var
  mode         :modetype;
  mantnegative,
  expnegative  :boolean;
  accum,
  factor       :real;
  current,                  (* Pointer to processed character *)
  exponent,
  magnitude,
  temp         :integer;
  lookahead,
  ch           :char;

  procedure getlookahead;
    (* Reads the lookahead character *)

  begin
    if (current+1)>length(instring) then
      lookahead := chr(0)
    else
      lookahead := instring[current+1];
  end; (* getlookahead *)

  function nextchar:char;
    (* Returns next character to process *)

  begin
    nextchar := lookahead;
    if lookahead<>chr(0) then
    begin
      current := current+1;
      getlookahead;
    end;
  end; (* nextchar *)

begin (* str_real *)
  mantnegative := false;
  expnegative := false;
  inputresult := false;

  accum := 0;
  exponent := 0;
  str_real := 0;        (* Proper value in case of aborting error *)

  (* Get rid of preceeding spaces *)
  if length(instring)>0 then
    temp := scan(length(instring),<>' ',instring[1])
  else
    exit(str_real);     (* No string is wrong *)
  if temp=length(instring) then  (* Only spaces *)
    exit(str_real);
```

```
if temp<>0 then       (* Cut off spaces *)
  instring := copy(instring,temp+1,length(instring)-temp);

uppercase(instring,instring);  (* Get rid of 'e' *)
current := 0;         (* Processing pointer *)
getlookahead;
mode := started;

while lookahead in ['+','-'] do
begin
  ch := nextchar;
  if ch='-' then
    mantnegative := not mantnegative;
end;

while lookahead in ['0'..'9'] do
begin
  ch := nextchar;
  accum := accum*10+ord(ch)-48;
  inputresult := true;
end; (* while *)

(* End of integer mantissa part. See what follows *)
case lookahead of
  '.' : mode := lookfrac;     (* Look for fractions next *)
  'E' : mode := lookexp;      (* Look for exponent *)
end; (* case *)
(* Remove the character (if there is any) *)
if lookahead<>chr(0) then
  ch := nextchar;

if mode=lookfrac then
begin
  factor := 1;
  while lookahead in ['0'..'9'] do
  begin
    ch := nextchar;
    factor := factor/10;
    accum := accum+factor*(ord(ch)-48);
    inputresult := true;
  end; (* while *)

  (* Prepare for exponent following fraction *)
  if lookahead='E' then
  begin
    mode := lookexp;
    ch := nextchar;
  end;
end;
```

```
  if mode=lookexp then
  begin
    while lookahead in ['+','-'] do
    begin
      ch := nextchar;
      if ch='-' then
        expnegative := not expnegative;
    end; (* while *)

    while lookahead in ['0'..'9'] do
    begin
      ch := nextchar;
      exponent := exponent*10+ord(ch)-48;
    end; (* while *)

    if expnegative then
      exponent := -exponent;
  end; (* if *)

  (* Calculate result if possible *)
  if inputresult then
  begin
    (* Rangecheck exponent for overflow before multiplying with mantissa.
       Not necessary if accum=0. *)
    if accum<>0 then
    begin
      magnitude := round(log(accum));
      (* Ranges depending on mantissa *)
      if magnitude+exponent>127 then
        exponent := 127-magnitude;
      if magnitude+exponent<-127 then
        exponent := -127-magnitude;
      (* Independent ranges *)
      if exponent>127 then
        exponent := 127;
      if exponent<-127 then
        exponent := -127;
      if mantnegative then
        accum := -accum;
      str_real := accum*pwroften(exponent);
    end
    else
      str_real := 0;
  end; (* if *)
end; (* str_real *)


function valid_real;
  (* Reads reals with validation *)

var
```

```
  temp:string;

begin
  valid_string(temp,size,valider);
  valid_real := str_real(temp);
end; (* valid_real *)


function readreal;
  (* Reads reals *)

begin
  readreal := validreal(0,['-','+','.','E','e','0'..'9']);
end; (* readreal *)


function goodinput;
  (* Returns true if last input was valid *)

begin
  goodinput := inputresult;
end; (* goodinput *)


function abortinput;
  (* Returns true if last input was aborted *)

begin
  abortinput := abortresult;
end; (* abortinput *)


begin (* extrascreen *)
  inputresult := true;
  abortresult := false;
  digit := ['0'..'9'];
  numeric := digit+['-','+','.','E','e'];
  ualpha := ['A'..'Z'];
  terminator := [chr(13),chr(138),chr(139)];    (* CR, UP and DOWN *)
  abortkeys := [];
end.
```

Disk 96. Contents of file EXTRAASM.TEXT

```
;-------------------------------------
;
;      Support for unit EXTRASCREEN
;      ============================
; Bscan and Kscan comes from SCANNERS. Their function is identical.
;
; A-DATA 860114
;

EMPTY    .EQU 2D9EH     ;Empty bit mask
FULL     .EQU 2DA0H     ;Full bit mask
HEXFF    .EQU 40B8H     ;0FFH in the PME ROM

BUFPOINT .EQU 288AH     ;Pointer to keyboard buffer
BUFFLAG  .EQU 28A2H     ;Buffer mode flag
BUFGIVE  .EQU 28A4H     ;Buffer store pointer
BUFTAKE  .EQU 28A6H     ;Buffer fetch pointer
BUFEND   .EQU 31        ;Last index in key buffer

VDPWA    .EQU 8C02H
VDPRD    .EQU 8800H

SCR80    .EQU 2000H     ;80 col screen address
SCR80END .EQU 2780H     ;80 col screen end

SP       .EQU 10
LINK     .EQU 11

VDPR1    .EQU 2811H     ;Pascals copy of VDP R1
VDPCOPY  .EQU 83D4H     ;SCAN's expected VDP R1 copy location
SAVE1    .EQU 83C6H     ;Address to area that has to be saved

SCAN     .EQU 000EH     ;Address of monitor scan routine

         .ASECT
         .ORG 8374H
KEYBOARD .BYTE
KEYVALUE .BYTE
         .BLOCK 6
GPLST    .BYTE
         .PSECT

PASCALWS .EQU 8380H
GPLWS    .EQU 83E0H

;-------------
;
; function kscan(keyboard:integer):integer;
```

```
;
; Returns the keycode of the currently pressed key.
; If no key was pressed, returns -1.
; Also clears the keyboard buffer
;


         .RELFUNC KSCAN,1

         MOV  *SP+,R0            ;Load selected keyboard value
         MOVB @KEYBOARD,R1       ;Save keyboard configuration
         MOVB @PASCALWS+1,@KEYBOARD

         LWPI SCANWS             ;Save some of the RAM PAD
         LI   R0,SAVE1
         MOV  *R0+,R1
         MOV  *R0+,R2
         MOV  *R0,R3
         LI   R0,VDPCOPY
         MOV  *R0+,R4
         MOV  *R0+,R5
         MOV  *R0,R6

         MOVB @VDPR1,@VDPCOPY    ;Pascals copy of VDP R1
         LWPI GPLWS
         BL   @SCAN              ;Do the scanning
         LWPI SCANWS

         LI   R0,SAVE1           ;Restore saved values
         MOV  R1,*R0+
         MOV  R2,*R0+
         MOV  R3,*R0
         LI   R0,VDPCOPY
         MOV  R4,*R0+
         MOV  R5,*R0+
         MOV  R6,*R0

         LWPI PASCALWS
         CLR  R0
         MOVB @KEYVALUE,R0       ;Fetch detected key value
         CB   R0,@HEXFF          ;Check if no key was pressed
         JNE  KEY_DOWN
         SETO R0                 ;If not, return -1
KEY_DOWN SWPB R0                 ;Full word
         MOV  R0,*SP
         SZC  @FULL,@BUFFLAG     ;Set buffer flag to empty
         SOC  @EMPTY,@BUFFLAG
         MOV  @BUFTAKE,@BUFGIVE
         MOVB R1,@KEYBOARD       ;Restore configuration
         B    *LINK
```

```
SCANWS   .BLOCK 32


;--------------
;
; function bscan:integer;
;
; Returns the code of the next character in the keyboard buffer.
; Returns -1 if the buffer is empty.
; Also removes eventually read character from the buffer.
;
;

         .RELFUNC BSCAN

         MOV  @BUFFLAG,R0       ;Anything in the buffer?
         COC  @EMPTY,R0
         JEQ  IS_EMPTY

         MOV  @BUFTAKE,R1       ;Calculate address in buffer
         MOV  R1,R2
         A    @BUFPOINT,R2
         SWPB R2                ;Write to VDP
         MOVB R2,@VDPWA
         SWPB R2
         MOVB R2,@VDPWA

         INC  R1                ;pointer := (pointer+1) mod 32;
         ANDI R1,BUFEND
         MOV  R1,@BUFTAKE

         C    R1,@BUFGIVE       ;Is buffer empty now?
         JNE  SOME_LEFT
         SOC  @EMPTY,R0         ;Set flag to empty
SOMELEFT SZC  @FULL,R0          ;Always set flag to not full
         MOV  R0,@BUFFLAG
         CLR  *SP
         MOVB @VDPRD,@1(SP)     ;Fetch data from buffer
         B    *LINK

IS_EMPTY SETO *SP
         B    *LINK


;--------------
;
; Getlast returns the last character read from the buffer.
; Can be used for testing if ENTER or something else terminated input.
;
```

```
         .RELFUNC GETLAST

         MOV  @BUFTAKE,R1       ;Points to next character to fetch
         DEC  R1                ;Hence, back one
         ANDI R1,BUFEND
         A    @BUFPOINT,R1

         SWPB R1
         MOVB R1,@VDPWA
         SWPB R1
         MOVB R1,@VDPWA

         JMP  $+2       ;Takes time
         MOVB @VDPRD,R1
         SRL  R1,8
         MOV  R1,*SP
         B    *LINK

;--------------
;
; procedure horchar(x,y,length:integer; ch:char);
; Same as HCHAR in BASIC.
;
         .RELPROC HORCHAR,4

         MOV  *SP+,R0   ;Character
         SWPB R0
         MOV  *SP+,R1   ;Length
         JNE  CONT
         AI   SP,4
         B    *LINK     ;Leave if no chars

CONT     MOV  *SP+,R2
         LI   R4,80
         MPY  R4,R2
         A    *SP+,R3
         AI   R3,SCR80  ;R3 := y*80+x+screenbase

CHARLOOP MOVB R0,*R3+
         CI   R3,SCR80END       ;Out of screen?
         JL   ONSCREEN
         LI   R3,SCR80
ONSCREEN DEC  R1        ;Byte count
         JNE  CHARLOOP

         MOV  LINK,R0
         BL   @47D2H    ;Show current window
         B    *R0
```

```
;---------------
;
; procedure verchar(x,y,length:integer; ch:char);
; Same as VCHAR in BASIC.
;

         .RELPROC VERCHAR,4

         MOV  *SP+,R0   ;Character
         SWPB R0
         MOV  *SP+,R1   ;Length
         JNE  CONT
         AI   SP,4
         B    *LINK

CONT     MOV  *SP+,R2   ;Y
         LI   R4,80
         MPY  R4,R2
         A    *SP+,R3
         AI   R3,SCR80  ;R3 := y*80+x+screenbase

CHARLOOP MOVB R0,*R3
         AI   R3,80     ;Calculate next screen position
         CI   R3,SCR80END       ;Check if on screen
         JL   ONSCREEN
         AI   R3,-1919
         CI   R3,SCR80+80
         JL   ONSCREEN
         LI   R3,SCR80
ONSCREEN DEC  R1
         JNE  CHARLOOP

         MOV  LINK,R0
         BL   @47D2H    ;Show current window
         B    *R0

         .END
```

Disk 96. Contents of file ERRORASM.TEXT

```
;--------------------------
;
; Support for error message unit.
; Putmessage places message on screen.
; Restore removes all messages.
;
; A-DATA 851205
;

;-----------------
; Errws
;
; R0  Temp
; R1  Message pointer
; R2  Last index in message
; R3  Current line position
; R4  Text positions on line
; R5  Current frame code
; R6  Counter
; R7  VDP position
; R8  Temp for multiply
; R9  Temp
; R10 Stack pointer
; R11 Return linkage
; R12 Next word length
; R13 Local string pointer for space scanning
; R14 height-1
; R15
;
;-----------------

VDPWD    .EQU 8C00H
VDPWA    .EQU 8C02H

WIDTH    .EQU 2CC6H     ;P-system screen width
HEIGHT   .EQU 2CC8H     ;P-system screen height

SP       .EQU 10
LINK     .EQU 11

LINE     .EQU 2         ;SP offsets for arguments
COL      .EQU 4
ROW      .EQU 6
FRAME    .EQU 8

PASCALWS .EQU 8380H
SCRBASE  .EQU 0800H     ;Screen image table
```

```
          .RELPROC PUTMESSAGE,5

          LWPI ERRWS
          MOV  @PASCALWS+20,SP   ;Copy SP

          MOV  @ROW(SP),R0       ;Check row, col and line for negative values
          JGT  ROWPOS
          CLR  @ROW(SP)
ROWPOS    MOV  @COL(SP),R0
          JGT  COLPOS
          CLR  @COL(SP)
COLPOS    LI   R0,3
          C    @LINE(SP),R0
          JGT  LINEBIG
          MOV  R0,@LINE(SP)

LINEBIG   MOV  @HEIGHT,R0        ;if row>height-3 then row := height-3
          MOV  R0,R14
          DEC  R14               ;height-1
          AI   R0,-3
          C    @ROW(SP),R0
          JLE  COLCHECK
          MOV  R0,@ROW(SP)

COLCHECK  MOV  @WIDTH,R0         ;if col>width-3 then col := width-3
          AI   R0,-3
          C    @COL(SP),R0
          JLE  LINECHECK
          MOV  R0,@COL(SP)

LINECHECK
          MOV  @COL(SP),R0       ;if col+line>width then line := width-col
          MOV  @LINE(SP),R4
          A    R4,R0
          C    R0,@WIDTH
          JLE  LINEOK
          MOV  @WIDTH,R4
          S    @COL(SP),R4

LINEOK    DECT R4                ;tline := line-2

          MOV  *SP,R1    ;Message pointer
          MOVB *R1,R2    ;Message length
          SRL  R2,8

          MOV  @HEIGHT,R8
          DECT R8
          S    @ROW(SP),R8
          MPY  R4,R8     ;R9 := (height-2-row)*tline
          C    R9,R2     ;Keep the smallest
          JHE  NOACT
```

```
         MOV  R9,R2
NOACT    A    R1,R2     ;Convert to pointer to last char in string

         MOV  @ROW(SP),R8
         MPY  @WIDTH,R8
         MOV  R9,R7
         A    @COL(SP),R7       ;VDP pos := row*width+col
         AI   R7,SCRBASE
         BL   @MOVVDPA

         MOV  @FRAME(SP),R5
         MOVB @ERRWS+11,@VDPWD  ;Upper left corner
         BL   @DRAWLINE         ;Horizontal line
         MOV  @FRAME(SP),R5
         AI   R5,5
         MOVB @ERRWS+11,@VDPWD  ;Upper right corner

         MOV  @FRAME(SP),R5
         INC  R5        ;Vertical line

         A    @WIDTH,R7         ;New line
         BL   @MOVVDPA
         INC  @ROW(SP)  ;Count rows for screen end detection
         CLR  R3        ;Current := 0
         MOVB @ERRWS+11,@VDPWD  ;Beginning vertical line

WHILE    SETO R0
         CLR  R12       ;Length := 0
         MOV  R1,R13    ;Localpoint := messagepoint
WHILE1   C    R13,R2    ;if localpoint>=lastindex then at end of string
         JHE  WHILE2
         INC  R13       ;Check next char
         CB   *R13,@SPACE
         JEQ  ISSPACE
         INC  R12       ;Increase length
         CLR  R0
         JMP  WHILE1
ISSPACE  MOV  R0,R0     ;If the space comes first, then it counts
         JEQ  WHILE2
         INC  R12
         JMP  WHILE1

WHILE2
         MOV  R12,R12   ;if length=0 then no more text
         JEQ  LASTLINE
         C    R12,R4    ;if length<=tline, then word fits on one line
         JLE  WRAP
         MOV  R3,R3     ;Long word. See if at beginning of line already
         JEQ  NEXTLONG  ;If not, wrap
         MOV  R4,R0     ;Calculate number of positions to fill on current line
```

```
         S    R3,R0
         BL   @WRAPPER

NEXTLONG MOV  R4,R6     ;Write full line with part of long word
LONGLOOP INC  R1
         MOVB *R1,@VDPWD        ;Next character
         DEC  R6
         JNE  LONGLOOP
         BL   @NEWROW   ;Move to next row
         S    R4,R12    ;Decrease remaining word length
         JEQ  LONGEND   ;If complete end of this word
         C    R12,R4
         JLE  NORMAL
         JMP  NEXTLONG
LONGEND  INC  R1
         JMP  WHILE

WRAP     MOV  R4,R0     ;if length>(tline-current) then wrap
         S    R3,R0
         C    R12,R0
         JLE  NORMAL    ;Short word, wrapping not needed
         BL   @WRAPPER  ;Fill end of line with spaces, start next line

NORMAL   MOV  R12,R6    ;Write length character from string
NORMPRINT
         INC  R1
         MOVB *R1,@VDPWD
         DEC  R6
         JNE  NORMPRINT

         A    R12,R3    ;Current := current+length
         INC  R1        ;Pass space in string
         C    R3,R4     ;Space not written at end of line
         JHE  WHILE     ;Next word
         MOVB @SPACE,@VDPWD
         INC  R3        ;Current := current+1
         JMP  WHILE

LASTLINE MOV  R4,R6     ;Calculate number of locations to fill on last line
         S    R3,R6
         JEQ  ENDBLANK  ;None?
BLANKS   MOVB @SPACE,@VDPWD
         DEC  R6
         JNE  BLANKS

ENDBLANK MOVB @ERRWS+11,@VDPWD  ;Ending vertical line

BOTTOM   A    @WIDTH,R7
         BL   @MOVVDPA
         MOV  @FRAME(SP),R5     ;Lower left corner
         INCT R5
```

```
         MOVB @ERRWS+11,@VDPWD
         BL   @DRAWLINE          ;Horizontal line
         MOV  @FRAME(SP),R5
         AI   R5,4
         MOVB @ERRWS+11,@VDPWD   ;Lower right corner

         LWPI PASCALWS           ;Back to pascal host
         AI   SP,10
         B    *LINK


MOVVDPA  MOVB @ERRWS+15,@VDPWA
         ORI  R7,4000H
         MOVB R7,@VDPWA
         B    *LINK


DRAWLINE
         MOV  R4,R6
         JEQ  NODRAW
         MOV  @FRAME(SP),R5
         AI   R5,3
         SWPB R5
DRAW     MOVB R5,@VDPWD
         DEC  R6
         JNE  DRAW
NODRAW   B    *LINK


NEWROW   MOV  LINK,R0
         MOVB @ERRWS+11,@VDPWD   ;vertical line
         INC  @ROW(SP)   ;Check if outside screen soon
         C    @ROW(SP),R14
         JHE  BOTTOM     ;if so, draw the bottom line
         A    @WIDTH,R7
         BL   @MOVVDPA
         MOVB @ERRWS+11,@VDPWD   ;Beginning vertical line
         CLR  R3         ;Current := 0
         B    *R0


WRAPPER  MOV  LINK,R6    ;Fills end of line when word doesn't fit
         MOV  R0,R0      ;Number of bytes to fill on current line
         JEQ  OUTFILL
         JLT  OUTFILL
FILLING  MOVB @SPACE,@VDPWD
         DEC  R0
         JGT  FILLING

OUTFILL  BL   @NEWROW    ;Current line filled, move to next
```

```
         B    *R6


ERRWS    .BLOCK 32
SPACE    .BYTE " "
         .ALIGN 2


         .RELPROC RESTORE

         MOV  LINK,R0
         BL   @47D2H    ;Call PME screen window routine
         B    *R0

         .END
```

Disk 96. Contents of file ERRORTEST.TEXT

```
program errortest;
  (* Error message example   A-DATA 860212 *)

uses
  showerrors,
  support;

var
  i    :integer;
  frame:integer;


procedure fill1;

var i:integer;

begin
  for i := 1 to 24 do
    writeln('Message demonstration in text mode');
end;


procedure fill2;

var
  i:integer;

begin
  for i := 1 to 24 do
    writeln('Message demonstration in graphics mode');
end;


procedure messages;

begin
  errormessage(0,0,20,'This is the first message in this test.');
  errormessage(15,20,20,concat
    ('These messages will be shown for about 20 seconds. When that time has ',
    'passed, they will be erased'));
  errormessage(3,7,15,'This is the last message here.');
end; (* messages *)



begin (* main *)
  frame := 140;
  initerror(frame);
```

```
  (* Text mode *)
  fill1;
  messages;
  for i := 1 to 20000 do;
  errorrestore;
  for i := 1 to 2000 do;


  (* graphics mode *)
  set_screen(2);
  fill2;
  messages;
  for i := 1 to 20000 do;
  errorrestore;
  for i := 1 to 2000 do;

  set_screen(1);
end.
```

Disk 96. Contents of file EXTRATEST.TEXT

```
program extratest;
  (* Demonstration of capabilities provided by unit extrascreen.
     A-DATA 870304 *)

uses
  extrascreen;

var
  xpos,
  ypos,
  i :integer;
  x :real;

begin
  (* The structure used in this program, without procedures and functions,
     is not recommended for serious programming. This is only a demonstration
     program, not intended for further expansion or modification. *)

  write(chr(12));
  writeln('Press any key to stop');

  gotoxy(0,10);
  i := 0;
  repeat
    i := i+1;
    write('Demonstration line ',i);
    newline(0,79,10,23);
  until bufscan<>-1;
  (* Keyscan could have been used, if buffering wasn't desired. *)

  writeln;
  writeln;
  write('Enter an integer ');
  i := readint;
  writeln;
  writeln('You entered ',i);
  writeln;

  write('Enter a real ');
  x := readreal;
  writeln;
  writeln('You entered ',x);
  writeln;

  writeln('Enter a real, but with only some');
  writeln('digits allowed.');
  x := validread(6,digit-['8','9']);
  writeln;
```

```
  writeln('You entered ',x);
  (* Validint and validstring are used in similar manners *)

  writeln('Last key accepted is chr(',ord(lastchar),')');
  if goodinput then
    writeln('Last input was OK')
  else
    writeln('Last input was wrong');

  writeln;
  writeln;
  findxy(xpos,ypos);
  writeln('Current write position is ',xpos,',',ypos);

  writeln;
  writeln;
  hchar(0,0,400,'#');
  vchar(20,0,10,'*');
  writeln('Character at 3,20 is ',getchar(3,20));
end. (* extratest *)
```

Disk 96. Contents of file ERRORINFO.TEXT

ERROR MESSAGES

The name of this unit is perhaps not the best. The unit might be used for any kind of messages. But since it's so easy to remove the message windows it creates, it's especially handy for error messages. If, for example, some error occurs when the screen is set up in some fancy way, the message can be shown and then removed without disturbing the original display.

ERRORTEST.CODE demonstrates the use of these functions. Note that ERRORTEST.CODE expects to find the unit SHOWERRORS in *SYSTEM.LIBRARY. If the unit isn't present, ERRORTEST will refuse to work. The SYSTEM.LIBRARY file you received with this package contains SHOWERRORS.

ERRORTEST.TEXT is the source of the test program. ERRORPAS.TEXT also contains short instructions.

Initerror must be called before use of errormessage. The number given to initerror designates the character code of the first of the six characters used to define the patterns that builds the frame around a window. All the six characters must belong to the same color group. If you fail to obey this rule, initerror will change the value for you. This is of no importance as long as you use the text mode, but SHOWERRORS works in graphics mode as well. This detail makes sure that the entire frame has the same color.

Errormessage accepts messages up to 255 characters long. The length of each row in the window should be given when the procedure is called. Note that two positions are used by the frame. Hence, if the length is given as ten, eight writeable positions remains on each row.

Errormessage has an automatic word-wrap feature. The line is split at spaces only. Only words too long to fit on a single row are split.

Errorrestore removes all windows currently on the screen. Errorrestore is really the routine which is executed when you press "Screen Left" or "Screen Right". The only difference is that there is no sideways movement.

Two consequences of this is first that message windows might be erased by pressing **FCTN 7** or **FCTN 8** and second that windows can only be opened on the visible part of the screen. On the other hand, the latter fact means that messages are always visible, no matter which part of the screen that is currently displayed.

All CPU-intensive parts in unit showerrors (i.e. the creation of a window) are implemented in assembly language. The result is that, in spite of all the manouvers required to achieve the word-wrap function, the window is created on the screen faster than the eye can see.

If you define a window that doesn't fit on the screen, the size of the window is automatically changed. If the text doesn't fit in the window, as many words as possible are shown.

Windows can be created in both text and graphics mode. With only 32 characters per line, windows obviously can't be as big as when 40 columns are available. Whichever mode you use, showerrors will adapt the window size to fit the length of the message string. Inspect ERRORTEST.TEXT for more details.

Disk 96. Contents of file EXTRAINFO.TEXT

SCREEN HANDLING

Input of numbers in Pascal is by tradition troublesome. One erroneous keypress is enough to create a run-time error.

Unit extrascreen provides a solution to these problems. Among other things, it contains routines which allows input of numbers, no matter which keys are accidentally pressed.

The declarations of the functions are found in EXTRAPAS.TEXT. Below is a description of their use.

Keyscan corresponds to CALL KEY in BASIC. The function requires the keyboard mode as an argument. Pascal usually uses number four (4), but it's possible to use any number in the range 1..5. When keyscan is called, the keyboard is scanned in that very moment. The code of the pressed key is returned, or -1 if no key was pressed.

Eventual prior key presses, stored in the input buffer, are ignored. The buffer is also erased every time keyscan is called. This is done to avoid problems with buffer overflow, if only keyscan is used for input for a long time. Their might be some character left in the buffer anyway, if some key was pressed after the last invocation of keyscan. If you are accepting some kind of input later, using the read statement, it might be wise to let a keyscan precede the read statement.

This gives some problems when keyscan is used, but it's still necessary in real-time applications.

Bufscan also reads the keyboard without halting the program. The difference is that bufscan inspects the buffer, and accepts the first character in the buffer. If there is none, -1 is returned.

Readint reads an integer. The cursor is placed at the current screen location, which can be controlled by gotoxy. Only characters normally allowed in integers are accepted. Eventual erroneous digits are erased as usual with the left arrow.

If readint can't interpret the entered characters as a valid number, zero (0) is returned. This occurs for example if only a minus sign is entered. This is legal, since that character is allowed in an integer, but it still doesn't give a legal number.

Readreal is the same as readint, but for real numbers. All kinds of reals are allowed. Exponent might be given with upper- or lower-case E. Just like readint, readreal returns zero (0) if the entered characters doesn't form a legal number.

The interested user might want to know that all input is placed in a string. When **ENTER** is pressed, the string is converted to a number. The interpretation of floating point numbers is by far the most complicated routine in extrascreen. . .

Validint works like readint, but gives you more opportunities. You might specify the maximal number of characters allowed, as well as which characters. This corresponds to the ACCEPT SIZE VALIDATE in X-BASIC. Validint(5,['0'..'7']) reads integers with a maximal length of five characters and allows only the digits 0 to 7 to be entered. If you don't want to limit the number of characters, specify 0 (zero).

In order to simplify the use of these routines, some character sets are predefined. Validint(0,digit) reads an integer with unlimited length. It might contain any digit, but not '+' or '-'.

Validreal corresponds to validint, but returns a real value. The set called numeric contains all the characters normally allowed in floating point notation. If you want to read a real number which not is allowed to have any exponent part, call validread(0,numeric-['E','e']).

Inside extrascreen readint calls validint with suitable arguments. The same goes for readreal and validreal.

Validstring allows controlled entry of strings. The read string is returned in result. Here too, the size might be given as zero if you don't want any limit. Validstring(filename,15,ualpha) reads a filename, which should be declared as a string, with no more than 15 characters. Only capital letters are accepted.

Actually validstring is the only procedure that interacts with the user in extrascreen. Validint and validreal both calls validstring.

Lastchar gives the last character already accepted from the input buffer. This makes it possible to determine how an entry was terminated. The use of this feature is explained below.

Goodinput can be called to examine the result of an input. When reading integers and reals, zero is returned if the number is illegal. Hence, it's impossible to determine whether it was zero or something illegal that was entered. Goodinput returns false if something was wrong in the last entry, otherwise true.

Abortinput returns true if the last input was interrupted. Normally false is returned. The use of this feature is explained below.

Getchar works like GCHAR in BASIC. The character in a certain place on the screen is returned. Note that the entire 80 column screen is read. Thus you might read characters that aren't currently visible. However, getchar can't read "ghost characters" like the ones placed on the screen by errormessage. Even if the original characters seems to disappear under the message window, they are returned by getchar.

Hchar corresponds to HCHAR in BASIC. The difference is that the number of characters must always be given, even if it's only one. Further on, a character, not a character code, is given as an argument. Hchar draws on the entire 80 column screen. Message windows created by errormessage are erased, even if there is no coincidence between the window and the characters placed on the screen by hchar.

Vchar is the same as hchar, except that characters are repeated vertically, not horizontally.

Newline allows scrolling of parts of the screen. A typical example is a program that prints a list with some kind of heading on the screen. If the entire list contains more lines than the screen, the heading is lost when the scrolling starts. Then newline might be used to scroll the list only. Newline works withing the area specified in the arguments. The program EXTRATEST.CODE uses this ability. Look at EXTRATEST.TEXT for further details.

Findxy if the opposite of gotoxy. Normally, it's impossible to retrieve the current write position. But findxy returns the position, in the same format as it should have been given to gotoxy.

INTERRUPTING INPUT

Extrascreen has some special capabilities for handling unusual input requirements. Two sets, terminator and abortkeys, contains characters which terminates or aborts the input. When extrascreen starts, terminator is loaded with ENTER and the up and down arrows. All of these might be used to indicate the end of a number or a string. The function lastchar can be called to determine which key was actually used. Hence, you have the ability to perform different actions depending upon the key used to terminate an entry.

Abortkeys contains characters used to interrupt the input. If you want some special key to allow you to get out of some data entry sequence, then you might use this feature to accomplish that. The function abortinput returns true if the entry was aborted. When extrascreen starts, the character set called abortkeys is empty. Hence, no key is able to interrupt the input. But both abortkeys and terminator are accessible outside extrascreen. You can decide which keys you want to use for the different purposes.

Of course, only the terminator set, together with lastchar, is really necessary to accomplish this. But by separating the termination of a normal entry form the abortion, the proper response from your program is easier to detect.

All together, this makes advanced (seen from the programmers point of view) routines, which seems simple (from the users point of view) easy to write. Everything is designed in a way that allows easy access if you don't need the fancy parts. Extrascreen automatically initiates its variables to values designed for normal needs. The extra capacity is there when you need it.

As is the case with showerrors, complicated tasks are either written in assembly language or efficiently implemented in Pascal, using the special intrinsics provided by UCSD Pascal. The exception is the interpretation of numbers, but this doesn't matter, since the computer is fast anyway, compared to the user.

Disk 96. Contents of file FORMATTER.TEXT

```
program dformat;
  (* Disk formatter utility by A-DATA 851228 *)

uses
  extrascreen;      (* For readint *)

var
  sdtable:array [1..2] of string;



procedure asmformat(var errcode:integer;
                    drive,tracks,sides,density:integer;
                    var buffer);                external;
  (* Low level formatter procedure *)


procedure handle_format;

var
  unitno,
  tracks,
  sides,
  density,
  drive,
  errcode:integer;
  ok     :boolean;
  ch     :char;


  procedure format(var errcode:integer;
                   drive,tracks,sides,density:integer);
    (* Higher level formatter *)

  type
    buftype = packed array[0..4095] of integer;

  var
    buffer:buftype;

  begin  (* format *)
    asmformat(errcode,drive,tracks,sides,density,buffer);
  end; (* format *)


  function sdconv(ch:char; var ok:boolean):integer;
    (* Converts 's' and 'd' to 1 and 2 *)
```

```
begin
  ok := false;
  case ch of
    'S','s':begin
              sdconv := 1;
              ok := true;
            end;
    'D','d':begin
              sdconv := 2;
              ok := true;
            end;
  end; (* case *)
  if not ok then
    writeln(chr(7),'  ERROR - Response must be S or D');
end; (* sdconv *)


function unit_to_drive(unitno:integer):integer;

begin
  case unitno of
    4 : unit_to_drive := 1;
    5 : unit_to_drive := 2;
    9 : unit_to_drive := 3;
   10 : unit_to_drive := 4;
  end; (* case *)
end; (* unit_to_drive *)


procedure error_report(code:integer);

begin
  case code of
     6 : begin
           writeln(chr(7));
           writeln('  ERROR - Can''t find unit/diskette');
         end;
     7 : begin
           writeln(chr(7));
           writeln('  ERROR - Unable to format');
         end;
    52 : begin
           writeln(chr(7));
           writeln('  ERROR - Diskette write protected');
         end;
  end; (* case *)
end; (* error_report *)


procedure writeinfo(unitno,tracks,newsides,newdensity:integer);
```

```
  (* Creates a disk head on the formatted disk *)

type
  nametype  = packed array[0..9] of char;
  byte      = 0..255;
  dinfotype = packed record
    diskname   :nametype;
    disksize   :integer;
    valider    :packed array[0..3] of char;
    trackonside:byte;
    protection :char;
    sides,
    density    :byte;
    fill1      :packed array[20..55] of char;
    allocation :packed array[0..199] of char;
    headpoint  :packed array[0..127] of integer;
  end; (* dinfotype *)

  finfotype = packed record
    filename  :nametype;
    fill1     :integer;
    status,
    maxrecord :byte;
    used      :integer;
    eofoffset,
    recl      :byte;
    sectused  :integer;
    fill2     :packed array[20..27] of char;
    cluster   :packed array[0..227] of char;
    fill3     :packed array[0..255] of char;
  end; (* finfotype *)

var
  diskinfo:dinfotype;
  fileinfo:finfotype;

  procedure swapbyte(var x:integer);
    (* Takes a word and swaps the bytes *)

  type
    byteword = record
      case boolean of
        true :(addr:integer);
        false:(bytes:packed array[1..2] of byte);
    end;

  var
    word :byteword;
    tbyte:byte;

  begin (* swapbyte *)
```

```
    with word do
    begin
      addr := x;
      tbyte := bytes[1];
      bytes[1] := bytes[2];
      bytes[2] := tbyte;
      x := addr;
    end; (* with *)
  end; (* swapbyte *)


begin (* writeinfo *)
  with diskinfo do
  begin
    diskname := 'PASCAL    ';
    disksize := tracks*newsides*9*newdensity;
    valider  := ' DSK';
    valider[0] := chr(9*newdensity);
    trackonside := tracks;
    protection  := ' ';
    density     := newdensity;
    sides       := newsides;
    fillchar(fill1,sizeof(fill1),chr(0));
    fillchar(allocation,sizeof(allocation),chr(255));
    fillchar(headpoint,sizeof(headpoint),chr(0));
    headpoint[0] := 2;
  end; (* with *)
  unitwrite(unitno,diskinfo,512,0);

  with fileinfo do
  begin
    filename := 'PASCAL    ';
    fill1    := 0;
    maxrecord := 2;
    status    := 0;   (* Display/Fixed *)
    used      := diskinfo.disksize-3;
    recl      := 128;
    eofoffset := 0;
    sectused  := maxrecord*used;
    swapbyte(sectused);
    fillchar(fill2,sizeof(fill2),chr(0));
    fillchar(cluster,sizeof(cluster),chr(0));
    cluster[0] := chr(3);
    cluster[1] := chr(((used-1) mod 16)*16);
    cluster[2] := chr((used-1) div 16);
    fillchar(fill3,sizeof(fill3),chr(0));
  end; (* with *)
  unitwrite(unitno,fileinfo,512,1);
end; (* writeinfo *)
```

```
begin (* handle_format *)
  repeat
    page(output);
    writeln('D I S K  F O R M A T  U T I L I T Y');
    gotoxy(30,1);
    writeln('A-DATA 85');

    repeat
      writeln;
      writeln('Enter the number of the unit');
      write('holding the diskette (4,5,9,10). ');
      unitno := validint(2,digit);
      writeln;
      ok := unitno in [4,5,9,10];
      if not ok then
        writeln(chr(7),'  ERROR - Unit must be 4,5,9 or 10');
    until ok;

    repeat
      writeln;
      writeln('How many tracks do you want');
      write('on a side (normally 40)? ');
      tracks := validint(2,digit);
      writeln;
      ok := tracks in [1..99];
      if not ok then
        writeln(chr(7),'  ERROR - Tracks must be from 1 to 99');
    until ok;

    repeat
      writeln;
      writeln('Format diskette as single sided (S)');
      write('or double sided (D)? ');
      read(ch);
      writeln;
      sides := sdconv(ch,ok);
    until ok;

    repeat
      writeln;
      writeln('Format diskette as single density (S)');
      write('or double density (D)? ');
      read(ch);
      writeln;
      density := sdconv(ch,ok);
    until ok;

    writeln;
    writeln('You have selected:');
    writeln('  Unit number ',unitno);
    writeln('  Tracks per side ',tracks);
```

```
    writeln('  ',sdtable[sides],' sided');
    writeln('  ',sdtable[density],' density');

    repeat
      writeln;
      write('Ready to format the disk (Y/N)? ');
      read(ch);
      writeln;
    until ch in ['Y','y','N','n'];

    if ch in ['Y','y'] then
    begin
      drive := unit_to_drive(unitno);
      format(errcode,drive,tracks,sides,density);
      error_report(errcode);
      if errcode=0 then
        writeinfo(unitno,tracks,sides,density);
    end;

    repeat
      writeln;
      write('Format another disk (Y/N)? ');
      read(ch);
      writeln;
    until ch in ['N','n','Y','y'];
  until ch in ['N','n'];
end; (* handle_format *)


begin (* main *)
  sdtable[1] := 'Single';
  sdtable[2] := 'Double';
  handle_format;
end.
```

Disk 96. Contents of file FORMASM.TEXT

```
         .PAGEHEIGHT 72
         .TITLE "Disk formatter support"
;------------------------
;
; Support for disk formatter
;
; procedure asmformat(var errcode:integer;
;                     drive,tracks,sides,density:integer;
;                     var buffer:buftype);
;
; buftype = packed array[0..4095] of integer; (* for example *)
;
; Uses the subprogram for formatting on the disk controller card. Since the
; default sector interlace used for double density with CorComp cards isn't
; optimal for Pascal, custom interlacing is used in case of the combination
; CorComp - DD is requested.
;
;  A-DATA 870309
;

;--------------------
; Register usage (PASCALWS)
;
; R0  Temp
; R1  Temp
; R2  Temp
; R3
; R4  TI vs. CorComp and DD flag
; R5  Pascal return address
; R6  Error code return pointer
; R7  Buffer pointer
; R8  PME use
; R9  PME
; R10 Stack pointer
; R11 Return link
; R12 CRU base
; R13 PME
; R14 PME
; R15 PME
;

SP       .EQU 10
LINK     .EQU 11

VDPWA    .EQU 8C02H
VDPWD    .EQU 8C00H
VDPRD    .EQU 8800H
```

```
FAC      .EQU 834AH
GPLWS    .EQU 83E0H
PASCALWS .EQU 8380H

DISKCRU  .EQU 1100H
PCODECRU .EQU 1F00H

VDPBUF   .EQU 0E00H
BUFSIZE  .EQU 2000H

CUSTFLAG .EQU 400BH     ;Custom interlacing flag
INTERLACE .EQU 8338H    ;Custom interlacing data buffer

         .PROC ASMFORMAT,6

         MOV  LINK,R5
         LI   R0,FAC    ;Save FAC
         LI   R1,FACSAVE
         LI   R2,8
         BL   @MOVIT

         LI   R0,GPLWS
         LI   R1,GPLSAVE
         LI   R2,32
         BL   @MOVIT

         MOV  *SP+,R7   ;Buffer pointer

         STWP R1
         INC  R1        ;Pointer to R0(low byte)
         MOV  *SP+,R0   ;Density
         MOVB *R1,@FAC+6
         MOV  *SP+,R0   ;Sides
         MOVB *R1,@FAC+7
         MOV  *SP+,R0   ;Tracks
         MOVB *R1,@FAC+3
         MOV  *SP+,R0   ;Drive
         MOVB *R1,@FAC+2

         MOV  *SP+,R6   ;Error code pointer

         LI   R0,VDPBUF
         MOV  R0,@FAC+4 ;VDP buffer pointer
         SWPB R0        ;Save VDP buffer
         MOVB R0,@VDPWA
         SWPB R0
         MOVB R0,@VDPWA
         MOV  R7,R1     ;Buffer pointer
         LI   R2,BUFSIZE
VDPLOP1  MOVB @VDPRD,*R1+
```

```
          DEC  R2
          JNE  VDPLOP1

          MOV  R12,@CRUSAVE
          LI   R12,PCODECRU
          SBZ  0          ;Disable p-code
          LI   R12,DISKCRU
          SBO  0          ;Enable disk controller

          CLR  R4         ;Assume TI card
          CB   @4000H,@VALIDDSR  ;Disk controller present?
          JNE  UNABLE

          LI   R2,8       ;See if it's a CorComp card
          MOV  @400AH,R1
CCLOOP    MOV  *R1,R1     ;Find 9th subprogram
          DEC  R2
          JNE  CCLOOP
          AI   R1,5       ;Point to text
          LI   R0,MGRTXT ;Compare with 'MGR'
          LI   R2,3
CHKLOOP   CB   *R0+,*R1+
          JNE  NOTCC      ;Can't be CorComp
          DEC  R2
          JNE  CHKLOOP

          SETO R4         ;Yes, it's a CorComp controller
          CB   @H02,@FAC+6        ;Double density?
          JEQ  DOUBLE
          INV  R4         ;Reset flag, since custom interlacing is used only if
;                          formatting in DD.
DOUBLE    MOV  R4,R4
          JEQ  NOTCC      ;Use custom interlacing?

          SBO  11         ;Set custom bit in disk controller RAM
          MOVB @CUSTFLAG,R0
          ANDI R0,0F000H
          ORI  R0,1000H
          MOVB R0,@CUSTFLAG
          SBZ  11

          LI   R0,INTERLACE       ;Save interlace buffer
          LI   R1,INTSAVE
          LI   R2,18
          BL   @MOVIT

          LI   R0,INTDATA         ;Load interlace buffer
          LI   R1,INTERLACE
          LI   R2,18
          BL   @MOVIT
```

```
NOTCC    MOV  @400AH,R2 ;Subprogram link
         JEQ  UNABLE
SEARCH   C    @4(R2),@FORMPGM   ;Formatter found?
         JEQ  DOFORM
         MOV  *R2,R2    ;Next subprogram
         JNE  SEARCH

UNABLE   LI   R0,7      ;Error code - unable to format
         MOV  R0,*R6    ;Return error code
         JMP  LEAVE

DOFORM
         MOV  @2(R2),@GPLWS+2   ;Address of formatter in GPLWS(R1)
         LWPI GPLWS
         LI   R12,DISKCRU
         LI   R15,VDPWA
         BL   *R1       ;Run formatter
         NOP
         LWPI PASCALWS
         CLR  *R6       ;Return error code
         MOVB @FAC+6,@1(R6)

LEAVE    MOV  R4,R4     ;Custom interlacing used?
         JEQ  NOTCC2

         LI   R0,INTSAVE        ;Reload interlace buffer
         LI   R1,INTERLACE
         LI   R2,18
         BL   @MOVIT

         SBO  11        ;Reset custom control bit
         MOVB @CUSTFLAG,R0
         ANDI R0,0E000H
         MOVB R0,@CUSTFLAG
         SBZ  11

NOTCC2   SBZ  0         ;Disable disk controller
         LI   R12,PCODECRU
         SBO  0         ;Enable p-code

         LI   R0,VDPBUF+4000H   ;Restore VDP RAM buffer
         SWPB R0
         MOVB R0,@VDPWA
         SWPB R0
         MOVB R0,@VDPWA
         MOV  R7,R1
         LI   R2,BUFSIZE
VDPLOP2  MOVB *R1+,@VDPWD
         DEC  R2
         JNE  VDPLOP2
```

```
         LI   R0,FACSAVE        ;Restore FAC
         LI   R1,FAC
         LI   R2,8
         BL   @MOVIT

         LI   R0,GPLSAVE        ;Restore GPLWS
         LI   R1,GPLWS
         LI   R2,32
         BL   @MOVIT

         MOV  @CRUSAVE,R12
         B    *R5       ;Saved return address

MOVIT    MOV  *R0+,*R1+
         DECT R2
         JNE  MOVIT
         B    *LINK

FACSAVE  .BLOCK 8       ;FAC save area
GPLSAVE  .BLOCK 32      ;GPLWS save area
INTSAVE  .BLOCK 18      ;Save area for custom interlacing buffer
CRUSAVE  .WORD  0
FORMPGM  .WORD  0111H   ;Formatter subprogram
INTDATA  .BYTE  0,2,4,6,8,10,12,14,16   ;Custom interlacing scheme
         .BYTE  1,3,5,7,9,11,13,15,17
MGRTXT   .ASCII "MGR"
H02      .BYTE  2
VALIDDSR .BYTE  0AAH
         .ALIGN 2
         .END
```

Disk 96. Contents of file FILEPRINT.TEXT

```
program fileprinter;
  (* Prints text files on the printer. Any number of files can be specified.
     They are printed in succession, with a form feed between each.
     The program uses the fastio package for reading text files.
     A combination of conventional writeln and unitwrite is used for printer
     output.

     A-DATA 871113 *)

uses
  realtime,
  misc,
  epsoncodes,
  fastio;

type
  national = (english,swedish);
  listlink = ^listtype;
  listtype = record
    link :listlink;
    filename :string;
  end; (* listtype *)

var
  language :national;
  listname :boolean;
  filelist :listlink;


procedure headline;
  (* Prints the screen heading *)

begin
  writeln(chr(12),'Text file printer');
  gotoxy(30,1);
  writeln('A-DATA 87');
end; (* headline *)


function langselect :national;
  (* Selects language.
     This affects both the text preceeding the date and the character set
     used by the printer. *)

var
  ch :char;

begin
```

```
  gotoxy(0,3);
  writeln('Select language');
  writeln('English/Swedish?');

  repeat
    gotoxy(17,4);
    read(ch);
  until ch in ['E','e','S','s'];

  gotoxy(17,4);
  case ch of
    'E','e' :begin
               langselect := english;
               writeln('English');
             end;
    'S','s' :begin
               langselect := swedish;
               writeln('Swedish');
             end;
  end; (* case *)
end; (* langselect *)


function shownames :boolean;
  (* Selects printing of file names *)

var
  ch :char;
  temp :boolean;

begin
  gotoxy(0,6);
  write('Print file names? ');
  repeat
    gotoxy(18,6);
    read(ch);
  until ch in ['Y','y','N','n'];
  temp := ch in ['Y','y'];
  shownames := temp;

  gotoxy(18,6);
  if temp then
    write('Yes')
  else
    write('No');
end; (* shownames *)


function build_name_list :listlink;
  (* Reads the file names to print and creates a linked list with the names.
     The maximal number of files is limited by available memory only. *)
```

```
var
  current :listlink;
  name :string;


  procedure readname(var name :string);

  begin
    write('File? ');
    readln(name);
  end; (* readname *)

begin (* build_name_list *)
  gotoxy(0,8);
  writeln('Enter file names to print.');
  writeln('Terminate with a blank name.');

  new(current);
  build_name_list := current;

  repeat
    readname(name);
    with current^ do
      if name<>'' then
      begin
        new(link);
        filename := name;
        current := link;
      end
      else
        link := nil;
  until name='';
end; (* build_name_list *)


procedure print_all(current :listlink; language :national; printnames :boolean);
  (* Lists all files *)

var
  outfile :text;


  procedure printfile
   (* Lists one file *)

    (    name :string;
     var destination :text;
         language :national;
         printnames :boolean);
```

```
const
  spaces = '                                                ';
  vol_inf_addr = 13974;       (* Address of volume info array *)

type
 (* These definitions are used to get access to the volume information
    array in the kernel global data area. *)

  vol_id_type = string[7];
  volume = record
    volume_id :vol_id_type;
    is_blocked :boolean;
    no_of_blocks :integer;
  end; (* volume *)
  vol_array = array[0..32] of volume;
  dual_vol = record
    case boolean of
      true  :(int :integer);
      false :(ptr :^vol_array);
  end; (* dual vol *)

var
  line :faststring;
  source :fastfile;
  index,
  iocode :integer;
  vol_info :dual_vol;
  prefix :vol_id_type;
  numeric :boolean;


  procedure ioreport(iocode :integer);
    (* Takes an I/O error code and translates it into a message *)

  begin (* ioreport *)
    write('  ERROR - ');
    case iocode of
      1: writeln('Bad block, parity error (CRC)');
      2: writeln('Bad device number');
      3: writeln('Illegal I/O request');
      4: writeln('I/O operation cancelled by user');
      5: writeln('Volume is no longer on-line');
      6: writeln('File is no longer in directory');
      7: writeln('Bad file name');
      8: writeln('No room, insufficient space on volume');
      9: writeln('No such volume on-line');
     10: writeln('No such file on volume');
     11: writeln('Duplicate directory entry');
     12: writeln('Not closed; attemp to close an open file');
     13: writeln('Not open; attemp to access a closed file');
     14: writeln('Bad format; error in reading real or integer');
```

```
    15: writeln('Ring buffer overflow');
    16: writeln('Volume is write-protected');
    17: writeln('Illegal block number');
    18: writeln('Illegal buffer');
   end; (* case *)
end; (* ioreport *)


procedure fixname
 (* First attempt to modify the source file name in order to make it what
    the user actually meant. *)
  (var name :string;
   var prefix :vol_id_type;
   var numeric :boolean;
   var index :integer);

const
  suffix = '.TEXT';
  pref_addr = 13824; (* Address in kernel data for default prefix. *)

type
  dual = record
    case boolean of
      true  :(addr :integer);
      false :(ptr :^vol_id_type);
  end; (* dual *)

var
  pref_window :dual;
  i :integer;

begin (* fix name *)
  upper_case(name,name);
  if pos(suffix,name)<>(length(name)-4) then
    (* Not ending with .TEXT already *)
    if name[length(name)]='.' then
      (* Trailing dot indicates no suffix *)
      delete(name,length(name),1)
    else
      name := concat(name,suffix);

  (* Now check if a volume id is given. If not, add the current prefix so
     it shows on the printouts. *)
  if scan(length(name),=':',name[1])=length(name) then
  begin
    pref_window.addr := pref_addr;
    prefix := pref_window.ptr^;
    numeric := false;
    if prefix[1]='#' then
    begin
```

```
        numeric := true;
        index := 0;
        for i := 2 to length(prefix) do
          index := index*10+ord(prefix[i])-48;
        prefix := volinfo.ptr^[index].volume_id;
      end;
      name := concat(prefix,':',name);
    end;
  end; (* fix name *)


 procedure refix_name(var name :string; prefix :vol_id_type; index :integer);
   (* Re-checks prefix if first attempt to open the file was unsuccessful.
      This might occur if the prefix is a drive, and the volume in that
      drive was changed just prior to running this program. *)

  var
    new_prefix :vol_id_type;
    colon :integer;

  begin (* refix name *)
    new_prefix := volinfo.ptr^[index].volume_id;
    if prefix<>new_prefix then
    begin
      colon := scan(length(name),=':',name[1])+1;
      name := concat(new_prefix,copy(name,colon,length(name)-colon+1));
    end;
  end; (* refix name *)


begin (* printfile *)
  volinfo.int := vol_inf_addr;
  fixname(name,prefix,numeric,index);
  (*$I-*)
  fastreset(source,name);
  iocode := ioresult;
  if (iocode=9) and numeric then
  begin
    refix_name(name,prefix,index);
    fastreset(source,name);
    iocode := ioresult;
  end;
  (*$I^*)

  writeln(name);      (* Report on screen *)
  if iocode=0 then    (* OK to print *)
  begin
    (* Page heading *)
    write(destination,spaces);
    case language of
      english : begin
```

```
                    engtext(line);
                    write(destination,'Created at ');
                  end;
        swedish : begin
                    swetext(line);
                    write(destination,'Utskrivet ');
                  end;
      end; (* case *)
      writeln(destination,line);
      if printname then
        writeln(destination,spaces,name);
      writeln(destination);

      while not source.end_of_file do
      begin
        fastreadln(source,line);
        writeln(destination,line);
      end; (* while *)
      fastclose(source);
      write(destination,chr(12));
      writeln('  --> PRINTER:');
    end
    else (* I/O error *)
      ioreport(iocode);
  end; (* printfile *)


  procedure initprinter(language :national);
    (* Sets font, margin and language *)

  begin
    elite(on);
    leftmargin(6);
    skipover(4);
    case language of
      english :international(USA);
      swedish :international(Sweden);
    end; (* case *)
  end; (* initprinter *)

begin (* print all *)
  writeln('----------------------------------------');
  rewrite(outfile,'printer:');
  initprinter(language);

  while current^.link<>nil do
    with current^ do
    begin
      printfile(filename,outfile,language,printnames);
      current := link;
```

```
    end; (* with *)
end; (* print all *)


begin
  headline;
  language := langselect;
  listname := shownames;
  filelist := build_name_list;

  print_all(filelist,language,listname);
end.
```

Disk 97. p-Code Units — Source 2

Version:
Requires: p-Code
Author: Anders Persson
Language: Pascal
Updated: 3/14/88

Second disk of source code and documentation for disk 95.

```
dskdir. v2.0. 12-dec-96
Disk name               = SIG-TX2
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 PASCAL      358 DIS/FIX128   >022 326 >003 031

p-System first block  = 0 (2)
Duplicate directories = yes
p-System disk name    = SIG-TX2
p-System block count  = 180
p-System file count   = 11
Date of last boot     = 20-Nov-87

File name -----  Blk  Date ----  Strt  End   Last  Type
AUTOSTART.TEXT   028  26-Nov-87  >00a  >025  >200  text
DISKMAP.TEXT     006  12-Jan-86  >026  >02b  >200  text
DISMACH.TEXT     006  24-Sep-84  >02c  >031  >200  text
DISPAS.TEXT      026  08-Jan-86  >032  >04b  >200  text
DISKINFO.TEXT    018  20-Nov-87  >04c  >05d  >200  text
MYDEBUG.TEXT     004  22-Oct-85  >05e  >061  >200  text
MINIMEMORY.TEXT  006  07-Dec-85  >062  >067  >200  text
MINITEST.TEXT    004  11-Feb-86  >068  >06b  >200  text
MINI.INFO.TEXT   006  04-Mar-87  >06c  >071  >200  text
SYS.MAP.TEXT     028  20-Nov-87  >072  >08d  >200  text
WELCOME.TEXT     004  04-Mar-87  >08e  >091  >200  text
```

Disk 97. Contents of file AUTOSTART.TEXT

```
(*$C Startup file for the p-system. A-DATA 871126. Anders Persson, Sweden *)
program autostart;
  (* Provides setup of new date, port configurations and default prefix.
     Installes I/O unit #10.
     Should be executed when the system is initiated.

     Code type must be M_9900!

     A-DATA 871125 *)

uses
  commandio,    (* Redirection *)
  misc,         (* String manipulation *)
  screenops,    (* Date change *)
  swedish;      (* Screen headings *)

type
  window = record
    case boolean of
      true:  (int: integer);
      false: (ptr: ^integer);
  end; (* window *)

var
  line: string;


procedure poke(addr,value :integer);
  (* Stores value at addr in CPU RAM *)

var
  memaddr :window;

begin (* poke *)
  memaddr.int := addr;
  memaddr.ptr^ := value;
end; (* poke *)


function peek(addr: integer): integer;
 (* Reads value at addr in CPU RAM *)

var
  memaddr: window;

begin (* peek *)
  memaddr.int := addr;
  peek := memaddr.ptr^;
```

```
end; (* peek *)


procedure setport
 (* Connects a logical p-system I/O unit with a physical device. *)

  (unitno: integer;     (* Logical I/O unit # *)
   pabname: string);    (* Physical device *)

const
  rddata  = -30720;
  wrtdata = -29696;
  wrtaddr = -29694;
  wrtenab =  16384;
  pabtbl  =  10716;

type
  byte = 0..255;

var
  curlen,
  period,
  vdpaddr,
  savevdpaddr,
  i: integer;


  procedure wrtvdpaddr (vdpaddr : integer);
   (* This procedure initiates the VDP ram chip to read/write from the
      address passed in the parameter vdpaddr.  *)

    procedure swapbyte(var x:integer);
     (* This procedure takes a word and reverses the order of the bytes *)

    type
      byteword = record
        case boolean of
          true: (addr:integer);
          false:(bytes: packed array[1..2] of byte);
        end;

    var
      word:  byteword;
      tbyte: byte;

    begin
      with word do
      begin
        addr := x;
        tbyte := bytes[1];
```

```
        bytes[1] := bytes[2];
        bytes[2] := tbyte;
        x := addr;
      end;  (* with statement *)
    end;  (* swapbyte *)


  begin (* wrtvdpaddr *)
    swapbyte( vdpaddr );
    poke(wrtaddr,vdpaddr);
    swapbyte( vdpaddr );
    poke(wrtaddr,vdpaddr);
  end;  (* wrtvdpaddr *)


  function rdvdp (var vdpaddr : integer) : integer;

   (* This function reads a byte of data from the VDP ram address specified *
    * in the parameter vdpaddr.  *)

  begin
    wrtvdpaddr( vdpaddr );
    rdvdp := peek(rddata) div 256;  (* Right justify byte in word *)
    vdpaddr := vdpaddr + 1;
  end;  (* rdvdp *)


  procedure wrtvdp( var vdpaddr : integer;
                        data    : integer);
   (* This procedure writes the byte of data passed in the parameter *
    * data to the VDP ram address specified in vdpaddr.  *)

  var
    temp : integer;

  begin
    temp := vdpaddr + wrtenab;  (* Write enable the address *)
    wrtvdpaddr(temp);
    poke(wrtdata,data*256); (* Left justify byte in word and write *)
    vdpaddr := vdpaddr + 1;
  end; (* wrtvdp *)


begin (* setport *)
(*$R-*)
  (* Find the address of the PAB entry in VDP ram. If no device
     currently defined, do nothing. *)

  vdpaddr := peek(pabtbl+unitno*2)+17;
  savevdpaddr := vdpaddr;
  curlen := rdvdp(vdpaddr);
```

```
  if curlen>0 then     (* Set the new type *)
  begin
    period := 0;
    for i := 1 to length(pabname) do
    begin
      if (period=0) and (pabname[i]='.') then
         period := i-1;
      if pabname[i] in ['a'..'z'] then
         pabname[i] := chr(ord(pabname[i])-32);
    end; (* for *)

    if period=0 then
       period := length(pabname);
    vdpaddr := savevdpaddr-17;
    for i := 1 to 2 do
       wrtvdp(vdpaddr,0);
    wrtvdp(vdpaddr, period);
    for i := 1 to 3 do
       wrtvdp(vdpaddr, 0);
    vdpaddr := savevdpaddr;
    for i := 0 to length(pabname) do
       wrtvdp(vdpaddr, ord(pabname[i]));
    unitclear(unitno);
  end;
(*$R^*)
end; (* setport *)


procedure datecheck;
 (* Displays the current system date and allows the user to change it, if
    required. *)

const
  sysdate = 13840;

type
  datetype = sc_date_rec;
  convdate = record
    case boolean of
      true:  (int: integer);
      false: (dat: datetype);
  end; (* convdate *)

  dirtype = packed record
    fill1: packed array[0..19] of char;
    the_date: datetype;
    fill2: packed array[0..489] of char;
  end; (* dirtype *)
```

```
  texttab = array[0..12] of string[3];


var
  twindate: convdate;
  newdate:  datetype;
  gooddate: boolean;
  datein,
  dateout:  string;
  info:     sc_info_type;
  directory:dirtype;
  monthtab: texttab;


  function days_in_month(mo,yr:integer):integer;

  (* Calculates maximal number of days in a month *)

  begin (* days in month *)
    case mo of
      1,3,5,7,8,10,12: days_in_month := 31;
             4,6,9,11: days_in_month := 30;
                    2: if (yr mod 4)=0 then
                         days_in_month := 29
                       else
                         days_in_month := 28
    end (* case *)
  end; (* days in month *)


  function stringtodate(datein:string; currdate:datetype;
                        var newdate:datetype):boolean;

  (* Stringtodate converts the entered date to the format used in the diskette
     directory. The function also performs error checking on the input and re-
     turns as the function value a boolean, which is true if the date is "good".
     In case just part of a new date was entered, e. g. only the month, the other
     parts remain unchanged. *)

  var inlen,hyppos1,hyppos2,index,tempdate:integer;
      ishyp1,ishyp2,monthok:boolean;
      tempch:string;

  begin (* string to date *)
    newdate := currdate;
    stringtodate := true;
    inlen := length(datein);

    (* Find the first hyphen in datein *)

    hyppos1 := scan(inlen,='-',datein[1]);
```

```
ishyp1 := hyppos1<inlen;

tempch := copy(datein,1,hyppos1);

(* Convert to internal date format, if possible *)

if tempch<>'' then
begin
  if length(tempch)=1 then
    tempch := concat('0',tempch);

  if (length(tempch)=2) and (tempch[1] in ['0'..'3']) and
                            (tempch[2] in ['0'..'9']) then
    tempdate := (ord(tempch[1])-48)*10+ord(tempch[2])-48
  else
  begin
    stringtodate := false;
    exit(stringtodate)
  end;

  (* Date value range check *)

  if (tempdate<1) or (tempdate>31) then
  begin
    stringtodate := false;
    exit(stringtodate)
  end
  else
    newdate.day := tempdate;
end;

if not ishyp1 then
begin
  stringtodate := newdate.day<=daysinmonth(newdate.month,newdate.year);
  exit(stringtodate)  (* If no hyphen at all: exit *)
end;

(* Find second hyphen in datein *)

hyppos2 := scan(inlen-hyppos1-1,='-',datein[hyppos1+2]);
ishyp2 := hyppos2<(inlen-hyppos1-1);

tempch := copy(datein,hyppos1+2,hyppos2);

if length(tempch) in [1..2] then
begin
  stringtodate := false;      (* Not a valid month *)
  exit(stringtodate)
end;
```

```
  tempch := copy(tempch,1,3);   (* Use first three characters only *)
  upper_case(tempch,tempch);

  (* Calculate month number, if possible *)

  monthok := false;
  index := 1;
  while (index<13) and not monthok do
    if tempch=monthtab[index] then
    begin
      newdate.month := index;
      monthok := true
    end
    else
      index := index+1;

  if (tempch<>'') and not monthok then
  begin
    stringtodate := false;      (* Not a valid month entered *)
    exit(stringtodate)
  end;

  if not ishyp2 then
  begin
    stringtodate := newdate.day<=daysinmonth(newdate.month,newdate.year);
    exit(stringtodate)         (* If no second hyphen: exit *)
  end;

  tempch := copy(datein,hyppos1+hyppos2+3,inlen-hyppos1-hyppos2-2);

  (* Convert the year to internal format, if possible *)

  if tempch<>'' then
  begin
    if length(tempch)=1 then
      tempch := concat('0',tempch);

    if (length(tempch)=2) and (tempch[1] in ['0'..'9']) and
                              (tempch[2] in ['0'..'9']) then
      newdate.year := (ord(tempch[1])-48)*10+ord(tempch[2])-48
    else
      stringtodate := false;

    stringtodate := newdate.day<=daysinmonth(newdate.month,newdate.year);
  end;
end; (* date to string *)


procedure inimontab(var monthtab:texttab);

(* Inimontab initializes a table with the months. The table is used by
```

```
    stringtodate when converting between character and numeric representation
    of the months. *)

  begin (* inimontab *)
    monthtab[1] := 'JAN';
    monthtab[2] := 'FEB';
    monthtab[3] := 'MAR';
    monthtab[4] := 'APR';
    monthtab[5] := 'MAJ';
    monthtab[6] := 'JUN';
    monthtab[7] := 'JUL';
    monthtab[8] := 'AUG';
    monthtab[9] := 'SEP';
    monthtab[10] := 'OKT';
    monthtab[11] := 'NOV';
    monthtab[12] := 'DEC';
  end; (* ini mon tab *)


  procedure datetostring(thedate:datetype; var dateout:string);

  (* Datetostring converts the internal date format to the display format.
     The array monthtab is not used, because the month is displayed lower-case
     but may be entered as lower- or upper-case. *)

  var yearchar,monthchar,datechar:string;

  begin
    with thedate do
    begin
      str(year,yearchar);
      str(day,datechar);
      case month of
         1:monthchar := 'Jan';
         2:monthchar := 'Feb';
         3:monthchar := 'Mar';
         4:monthchar := 'Apr';
         5:monthchar := 'Maj';
         6:monthchar := 'Jun';
         7:monthchar := 'Jul';
         8:monthchar := 'Aug';
         9:monthchar := 'Sep';
        10:monthchar := 'Okt';
        11:monthchar := 'Nov';
        12:monthchar := 'Dec';
      end;

      dateout := concat(datechar,'-',monthchar,'-',yearchar);
    end;
  end; (* date to string *)
```

```
begin (* date check *)
  inimontab(monthtab);
  (* Display the current system date *)
  twindate.int := peek(sysdate);
  date_to_string(twindate.dat,dateout);
  gotoxy(0,8);
  writeln('Nuvarande datum ',dateout);

  (* Accept a new date *)
  good_date := true;
  repeat
    gotoxy(0,9);
    write('Nytt datum? ',chr(141));
    readln(datein);

    write(chr(141));
    if length(datein)>0 then    (* Something entered? *)
    begin
      good_date := string_to_date(datein,twindate.dat,newdate);
      if not good_date then
        write('Felaktigt datum');
    end;
  until good_date;

  (* Check if the date is changed *)
  if (length(datein)>0) and (newdate<>twindate.dat) then
  begin
    (* Change date on the system disk *)
    unitread(4,directory,512,2);
    directory.thedate := newdate;
    unitwrite(4,directory,512,2);

    (* Change screenops date *)
    sc_use_info(sc_get,info);
    info.sc_date := newdate;
    sc_use_info(sc_give,info);

    (* Change the system date *)
    twindate.dat := newdate;
    poke(sysdate,twindate.int);

    date_to_string(newdate,dateout);
    write('Nytt datum: ',dateout);
  end;
end; (* datecheck *)


procedure unit_10;
 (* Installes unit #10 in the system *)

const
```

```
  pointaddr = 11316;    (* Entry in unit type pointer table *)
  descraddr = 14094;    (* Address of unit descriptor *)

type
  dvidtype = string[7];
  dirtype = record
    fill1,
    dlastblk,
    dfirstblk :integer;
    dvid :dvidtype;
    deovblk :integer;
  end; (* dirtype *)

  unittype = record
    volume :dvidtype;
    is_blocked :boolean;
    blocks :integer;
  end; (* unittype *)

  dual = record
    case boolean of
      true  :(int :integer);
      false :(ptr :^unittype);
  end; (* dual *)

var
  window :dual;
  dir :dirtype;

begin (* unit 10 *)
(*$R-*)
  (* Insert blocked unit pointer in pointer table *)
  poke(pointaddr,peek(pointaddr-2));
  (* Load something to begin with *)
  window.int := descraddr;
  unitclear(10);
  unitread(10,dir,sizeof(dir),2,0);   (* Fetch actual values *)
  with window.ptr^ do
  begin
    if (ord(dir.dvid[0])<=7) and (ioresult=0) then
    begin
      volume := dir.dvid;
      blocks := dir.deovblk;
    end
    else
    begin
      volume := '';
      blocks := 0;
    end;
  end; (* with *)
```

```
(*$R^*)
end; (* unit 10 *)


begin (* auto start *)
  page(output);
  gotoxy(10,1);
  write('V [ L K O M M E N');
  gotoxy(10,2);
  write('till p-systemet.');
  gotoxy(30,23);
  write('A-DATA 87');

  datecheck;    (* Show date and read new date *)
  gotoxy(0,12);
  write('Installerar I/O enhet #10.');
  unit_10;      (* Install fourth drive *)

  gotoxy(0,14);
  writeln('Modifierar kommunikationsportarna.');
  line := 'RS232/2.BA=4800.DA=8.PA=N';
  writeln('Printer: ',line);
  setport(6,line);
  line := 'RS232.BA=2400.DA=8.PA=N';
  writeln('Remote:  ',line);
  setport(7,line);

  gotoxy(0,18);
  write('Standard prefix {r #5.');
  if not redirect('p=#5') then
    exception(true);
end.
```

Disk 97. Contents of file DISKMAP.TEXT

```
program test;

uses support;

type
  dirtype  = packed record
    name   :packed array[0..9] of char;
    sector :integer;
    fill1  :packed array[0..43] of char;
    map    :packed array[0..1599] of boolean;
  end;

var
  dir   :dirtype;
  drive,
  i,
  code  :integer;


procedure mapswap(var dir:dirtype);

var
  charr:packed array[0..199] of char;
  ch   :char;
  i    :integer;

begin
  with dir do
  begin
    moveleft(map,charr,200);
    i := 0;
    while i<200 do
    begin
      ch := charr[i];
      charr[i] := charr[i+1];
      charr[i+1] := ch;
      i := i+2;
    end;
    moveleft(charr,map,200);
  end; (* with *)
end; (* mapswap *)


begin
  set_pattern(128,'');
  set_pattern(129,'E0E0E0E0E0E0E0E0');
  set_pattern(130,'1C1C1C1C1C1C1C1C');
  set_pattern(131,'FCFCFCFCFCFCFCFC');
```

```
  write('Read from what unit? ');
  readln(drive);
  unitread(drive,dir,256,0);
  mapswap(dir);
  with dir do
  begin
    i := 0;
    while i<sector-1 do
    begin
      code := 128;
      if map[i] then
        code := code+1;
      if map[i+1] then
        code := code+2;
      i := i+2;
      write(chr(code));
      if (i mod 78)=0 then
        writeln;
    end; (* while *)
    writeln;
    writeln('Disk ',name,' has ',sector,' sectors');
  end; (* with *)
end.
```

Disk 97. Contents of file DISMACH.TEXT

```
;Routines used to speed up disassembly
;A-DATA 840924
;Linked to DISPAS, producing DISASSEM as an executable file
;-----------------
;function peek(addr:integer):integer;
;Returns the 16-bit value at any address

         .FUNC PEEK,1

         MOV  *R10+,R0
         MOV  *R0,*R10
         B    *R11

;-----------------

;procedure dec_hex(dec:integer, var hex:string);

         .PROC DEC_HEX,2

         MOV  *R10+,R1      ;Fetch string pointer
         MOV  *R10+,R2      ;Fetch decimal value

         LI   R0,10H        ;Store shift count
         LI   R5,0400H
         MOVB R5,*R1+       ;Store string length
         SWPB R5            ;Use as nybble count

CONVLOOP MOV  R2,R4
         S    R5,R0         ;Reduce shift count
         SRC  R4,0          ;Full word
         ANDI R4,0FH        ;Mask out four bits
         MOVB @HEXTAB(R4),*R1+  ;Store character in string
         MOV  R0,R0
         JNE  CONVLOOP      ;If shiftcount=0 then finished
         B    *R11

HEXTAB   .ASCII  "0123456789ABCDEF"

;--------------------

;procedure int_text(value:integer; var text:string);

         .PROC INT_TEXT,2

         MOV  *R10+,R1      ;Fetch string pointer
         MOV  *R10+,R2      ;Fetch value
         MOV  R11,R6        ;Save return address
         MOV  R1,R3         ;Second copy of string pointer
```

```
         LI   R4,3          ;Byte count
         LI   R5,BLANK      ;Text to load
INITLOOP MOVB *R5+,*R3+     ;Initialize string as two spaces
         DEC  R4
         JNE  INITLOOP

         INC  R1            ;Point to first character
         BL   CHECK         ;Check if printable
         SWPB R2
         INC  R1
         BL   CHECK         ;Check next character
         B    *R6           ;Return to Pascal host

;Routine used to check if a byte can be represented as a printable character
;CHECK assumes that the byte is MSB of R4, and that R1 is the string pointer
CHECK    MOV  R2,R4         ;Fetch byte
         CI   R4,2100H      ;Lower than first printable?
         JL   OUT
         CI   R4,7F00H      ;Higher than last printable?
         JH   OUT
         MOVB R4,*R1        ;Use character
OUT      B    *R11

BLANK    .BYTE 2,20H,20H,0  ;String with two spaces
         .END
```

Disk 97. Contents of file DISPAS.TEXT

```
(* Disassembler *)
(* A-DATA 840921 *)
(*$C A-DATA Disassembler 840921*)

program disassembler;

uses misc;

const numofform=13;     (* Number of entries in the format table *)

type formentry=record
                opcode:integer;
                format:integer;
               end;
     gadrtype=0..3;     (* General address type indicator *)
     gadr=0..15;        (* General address field *)

var pc,stoppc,word,opform,i:integer;
    filename,newname,hexpc,disline:string;
    hextab:string[16];
    ch:char;
    found:boolean;
    outfile:text;
    formtab:array[0..numofform]of formentry;

(* Disline and pc are globals. The procedure cominfo assigns the information
   that is displayed for each line to disline. *)


function peek(addr:integer):integer; external;
  (* Returns word at addr in memory *)


function bufscan:integer;  external;
  (* Gives the ASCII code of the next character in the input buffer, or -1 if
     the buffer is empty. *)


procedure dec_hex(dec:integer; var hex:string); external;
  (* Converts from decimal to hexadecimal notation *)


function hex_dec(hex:string):integer;
  (* Convert from hexadecimal to decimal notation *)

var temp,factor,i:integer;

begin
```

```
  upper_case(hex,hex);
  factor := 4096;
  temp := 0;
  for i := 1 to 4 do
  begin
    temp := temp+scan(16,=hex[i],hextab[1])*factor;
    factor := factor div 16
  end;
  hex_dec := temp
end;


function checkkey:boolean;
  (* Returns true if 'S' pressed, and delays execution if space is pressed *)

var value:integer;

begin
  checkkey := false;
  value := bufscan;
  if value<>-1 then
    if value in [83,115] then  (* Value is 'S','s' *)
      checkkey := true
    else if value=32 then
      while bufscan<>32 do;
end; (* checkkey *)


procedure int_text(value:integer; var text:string); external;
  (* Convert integer to two characters, if printable *)


procedure cominfo(pc,word:integer);
  (* Calculates the address and value in hex format *)

var addr,value,txt:string;

begin
  dec_hex(pc-2,addr);
  dec_hex(word,value);
  int_text(word,txt);

  disline := concat(addr,' ',value,' ',txt,' ');
end;


procedure genadr(t:gadrtype; r:gadr; var address:string);

(* Determines a general address *)

var temp:integer;
```

```
    tempstr,strreg:string;

begin
  str(r,strreg);
  case t of
    0: address := concat('R',strreg);
    1: address := concat('*R',strreg);
    2: begin
         temp := peek(pc);
         pc := pc+2;
         dec_hex(temp,tempstr);
         address := concat('@',tempstr);
         if r>0 then
           address := concat(address,'(R',strreg,')');
       end;
    3: address := concat('*R',strreg,'+');
  end;
end;


procedure catchup(savepc:integer);

(* Catches up eventual words used by general address formats *)

var word:integer;

begin
  while pc>savepc do
  begin
    word := peek(savepc);
    savepc := savepc+2;
    cominfo(savepc,word);
    writeln(outfile,disline);
  end;
end;


procedure badformat(word:integer);

(* In case of undefined instructions *)

begin
  cominfo(pc,word);
  writeln(outfile,disline,'UNDEFINED');
end;


procedure format1(word:integer);

(* Disassembles format 1 *)
```

```
type opco=0..15;
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                           s:gadr;
                           ts:gadrtype;
                           d:gadr;
                           td:gadrtype;
                           opcode:opco;
                          end);
            end;

var instr:format;
    mnem,source,dest:string;
    savepc:integer;

begin
  savepc := pc;
  cominfo(pc,word);
  instr.int := word;
  with instr.oper do
  begin
    case opcode of   (* Determine operation *)
      4: mnem := 'SZC  ';
      5: mnem := 'SZCB ';
      6: mnem := 'S    ';
      7: mnem := 'SB   ';
      8: mnem := 'C    ';
      9: mnem := 'CB   ';
     10: mnem := 'A    ';
     11: mnem := 'AB   ';
     12: mnem := 'MOV  ';
     13: mnem := 'MOVB ';
     14: mnem := 'SOC  ';
     15: mnem := 'SOCB ';
    end;

    genadr(ts,s,source);
    genadr(td,d,dest);
    writeln(outfile,disline,mnem,source,',',dest);
    catchup(savepc);
  end;
end;


procedure format2(word:integer);

(* Disassembles format 2 *)

type part=0..255;
```

```
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                           disp:part;
                           opcode:part;
                          end);
            end;

var instr:format;
    mnem,addr:string;
    tempint:integer;

begin
  cominfo(pc,word);
  instr.int := word;

  with instr.oper do
  begin
    case opcode of
     16: mnem := 'JMP  ';
     17: mnem := 'JLT  ';
     18: mnem := 'JLE  ';
     19: mnem := 'JEQ  ';
     20: mnem := 'JHE  ';
     21: mnem := 'JGT  ';
     22: mnem := 'JNE  ';
     23: mnem := 'JNC  ';
     24: mnem := 'JOC  ';
     25: mnem := 'JNO  ';
     26: mnem := 'JL   ';
     27: mnem := 'JH   ';
     28: mnem := 'JOP  ';
     29: mnem := 'SBO  ';
     30: mnem := 'SBZ  ';
     31: mnem := 'TB   ';
    end;

    tempint := disp;
    if tempint>=128 then        (* Signed displacement *)
      tempint := tempint-256;

    if opcode>=29 then
      str(tempint,addr)         (* CRU *)
    else
    begin
      tempint := pc+tempint*2;  (* JUMP *)
      dec_hex(tempint,addr)
    end;
```

```
    writeln(outfile,disline,mnem,addr);
  end; (* with *)
end; (* format2 *)


procedure format349(word:integer);

(* Handles formats 3, 4 and 9, because of similarities *)

type opco=0..63;
     format=record
              case boolean of
               false:(int:integer);
               true :(oper:packed record
                           s:gadr;
                           ts:gadrtype;
                           d:gadr;
                           opcode:opco;
                          end);
            end;

var instr:format;
    mnem,source,dest:string;
    savepc:integer;

begin
  savepc := pc;
  cominfo(pc,word);
  instr.int := word;

  with instr.oper do
  begin
    case opcode of
       8: mnem := 'COC  ';
       9: mnem := 'CZC  ';
      10: mnem := 'XOR  ';
      11: mnem := 'XOP  ';
      12: mnem := 'LDCR ';
      13: mnem := 'STCR ';
      14: mnem := 'MPY  ';
      15: mnem := 'DIV  ';
    end; (* case *)

    genadr(ts,s,source);
    case opcode of
      8,9,10,14,15: genadr(0,d,dest);
      11,12,13: str(d,dest);
    end; (* case *)
    writeln(outfile,disline,mnem,source,',',dest);
    catchup(savepc);
  end; (* with *)
```

```
end; (* format349 *)

procedure format5(word:integer);

type opco=0..255;
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                           w:gadr;
                           c:gadr;
                           opcode:opco;
                          end);
            end;

var instr:format;
    mnem,reg,count:string;

begin
  cominfo(pc,word);
  instr.int := word;

  with instr.oper do
  begin
    case opcode of
       8: mnem := 'SRA  ';
       9: mnem := 'SRL  ';
      10: mnem := 'SLA  ';
      11: mnem := 'SRC  ';
    end; (* case *)

    genadr(0,w,reg);
    str(c,count);

    writeln(outfile,disline,mnem,reg,',',count);
  end; (* with *)
end; (* format5 *)


procedure format6(word:integer);

(* Disassembles format 6 *)

type opco=0..1023;
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                           s:gadr;
                           ts:gadrtype;
```

```
                        opcode:opco;
                end);
        end;

var instr:format;
    mnem,source:string;
    savepc:integer;

begin
  savepc := pc;
  cominfo(pc,word);
  instr.int := word;

  with instr.oper do
  begin
    case opcode of
      16: mnem := 'BLWP ';
      17: mnem := 'B    ';
      18: mnem := 'X    ';
      19: mnem := 'CLR  ';
      20: mnem := 'NEG  ';
      21: mnem := 'INV  ';
      22: mnem := 'INC  ';
      23: mnem := 'INCT ';
      24: mnem := 'DEC  ';
      25: mnem := 'DECT ';
      26: mnem := 'BL   ';
      27: mnem := 'SWPB ';
      28: mnem := 'SETO ';
      29: mnem := 'ABS  ';
    end; (* case *)

    genadr(ts,s,source);
    writeln(outfile,disline,mnem,source);
    catchup(savepc);
  end; (* with *)
end; (* format6 *)


procedure format7(word:integer);

(* Disassembles format 7 *)

type opco=0..2047;
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                          filler:0..31;
                          opcode:opco;
                    end);
```

```
            end;

var instr:format;
    mnem:string;

begin
  cominfo(pc,word);
  instr.int := word;

  with instr.oper do
  begin
    case opcode of
      25: mnem := 'UNDEFINED'; (* Easiest way to find out *)
      26: mnem := 'IDLE';
      27: mnem := 'RSET';
      28: mnem := 'RTWP';
      29: mnem := 'CKON';
      30: mnem := 'CKOF';
      31: mnem := 'LREX';
    end; (* case *)

    writeln(outfile,disline,mnem);
  end; (* with *)
end; (* format7 *)


procedure format8(word:integer);

(* Disassembles format 8 *)

type opco=0..2047;
     format=record
             case boolean of
              false:(int:integer);
              true :(oper:packed record
                          w:gadr;
                          filler:0..1;
                          opcode:opco;
                    end);
            end;

var instr:format;
    mnem,reg,imm:string;
    savepc,tempint:integer;

begin
  savepc := pc;
  cominfo(pc,word);
  instr.int := word;
```

```
  with instr.oper do
  begin
    case opcode of
      16: mnem := 'LI   ';
      17: mnem := 'AI   ';
      18: mnem := 'ANDI ';
      19: mnem := 'ORI  ';
      20: mnem := 'CI   ';
      21: mnem := 'STWP ';
      22: mnem := 'STST ';
      23: mnem := 'LWPI ';
      24: mnem := 'LIMI ';
    end; (* case *)

    case opcode of
     16,17,18,19,20:
       begin
         genadr(0,w,reg);
         tempint := peek(pc);
         pc := pc+2;
         dec_hex(tempint,imm);
         writeln(outfile,disline,mnem,reg,',',imm)
       end;

     21,22: begin
              genadr(0,w,reg);
              writeln(outfile,disline,mnem,reg)
            end;

     23,24: begin
              tempint := peek(pc);
              pc := pc+2;
              if opcode =23 then (* If LWPI *)
                dec_hex(tempint,imm)
              else
                str(tempint mod 16,imm);   (* 4 bits only for the int.mask *)
              writeln(outfile,disline,mnem,imm);
            end;
    end; (* case *)
    catchup(savepc);
  end; (* with *)
end; (* format8 *)



begin (* Main program *)

(* Initialize table to determine instruction formats *)

  formtab[0].opcode := 0;
  formtab[1].opcode := 512;
```

```
  formtab[2].opcode := 769;
  formtab[3].opcode := 832;
  formtab[4].opcode := 1024;
  formtab[5].opcode := 1920;
  formtab[6].opcode := 2048;
  formtab[7].opcode := 3072;
  formtab[8].opcode := 4096;
  formtab[9].opcode := 8192;
  formtab[10].opcode := 11264;
  formtab[11].opcode := 12288;
  formtab[12].opcode := 14336;
  formtab[13].opcode := 16384;

  formtab[0].format := 1;
  formtab[1].format := 0;
  formtab[2].format := 8;
  formtab[3].format := 0;
  formtab[4].format := 7;
  formtab[5].format := 6;
  formtab[6].format := 0;
  formtab[7].format := 5;
  formtab[8].format := 0;
  formtab[9].format := 2;
  formtab[10].format := 3;
  formtab[11].format := 9;
  formtab[12].format := 4;
  formtab[13].format := 9;

  hextab := '0123456789ABCDEF';     (* Global, used in conversions *)
  filename := 'CONSOLE:';   (* Default output filename *)

  writeln('A-DATA disassembler');

  repeat
    write('D(isass,Q(uit');
    repeat
      read(keyboard,ch);
    until ch in ['q','Q','d','D'];
    writeln;

(* Stop program here when desired *)

    if ch in ['q','Q'] then
      exit(disassembler);

    writeln;
    write('Output file [',filename,']? ');
    readln(newname);
    if length(newname)>0 then
      filename := newname;
```

```
    repeat
      writeln;
      write('Startaddress (4 hex digits)? ');
      readln(hexpc);
    until length(hexpc)=4;
    pc := (hex_dec(hexpc) div 2)*2;
    repeat
      writeln;
      write('Stopaddress  (4 hex digits)? ');
      readln(hexpc);
    until length(hexpc)=4;
    stoppc := (hex_dec(hexpc) div 2)*2;

(* Start the disassembly *)
    rewrite(outfile,filename);
    page(outfile);
    repeat
      word := peek(pc);
      pc := pc+2;

(* Determine instruction format *)
      opform := 1;
      i := 0;
      found := false;

      while (i<=numofform) and not found do
      begin
        if word<formtab[i].opcode then
        begin
          opform := formtab[i].format;
          found := true
        end
        else
          i := i+1;
      end;

(* Disassemble instruction *)
      case opform of
        0:badformat(word);
        1:format1(word);
        2:format2(word);
        3:format349(word);
        4:format349(word);
        5:format5(word);
        6:format6(word);
        7:format7(word);
        8:format8(word);
        9:format349(word);
      end;
    until (pc>=stoppc) or checkkey;
```

```
  close(outfile,lock);
  until false;          (* Keep on until quit *)
end.
```

Disk 97. Contents of file DISKINFO.TEXT

DISK INFORMATION

Due to space considerations, the files are placed on the disks in any order. Since this distribution is different for users with different disk capacity, I've avoided to say anything about on which disk you find a certain file. Please list the directory to find out which files your disks are holding.

DEBUGGER

MYDEBUG.TEXT is the source code of a unit which gives a "kinder" response if you accidentally press D(ebug at the command level. SYSTEM.PASCAL contains the debugger, among other things. If you have this SYSTEM.PASCAL on the disk in drive #4 (DSK1) at boot-time, this debugger replacement will be used automatically.

CHARACTER DEFINITIONS

SYSTEM.CHARAC contains definitions of proper lower case letters. Must be available in #4 when booting.

MISCELLANEOUS INFORMATION

SYSTEM.MISCINFO defines some key codes. If this file is present on #4 when booting, BREAK is moved from FCTN-4 to CTRL-Q. FCTN-4 is better used for handling communication problems with troublesome printers and similar.

DISK FORMATTING

DFORMAT.CODE is an executable file. It appears almost identical to the DFORMAT file described in the Utilities manual. The difference is that this one is able to format both double sided and double density disks, provided that your controller handles the latter. The program has been tested with a Cor Comp card, but not yet with the Myarc version. I suppose it will work, although I don't know which DD format it creates. FORMATTER.TEXT (Pascal) and FORMASM.TEXT (assembly) contains the source code.

DISASSEMBLER

DISASSEM.CODE is a disassembler. You can use it if you want to inspect memory contents under the p-system. You can't disassemble a code file which isn't loaded. However, any module with a ROM might be inspected, since Pascal is independent of the module currently plugged into the computer. DISPAS.TEXT (Pascal) and DISMACH.TEXT (assembly) contains the source code.

SECTOR MAP

DISKMAP.CODE gives a graphic display of used and available sectors on a disk. Used sectors are represented by a black rectangle, available by a blank one. All sectors are always used if you look at Pascal disks, but disks with BASIC programs and other files created with the proprietary operating system might have any distribution of used and available sectors. DISKMAP.TEXT contains the source. The idea behind this program is to show how easy it is to read a certain sector directly from Pascal.

SYSTEM INFORMATION

SYS.MAP.TEXT is a file intended to be read by the editor, or printed with the filer. The file might be of interest to those programmers who want to use or alter some of the facilities provided by the operating system. Users interested in incorporating a RAM disk with the p-system, changing the system date or file name prefix, should find the file useful. If your own explorations has revealed information not currently present in this file, I'm very interested in hearing about your discoveries.

ERROR MESSAGE WINDOWS

ERRORUNIT.CODE is a unit which allows displaying messages in windows on the screen. Any message can be shown, although the unit was developed with error messages in mind. The window can be placed anywhere on the screen, even on top of the text already present. When the message has been read, the window is removed, and the text hiding under it is automatically recovered. Given the starting position and the width of the window, the program automatically calculates the required height. Any ordinary string can be passed as an argument. A word wrap function prevents words from being broken at the end of the line. The window display feature is implemented entirely in assembly language, in order to give a lightning fast operation.

ERRORPAS.TEXT (Pascal) and ERRORASM.TEXT (assembly) is the source code. See the file ERRORINFO.TEXT for further information.

USE MINI MEMORY

MINIMEMORY.CODE contains some assembly routines which allows allocation of dynamic variables in the Mini Memory RAM. This way, your computer can handle more data or a longer program. MINIMEMORY.TEXT is the source code. Some programs may work only if this extra space is available, while others just work better. To make the latter kind easy to write, the Mini Memory routines were designed to allow the same program to adapt itself to the current memory situation. See the file MINI.INFO.TEXT for further information.

ADDITIONAL SCREEN HANDLING

Anyone who ever tried any serious Pascal program knows that screen I/O leaves much to be desired. Especially reading numeric values is hazardous, with a great risk for execution errors.

EXTRASCR.CODE is a unit which handles some of these problems. On the input side, it provides essentially the same functions as the ACCEPT VALIDATE in Extended BASIC. There are also features here that aren't provided by BASIC.

Routines corresponding to CALL KEY and CALL GCHAR are also present. When it comes to output, which in general is much less sensitive to user errors, horizontal and vertical character repetition (like CALL HCHAR and CALL VCHAR) is available, together with the ability to scroll parts of the screen.

EXTRAPAS.TEXT (Pascal) and EXTRAASM.TEXT (assembly) holds the source code. EXTRAINFO.TEXT describes the use of the routines.

PASCAL LIBRARY

SYSTEM.LIBRARY is the normal library, enhanced with the units above (extrascren, showerrors and minimemory). By placing this SYSTEM.LIBRARY on the boot disk (in #4), you'll always have these tools available.

WELCOME MESSAGE

WELCOME.CODE is the program you should have run prior to reading this, at least if you followed the written note. WELCOME.TEXT holds the source.

That's all for now. Please get in touch if you encounter any problems, have suggestions of further improvements or have programs of your own.

The software on this disk is placed in the public domain. However, the amount of detective work required to create some of the files stored on this disk was considerable. I'm myself sending this disk, which is continuously updated and expanded, to anyone who gives me any information about the UCSD p-system in general and especially the 99/4A implementation.

Hence, I'm very interested in anything you might have or know about Pascal for the TI 99/4A. Feel free to pass this on to other people; they might be interested too. But don't expect everybody else to do everything for you — give something in return!

Anders Persson
Kamnarsvagen 4:1078
S-222 45 LUND
SWEDEN

Phone:
Nat 046-11 69 19
Int +46-46 11 69 19

Disk 97. Contents of file MYDEBUG.TEXT

```
(*{$L debuglist.text}*)
{$U-}
unit debugger;

interface

procedure debug;
procedure dummy1;

implementation

  procedure debug;

  begin
    gotoxy(0,2);
    writeln('Sorry, I''m not real');
    writeln('There is no debugger in this system');
  end;

  procedure dummy1;

  begin
  end;

end.
```

Disk 97. Contents of file MINIMEMORY.TEXT

```
;----------------------------
;
; Mini memory allocation routines.
; Allows access of Mini memory to Pascal programs
;
; Contains:
; function initmini:boolean;   external;
; This function MUST be called before use of MMM, and returns true if MMM is
; available.
;
; function mininew(var pointer; size:integer):boolean;   external;
; Allocates space for the variable pointer^ in MMM and returns true if
; possible. Returns false if no MMM or if no space left.
; Example of calling:
; if not mininew(point,sizeof(type_pointed_to)) then
;    new(point);
;
; function inmini(var pointer):boolean;  external;
; Returns true if the pointer points to a variable which resides in MMM.
; Used for disposing.
; Example:
; if not inmini(point) then
;    dispose(point);
;
;  A-DATA 851207
;
;----------------------------


SP       .EQU 10
LINK     .EQU 11

FREE     .EQU 7000H
LOMINI   .EQU FREE+2
HIMINI   .EQU 8000H


         .RELFUNC INITMINI

         CLR  @FREE
         CLR  *SP
         CLR  R0
         CLR  @LOMINI           ;See if MMM responds
         C    @LOMINI,R0
         JNE  OUT
         SETO R0
         SETO @LOMINI
         C    @LOMINI,R0
```

```
          JNE  OUT
          INC  *SP       ;MMM available
          LI   R0,LOMINI ;Init free pointer
          MOV  R0,@FREE
OUT       B    *LINK


          .RELFUNC MININEW,2

          MOV  *SP+,R0   ;Size of argument
          MOV  *SP+,R1   ;Addr to pointer
          CLR  *SP

          MOV  @FREE,R2  ;Free pointer
          JEQ  OUT       ;No room?

          LI   R3,HIMINI ;Space remaining
          S    R2,R3
          C    R3,R0     ;Enough?
          JL   OUT

          INC  *SP
          MOV  R2,*R1    ;Set pointer
          A    R0,R2     ;Advance free
          CI   R2,HIMINI ;End of MMM?
          JL   MORE
          CLR  R2        ;If end of memeory then set free to nil
MORE      MOV  R2,@FREE
OUT       B    *LINK


          .RELFUNC INMINI,1

          MOV  *SP+,R0   ;Copy pointer address
          MOV  *R0,R0
          CLR  *SP
          CI   R0,LOMINI ;Within MMM?
          JL   OUT
          CI   R0,HIMINI
          JHE  OUT
          INC  *SP
OUT       B    *LINK

          .END
```

Disk 97. Contents of file MINITEST.TEXT

```
program minitest;

(* Get something big to allocate *)
type
  large = array[1..1000] of integer;

var
  p:array[1..5] of ^large;
  i:integer;

(* Declaration of Mini Memory support functions *)
function initmini:boolean;                              external;
function mininew(var pointer; size:integer):boolean;  external;
function inmini(var pointer):boolean;                  external;

begin
  writeln(varavail('')*2,' bytes available');

  if initmini then
    writeln('Mini memory used')
  else
    writeln('Can''t find Mini memory');

  for i := 1 to 5 do
  begin
    if mininew(p[i],sizeof(large)) then
      writeln('p[',i,'] allocated in Mini Memory')
    else
    begin
      writeln('p[',i,'] not allocated in Mini Memory');
      new(p[i]);
    end;
  end;

  writeln(varavail('')*2,' bytes available');
end.
```

Disk 97. Contents of file MINI.INFO.TEXT

MINI MEMORY

The routines for Mini Memory are rather simple, but might still be useful. The use of the functions is described in the source code MINIMEMORY.TEXT.

There is also an example in the file MINITEST.CODE. You may run that program with X(ecute, and see the difference when the module is present and when it isn't. The source is called MINITEST.TEXT.

When linking, you might give either MINMEMORY or *SYSTEM.LIBRARY as "Lib file".

Note that memory space in Mini Memory is not released when minidispose is called. To reuse memory in the module, initmini must be called. But that means that the data presently stored in the module disappears (sooner or later).

On the other hand, global buffers and the like are very suitable for allocation in Mini Memory at the start of the program. Then they might stay there during the entire execution of the program.

The Mini Memory routines might of course be changed to allow storage in modules with 8K bytes of RAM, like Maximem, a Super Cart or the GRAM Kracker. Nowadays, with several RAM disks available, the need for this routine might be less obvious. But there is the advantage that this program allows the easiest storage and retrieval of data. Instead of simulating a disk, which requires the use of file access, data is referenced with ordinary pointers.

Disk 97. Contents of file SYS.MAP.TEXT

The p-Code card

The p-Code card has a total memory capacity of 60 kilobytes. This memory consists of 12 K ROM and 48 K GROM.

ROM:
The ROM memory is located at 4000-5FFF. 5000-5FFF is paged in two pages, but the lower four kilobytes are always the same. The paging is done with CRU bit 1F80. Resetting the bit to zero gives access to the normal page. Setting the bit to one switches in the extra page.

GROM:
The GROM chips contain the files that are located in unit #14: (OS:). They also contain various data and assembly code, which is loaded into RAM when the card is initialized.

The p-Code GROM is accessed just like the console and module GROM chips, but at different addresses.

The following are used:

PGRMRD 5BFC Read data.
PGRMRA 5BFE Read address.
PGRMWD 5FFC Write data. (Not used. No GRAM here.)
PGRMWA 5FFE Write address.

When running, the p-system uses several different workspaces. The main one used, however, is located at 8380. Others used are 83A0, 83C0, 83E0 (GPLWS), 2896 and 28B6. Maybe others too, for special purposes.

R8 in the main workspace (8380) is the p-Code instruction pointer.
R9 is used as a frame pointer for activation records allocated on the stack.
R10 is the stack pointer.
R11 is the return link.
R12 contains the address of the PME instruction fetch routine.
R13 contains the Read Data address of the currently executing code segment. Could be PGRMRD, VDPRD or 0 (for CPU RAM).
R14 Global data frame pointer for current segment.
R15 is a flag used to remember where the p-Code currently interpreted is located.
=0: CPU RAM
<0: VDP RAM
>0: GROM

Map of 8K RAM under the p-system.
All addresses in hex.

```
================================================================================

2000-277F  80 column screen.
           Temporarily used for other purposes during power-up.
2780       INTMEM   Interpreters memory pointer in VDP RAM.
           Initialized to 0DF8.
2782       TOPMEM   Top memory pointer in VDP RAM.
           Initialized to 3EDF.
2784-27B7  Sound data.
***
27D8-27E6  Sound data.
***
2808       Last sprite coincidence counter.
280A-280C  Bit maps for moving sprites.
***
2810-281C  Copies of VDP register 1-7 (standard mode).
           Used when initializing the p-Code card.
281E       VDP status copy.
***
2886       Cursor addressing flag.
***
288A       Keyboard buffer pointer.
288C       Keyboard layout area pointer.
288E       Screen blanking timer.
2890       Stop flag.           False=0, true=FFFF
2892       Alpha lock flag.
2894       Flush flag
2896-28B4  Subsidiary workspace.
 28A2      Key buffer mode flag. (FF40=empty, FF00=something, FF80=full)
 28A4      Key buffer store pointer  [0..31]
 28A6      Key buffer fetch pointer  [0..31]
 28AA      VDPWA (at least sometimes).
28B6-28D4  "Interrupt" workspace.
***
2960       PAB pointer
***
2964-2982  Workspace
2984       Index into interrupt address table.
2986-29A2  CRU addresses of cards with interrupt routines. End marker=0000.
 29AA      Block number.
 29AC      Byte count.
 29AE      Buffer address.
 29B0      Drive number.
 29B2      Control parameter.
***
29A8-29B8  Data passed to I/O routines.
***
29DC-2A1C  PAB pointer table for units.
***
```

```
2A24-2A32  Register save area R0-R7.
2A34       Save area.
***
2AA0-2AA4  KSCAN data for 83C6-83CA
***
2AAA-2AAC  Vector for "interrupt".                          (28B6  32DC)
2AAE-2AB0  Vector for keyboard scanning and buffer storage.  (2896  41BA)

           (* Various PME data located at 2AB2-2AF6.
              Used for task switching and similar. *)

2AB2       Current SIB pointer.                      (CURSIB)
2AB4       Current Environment Record Pointer        (E_Rec_P)
2AB6       Pointer to TIB at head of ready queue.    (READYQ)
2AB8       Current Environment Vector Pointer        (E_Vec_P).
2ABA       Current TIB pointer.                      (CURTSK)
             See Internal Architecture Guide for descriptions of TIB, SIB and
             environment vectors and records.
***
2ABE       Current segment constant pool pointer (absolute).
2AC0       Current segment constant pool pointer (relative).
2AC3       Current procedure number (byte). (High byte not used?)
***
2ACC       P-Code IPC save area. Used by the PME.
***
2ADC       Return address (R11) save area.
***
2AE2       Segment base. IPC is segment relative.
2AE4       Routine dictionary pointer.
***
2AE8       PME filp indicator.
***
2AF2       Save area.
***
2AF8-2B12  'Segment ________ not found:'
2B14-2B31  'Put volume _______ in unit #__'
2B33-2B4F  'Press spacebar to continue...'
***
2BA4-2BA6  System clock (low,high).
***
2BB8       24 (Screen height).
2BBA       40 (Screen width).
2BBC       Code for arrow down.
2BBD       Code for arrow up.
2BBE       Code for arrow right.
2BBF       Code for arrow left.
2BC0       Code for FLUSH.
2BC1       Code for etx.
2BC2       Code for STOP.
2BC3       Code for BREAK.
```

```
2BC4        Code for editor unknown character?
2BC5        Code for backspace.
2BC6        Code for escape.
2BC7        Code for line delete.
2BC8        Code for etx.
2BC9        ?
2BCA        Code for alpha lock.
***
2BD0-2BEA   TIB for main task.
***
2BFC-2C1C   Corresponding drive number (byte) for blocked units.
***
2C20-2C60   Table with pointers for different units.
 2C20       0000     #0  (Reserved)
 2C22       2C68     #1  CONSOLE
 2C24       2C68     #2  SYSTERM
 2C26       0000     #3  (Graphic)  Reserved for Terak microcomputers.
 2C28       2C70     #4  Diskette name
 2C2A       2C70     #5  Diskette name
 2C2C       2C80     #6  PRINTER
 2C2E       2C80     #7  REMIN
 2C30       2C80     #8  REMOUT
 2C32       2C70     #9  Diskette name
 2C34       0000     #10 (Diskette name)
 2C36       0000     #11 (Diskette name)
 2C38       0000     #12 (Diskette name)
 2C3A       2C70     #13 Assigned to unit loaded from tape.
 2C3C       2C70     #14 OS
 2C3E       2C70     #15
   ...and so on...
 2C5C       2C70     #30
 2C5E       2C88     #31 TAPE
 2C60       2C78     #32 TP
***
            (* Addresses in secondary ROM page *)
2C68-2C6E   538E 547E 5270 5360  (Computer console)
2C70-2C76   5796 5740 568C 5652  (Blocked devices)
2C78-2C7E   0000 59FA 59CA 0000  (Thermal printer)
2C80-2C86   5ABE 5AD8 5AB4 0000  (RS232 card)
2C88-2C8E   5B68 5B22 5B10 0000  (Tape)
2C90-2C93   Sound data.
***
2CAC        DSR validation indicator  (AA).
***
2CB0-2CBC   Copies of VDP register 1-7.
2CBE        Pointer to 80 column screen memory.
2CC0        Current line.
2CC2        Current column.
2CC4        Current 80 col address.
2CC6        Screen width.
2CC8        Screen height.
```

```
2CCA        Current screen window.
2CCC        Number of bytes on 23 lines.
***
2CD0        Autorepeat counter
2CD2        Keyscan limitation counter. (Key buffering isn't done on every
            interrupt.)
2CD4        Keyscan limitation value.
***
2CFA-       Some table.
***
2D20-       Some table.
***
2D92-2DB0   Bit masks for general use.
2DB2-2FB0   Address table for p-Code interpreter. Opcode is a word index.
***
2FFE-3016   Code which resets the computer and branches to the normal OS.
3018-3024   Code calling something in secondary page on card.
3026-308C   Table with pointers to some of the keywords in the next table.
            Each pointer is followed by a value.
 3026       AND      2
 302A       BEGIN    1
 302E       CASE     2
 3032       DEFINITI 4
 3036       ELSE     3
 303A       FOR      4
 303E       GOTO     1
 3042       @@@@@@@@ 1
 3046       IF       4
 304A       @@@@@@@@ 1
 304E       @@@@@@@@ 1
 3052       LABEL    1
 3056       MOD      1
 305A       NOT      1
 305E       OF       2
 3062       PACKED   4
 3066       @@@@@@@@ 1
 306A       RECORD   2
 306E       SEPARATE 3
 3072       THEN     3
 3076       UNIT     3
 307A       VAR      1
 307E       WHILE    2
 3082       @@@@@@@@ 1
 3086       @@@@@@@@ 1
 308A       @@@@@@@@ 1
            Data for IDSEARCH.
308E        '@@@@@@@@' 000F  Unknown identifier code
3098        'AND     ' 2702
30A2        'ARRAY   ' 2C0F
30AC        'BEGIN   ' 130F
```

```
30B6        'CASE    ' 150F
30C0        'CONST   ' 1C0F
30CA        'DEFINITI' 000F
30D4        'DIV     ' 2703
30DE        'DO      ' 060F
30E8        'DOWNTO  ' 080F
30F2        'ELSE    ' 0D0F
30FC        'END     ' 090F
3106        'EXTERNAL' 350F
3110        'FOR     ' 180F
311A        'FILE    ' 2E0F
3124        'FORWARD ' 220F
312E        'FUNCTION' 200F
3138        'GOTO    ' 1A0F
3142        'IF      ' 140F
314C        'IMPLEMEN' 340F
3156        'IN      ' 290E
3160        'INTERFAC' 330F
316A        'LABEL   ' 1B0F
3174        'MOD     ' 2704
317E        'NOT     ' 260F
3188        'OF      ' 0B0F
3192        'OR      ' 2807
319C        'PACKED  ' 2B0F
31A6        'PROCEDUR' 1F0F
31B0        'PROCESS ' 380F
31BA        'PROGRAM ' 210F
31C4        'RECORD  ' 2D0F
31CE        'REPEAT  ' 160F
31D8        'SEPARATE' 360F
31E2        'SET     ' 2A0F
31EC        'SEGMENT ' 210F
31F6        'THEN    ' 0C0F
3200        'TO      ' 070F
320A        'TYPE    ' 1D0F
3214        'UNIT    ' 320F
321E        'UNTIL   ' 0A0F
3228        'USES    ' 310F
3232        'VAR     ' 1E0F
323C        'WHILE   ' 170F
3246        'WITH    ' 190F
3250-3296   Code for PME instruction decode. Used when p-Code is in mapped
            memory, i.e. VDP RAM or GROM.
3298-32DA   Code for PME kernel. Used when interpreted code is in CPU RAM.
            The code portions above are transferred to RAM PAD (8300-8344)
            when executing for speed reasons.
32DC-337C   Code for simulated interrupts.
            The p-system always runs with interrupts disabled. Interrupt
            service is provided by reading the interrupt request as an ordinary
            I/O bit. If it's active, this code is executed.
            The I/O bit is tested after every taken (p-Code) JUMP instruction,
```

```
           and also during various output tasks.
           Both vertical blanking and external interrupts are serviced.
337E-34AA  Code for some kind of I/O.
34AC-34C2  Code...
34C4-34DA  Code...
34DC-34EC  Code...
34EE-34FE  Code...
3500-3508  Pops two values, pushes zero and fetches next p-Code.
350A-350E  Pops one value, then next p-Code.
3510-352A  Code... Turns on normal page and some more things.
352C-353A  Code which switches in alternate p-Code ROM page.
353C-354A  Code for normal page. Sometimes entered @3540.
354C-355C  Code...
355E-3566  Code giving normal page and then run-time error (unknown instr.).
3568-357E  Code erasing the 80 column screen memory.
3580-38FC  Kernel global data area.
  3600-3607  Prefix volume name (string[7]).
  3608-360F  Root volume name   (string[7]).
  3610       System date.
             packed record
               month:1..12;   (* 4 bits *)
               date :1..31;   (* 5 bits *)
               year :0..99;   (* 7 bits *)
             end;
  ***
  3614-3616  Heap_Info.lock      :semaphore;
  3618       Heap_Info.HeapTop   :MemPtr;
  361A       Heap_Info.TopMark   :MemPtr;
  361C-361E  Task_Info.Task_Done :semaphore;
  3620-3622  Task_Info.Lock      :semaphore;
  3624       Task_Info.N_Tasks   :integer;
  3626           1 (decimal constants)
  3628          10
  362A         100
  362C        1000
  362E       10000
  ***
  3696-3820  Table with name and type of units #0..#32.
             record
               volume_id   :string[7];
               is_blocked  :boolean;
               no_of_blocks:integer;    (* maxint if not blocked *)
             end;
  3822-3839  File name for the assembler.      Type is string[23].
  383A-3851  File name for the compiler.
  3852-3869  File name for the editor.
  386A-3881  File name for the filer.
  3882-3899  File name for the linker.
  ***
  38CA-38E1  File name for the library text file (usually '*USERLIB.TEXT').
```

```
  ***
  38FA       '99' when running.
             'GO' when stopped because e.g. stack overflow.
             'NO' when stopped by H(alt.
            The p-system doesn't start on reset if this location contains 'NO'.
  38FC       '/4' when running.
***
================================================================================
```

Note: All information provided here has been gathered by inspection of the p-System in operation and by disassembly of the p-Code interpreter. No official information from TI or SofTech has been available. Correctness in the mapping of the p-System is not guaranteed.

A-DATA 87
Anders Persson, Lund, Sweden.

Disk 97. Contents of file WELCOME.TEXT

```
program welcome;
  (* Used for displaying a welcome message on the sig disk *)

var
  ch      :char;


begin (* welcome *)
  page(output);
  gotoxy(10,0);
  writeln('HELLO');
  writeln;
  writeln('You''ve now received some programs and');
  writeln('general information about the UCSD');
  writeln('p-system for the TI 99/4A.');
  writeln;
  writeln('Some programs here are ready to run, but');
  writeln('most of them are intended to help you');
  writeln('when you write your own programs.');
  writeln;
  writeln('These routines are usually units, which');
  writeln('can be referenced from your programs.');
  writeln('There are examples of this too.');
  writeln;
  writeln('The next thing to do now is to start the');
  writeln('editor and load the file ''diskinfo''.');
  writeln('It describes the contents of the other');
  writeln('files on the disk.');
  gotoxy(0,23);
  write('Press...');
  read(keyboard,ch);
end. (* welcome *)
```

Disk 98. c99 Windows — Prog and Docs

Version:
Author: Tom Bentley
Requires: c99
Language: c99
Updated: 3/14/88

This disk contains a program called "The Window" which allows for reading of documents in a windowed environment. It contains a second demo and documentation.

```
dskdir. v2.0. 12-dec-96
Disk name               = C99WINDOWS
Sectors total           = 360
Sectors used            = 290
Sectors available       = 68
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 INTRO         9 DIS/VAR 80   >022 008
002 >003 WDEMO        33 PROGRAM      >02a 032
003 >00b WDEMO2        1 DIS/VAR 80   No data sectors in file
004 >004 WDEMP        16 PROGRAM      >04a 015
005 >005 WINDOW       33 PROGRAM      >059 032
006 >006 WINDOWS1DC   70 DIS/VAR 80   >079 069
007 >007 WINDOWS2DC   75 DIS/VAR 80   >0be 074
008 >008 WINDOWSDOC    2 DIS/VAR 80   >108 001
009 >009 WINDOX       33 PROGRAM      >109 032
010 >00a WINDOY       18 PROGRAM      >129 017
```

Disk 98. Contents of file INTRO

INTRODUCTION

This submission is called c99 Windows. C99 Windows is a 'c' library designed to allow programmers to use windows within their programs. It is designed to be simple to use for beginner programmers, while also having the ability to do many complex operations required by more experienced programmers. Since the library by itself does no justice to what can be accomplished I have written two programs to demonstrate it's power.

Program 1 — WDEMO

Run WDEMO from any available program loader to see some of the features that are available in c99 Windows. This is an example of a simple program. The source for this program is in WDEMOC.

Program 2 — WINDOW

Run WINDOW from any available program loader. This program is a version of the original 'WINDOW' program except that it has been converted to use c99 Windows. This program gives a good example of how to use command lines, popup windows and other sophisticated options. The source is contained in WINDOWC. Note that when you are prompted on a command line or popup window use the arrow keys to change the selection, enter to select it and **FCTN 9 (BACK)** to backout.

Included in this submission are all of the source files plus a 40 page manual on how to build and use c99 Windows.

Author: Tom 'c' Bentley
For: Ottawa TI Users Group
Software Contest
1988

Disk 98. Contents of file WINDOWS1DC

Windows For C99

Edition 1.0 : 87/09/01

Software and Documentation written by Tom 'c' Bentley

The c99 Windows library allows the 'c' programmer to use windows inside of his/her programs. To use c99 Windows the programmer simply opens windows and puts or gets information from them. Other features such as a message line, command bar and pop-up menus allow the programmer to make his programs very user friendly and pleasing to the eye.

To make the use of c99 Windows simple most of the input and output routines were modeled after existing 'c' functions, for example puts would be replaced w_puts to print strings to a window. Using this approach will allow you to change existing programs to utilize c99 Windows very quickly with a minimal amount of code change. Another nice feature of c99 Windows is its' 80 column virtual display buffer, which will allow you to display up to 80 columns of information in the window, of course you will still have to flip through the virtual buffer using one of the supplied functions to see all of the information.

Since any type of windowing system takes a considerable amount of memory to run I have focused on creating a windowing library that has minimal memory usage, functionality and speed. Therefore there is very little to no error checking of parameters when calling any function so be careful.

There is no charge for this library, all I ask is if you develop any application using this library please send me a copy. If you have any problems or suggestions please send correspondence to:

Tom Bentley
5662 Gordon St.
Box 346
Osgoode, Ontario
Canada K0A 2W0

FIRST THINGS FIRST

The first thing you do is make a backup of the Windows library. Second build the object modules, to do this do the following:

1) Compile and assemble WINDOWSC to be WINDOWS.
2) Compile and assemble WCMDC to be WCMD.
3) Compile and assemble WCMDBARC to be WCMDBAR.
4) Compile and assemble WPOPUPC to be WPOPUP.
5) Compile and assemble WMSGC to be WMSG.
6) Compile and assemble WGETSC to be WGETS.
7) Compile and assemble WINDOWSG to be WINDOWSGO.

8) Compile and assemble WINDOWS1G to be WINDOWS1GO.
9) Compile and assemble WINDOWS2G to be WINDOWS2GO.
10) Compile and assemble WINDOWS3G to be WINDOWS3GO.

Compiler requirements: c99 version 2.1

The object file WINDOWS contain the kernel routines for c99 Windows and is always required when using c99 Windows. The other files are required when using other features of c99 Windows; WCMD is used to resize the window or scroll the virtual buffer, WCMDBAR is used when the command bar option is required, WPOPUP is used when you wish to use popup option windows, WMSG is required if you want to use the message line and WGETS is required if you want to get input from a window.

How to load the Libraries:

Using option 3 Load and Run or my program 'CLOAD' load the following modules in the specified order:

DSK1.C99PFI
DSK1.<your object module(s)>
DSK1.GRF1 (required)
DSK1.WINDOWS (required)
DSK1.WCMD (if required)
DSK1.WCMDBAR (if required)
DSK1.WPOPUP (if required)
DSK1.WMSG (if required)
DSK1.WGETS (if required)
DSK1.CSUP (required)
DSK1.C99PFF
DSK1.WINDOWSGO (contains globals must be placed here!)
or WINDOWS1GO/WINDOWS2GO/WINDOWS3GO (broken up WINDOWSGO)

Things for the future:

If this library is received well by c99 users I will continue adding and improving it, so please send me your comments.

Table of Functions and Variables

LIBRARY FUNCTIONS

buf_col variable

Contains the starting column to display in window.

Calling Sequence:

```
extern int buf_col;
```

Description:

This variable is used to control which virtual buffer column is the first column to be displayed within the current window.

Example:

```
/* Program for buf_col example */

#include "dsk1.windowsh"
```

```
extern int buf_col;

main()
  {
    w_open("Windows");
    w_puts("This is an example of how to use buf_col");
    buf_col = 5; /* start display at 'is' */
    w_flush();   /* flush it to force redisplay */
    w_close();
  }
```

fillb function

Fill byte array with specified value.

Calling Sequence:

```
#include "dsk1.windowsh"

char *string, c;

int   n;

fillb(string,c,n);
```

where

string	is the address to fill.
c	is the fill character.
n	is the number of c to fill into string.

Description:

Use fillb to fill a character value into some string array.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for fillb function */

#include "dsk1.windowsh"

main()
 {
  char string[81];

  fillb(string,'\0',81);  /* fill string with nulls */
   .
   .
 }
```

strlen function

Determine the length of a character string.

Calling Sequence:

```
#include "dsk1.windowsh"

char *string;

int   siz;

siz = strlen(string);
```

where

string	is the pointer to the string to inspect.
siz	receives the result of the count.

Description:

Use strlen to determine the length of a null terminated string.

Returns:

Returns the length of the string, in bytes.

Object File Location:

WINDOWS

Example:

```
/* Program of strlen function */

#include "dsk1.windowsh"

main()
  {
    char *string;
    int   len;

    string = "This is a string";
    len = strlen(string);
     .
     .
  }
```

use_vbuf variable

Flag specifying whether to use the virtual buffer.

Calling Sequence:

```
extern int use_vbuf;
```

Description:

This variable is used to control whether information is written into the 80 column virtual buffer. Set it to 1 to use the virtual buffer, otherwise set it to 0.

Example:

```
/* Program for use_vbuf variable */

#include "dsk1.windowsh"

extern int use_vbuf;

main()
  {
    w_open("Windows");
    use_vbuf = 0; /* don't use virtual buffer */
    w_puts("Not using virtual buffer");
    w_close();
  }
```

w_box function

Draw or erase a box on the screen.

Calling Sequence:

```
#include "dsk1.windowsh"

int top_row, top_left_col, bottom_row;

int bottom_right_col, clear;

w_box(top_row,top_left_col,bottom_row,bottom_right_col,clear);
```

where

top_row	is the top row of the box.
top_left_col	is the top left column of the box.
bottom_row	is the bottom row of the box.
bottom_right_col	is the bottom right column of the box.
clear	specifies whether the box is drawn or erased.

Description:

Use w_box to draw or erase a box on the screen. A box will be drawn if clear is false otherwise it will be erased from the screen.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for w_box function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_box(2,1,23,40,0); /* draw a box */
    w_box(2,1,23,40,1); /* erase the box just drawn */
     .
     .
  }
```

w_clear function

Clears the current window display

Calling Sequence:

```
#include "dsk1.windowsh"

w_clear();
```

Description:

Use w_clear if you wish to clear the current window.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_clear function */

#include "windowsh"

main()
  {
    w_init();
    w_open("Windows");
    w_puts("Lines 1\nLine 2\n);
    w_clear();
  }
```

w_close function

Closes the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

w_close();
```

Description:

Use w_close to close the current window. When the window is closed it also clears it from the screen.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_close function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Windows");
     .
     .
    w_close();
  }
```

w_clrbuf function

Initializes the virtual buffer to nulls.

Calling Sequence:

```
#include "dsk1.windowsh"

w_clrbuf();
```

Description:

Use w_clrbuf to initialize the virtual buffer to nulls. The virtual buffer consists of 20 rows by 80 columns.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_clrbuf function */

#include "dsk1.windowsh"

main()
  {
    w_init();
     .
     .
    w_clrbuf();
     .
     .
  }
```

w_cmd function

Resize window or flip virtual buffer in window.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_cmd();

char c;

c = w_cmd(c);
```

where

c is the entered command, or the command to execute.

Description:

Use w_cmd if you wish to resize the current window or scan through the virtual buffer. If you pass a non ULL character value to w_cmd then this value will be used to perform the action instead of getting input rom the keyboard. Control S,D,E or X will resize the window using the specified direction. Function 5 next window) will scan through the virtual buffer and redisplay it to the window.

Returns:

Returns the specified command as a character.

Object File Location:

WCMD

Example:

```
/* Program for the w_cmd function */

#include "dsk1.windowsh"
.br
extern w_cmd();

main()
  {
    w_init();
    w_open("Window");
    w_puts("This is an example of how to use w_cmd");
    while(1)
        w_cmd();
  }
```

w_cmdbar function

Display command bar with options.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_cmdbar();

char *options;

int   n;

n = w_cmdbar(options)
```

where

	options	is the string containing command options.
	n	is the option selected.

Description:

Use w_cmdbar to display a command bar on line 1 with the specified options. The option string is a stream of characters with each option separated with a newline (\n). There may be up to 20 options on the command bar but the total bar length may not exceed 40 characters, the newlines are not counted in the length. The first option is numbered option 1 and will be the first option on the left of the command bar (it will be inverted). To select a specific option use the arrow keys (function S and D) until you are on the option you wish to choose and press enter to choose it. If you do not wish to select anything then press **FCTN 9 (BACK).**

Returns:

The option you selected from 1 to the maximum options you have specified or 0 if **FCTN BACK** was entered. If w_raw is set to 1 then any character pressed will be returned, see w_raw for more detail.

Object File Location:

WCMDBAR

Example:

```
/* Program for the w_cmdbar function */

#include "dsk1.windowh"
.br
extern w_cmdbar();

main()
  {
    int opt;

    w_init();
    opt = w_cmdbar(" option 1 \n option 2 \n");
     .
     .
  }
```

w_drum variable

Contains pointers to access virtual buffer.

Calling Sequence:

```
extern int w_drum[];
```

Description:

This variable contains byte pointers to the virtual buffer lines. Using these pointers you may access or alter the contents of the virtual buffer at any time. This variable consists of 20 buffer pointers, each containing a pointer to its respective buffer line.

Example:

```
#include "dsk1.windowsh"

extern int w_drum[];

main()
  {
    char *lin;  /* used for cast */
    char *lin2;

    w_open("Windows");
    w_puts("Line 1.");
    lin = w_drum[0]; /* pointer to 1st line */
    lin2 = w_drum[1]; /* pointer to 2nd line */
    strcpy(lin2,lin); /* copy line 1 to line 2 */
    w_close();
  }
```

w_flush function

Flushes the virtual buffer to the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

w_flush();
```

Description:

Use w_flush to flush the 80 column virtual buffer to the current window.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_flush function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Window");
    w_puts("Information\n");
    w_open("New Window);
    w_flush(); /* flush previous windows' buffer to this one */
  }
```

Disk 98. Contents of file WINDOWS2DC

w_getchar function

Get a character from the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_getchar();

int c;

c = w_getchar();
```

where
c is equal to the next character.

Description:

Use w_getchar to get a character from the current window.

Returns:

Returns the entered character.

Object File Location:

WGETS

Example:

```
/* Program for the w_getchar function */

#include "dsk1.windowsh"

extern w_getchar();

main()
  {
    int c;

    w_init();
    w_open("Windows");
    w_puts("Press newline to continue");
    while((c = w_getchar()) != '\n');
     .
     .
  }
```

w_gets function

Gets a string from the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_gets();

char *string;

w_gets(string)
```

where

string is a pointer to a user-provided character array.

Description:

Use w_gets to get a string of information from the current window. If you reach the end of the window during input then it will be terminated and the string will end here. You may use this fact to limit the size of input. Inline editing is supported (function S). To end input enter a newline.

Returns:

Nothing

Object File Location:

WGETS

Example:

```
/* Program for the w_gets function */

#include "dsk1.windowsh"

extern w_gets();

main()
  {
    char string[81];

    w_init();
    w_open("Window");
    w_puts("Enter Your Name: ");
    w_gets(string);
     .
     .
  }
```

w_init function

Initializes the windowing library.

Calling Sequence:

```
#include "dsk1.windowsh"

w_init();
```

Description:

Use this functions to initialize the windows library. This must be the first call before any other window function is used. This call will set all of the initial window positions, clear the virtual buffer and set up the character definitions required by c99 Windows.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program to initialize Windows */

#include "dsk1.windowsh"

main()
  {
    w_init();
  }
```

w_inv function

Inverts the character in string.

Calling Sequence:

```
#include "dsk1.windowsh"

char *string;

w_inv(string)
```

where
string is a pointer to a null terminated string.

Description:

Use w_inv to turn on the high order bit for each character in the string.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_inv function */

#include "dsk1.windowsh"

main()
  {
    char *string;

    w_init();
    w_open("Windows");
    string = "Inverted string";
    w_inv(string);
    w_puts(string);
     .
     .
  }
```

w_locate function

Locates the current cursor row and column in window.

Calling Sequence:

```
#include "dsk1.windowsh"

int row, col;

w_locate(row,col)
```

where

row	is the row in the window to place the cursor.
col	is the column in the window to place the cursor.

Description:

Use w_locate to locate the cursor to any position within the current window. The row range is from row 1 to row 20 and the col range is from col 1 to col 80. If the specified row range is greater than the current windows row range then the current windows row range is used instead.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_locate function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Windows");
    w_locate(10,5);
    w_puts("I'm on row 10 column 5.");
     .
     .
  }
```

w_msg function

Displays a message to the screen.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_msg();

char *message;

int   invert;

w_msg(message,invert)
```

where

	message	is a pointer to a null terminated string.
	invert	specified whether to invert message or not.

Description:

Use w_msg to display a message on line 24. If invert is equal to one then the message will be inverted and then displayed otherwise if invert is equal to zero then the message will not be inverted before being displayed. Ensure that the message is not greater than 40 characters.

Returns:

Nothing

Object File Location:

WMSG

Example:

```
/* Program for the w_msg function */

#include "dsk1.windowsh"

extern w_msg();

main()
  {
    w_init();
    w_msg("Ready to open a window", 1);
  }
```

w_noinv function

Changes an inverted character to its' normal state.

Calling Sequence:

```
#include "dsk1.windowsh"

char *string;

w_noinv(string);
```

where
string is a pointer to a null terminated string.

Description:

Use w_noinv to turn off the high order inversion bit for each character in the string.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_noinv function */

#include "dsk1.windowsh"

main()
  {
    char *string;

    w_init();
    w_open("Windows");
    string = "Inverted string\n";
    w_inv(string);
    w_puts(string);
    w_noinv(string);
    w_puts(string);
     .
     .
  }
```

w_open function

Opens up a new window.

Calling Sequence:

```
#include "dsk1.windowsh"

char *title;

int  ok;

ok = w_open(title);
```

where

title	is a pointer to null terminated string, or NULL.
ok	is a 1 if window opened ok else 0.

Description:

Use the w_open function to allocate and open a window. Whenever windows are opened the position and size of the window will be starting on row 2 to row 23 and on column 1 to column 40. These assumptions may be changed by calling w_position with your new values. If title is not NULL then it will be displayed when the window is displayed.

Returns:

The w_open returns a 1 is a window was opened, otherwise a 0 is returned signifying an error.

Object File Location:

WINDOWS

Example:

```
/* Program to open a new Window */

#include "dsk1.windowsh"

main()
  {
    w_init();
    if(!w_open("This is a Window"))
        puts("Error Unable to open a Window.\n");
  }
```

w_popup function

Display popup window with options.

Calling Sequence:

```
#include "dsk1.windowsh"

extern w_popup();

char *title, *options;

int   n;

n = w_popup(title,options);
```

where

title	is pointer to a null terminated string.
options	is pointer to a string containing options.
n	is the selected option.

Description:

Use w_popup to display a command window with the specified options. The options string is a stream of characters with each option separated with a newline (\n). There may be up to 20 options in the command window and the total option length may not exceed 38 characters, the newlines are not counted in the length. The first option is numbered option 1 and will be the first command on the top line of the window (it will be inverted). To select a specific option use the arrow keys (function E and X) until you are on the option you wish to choose and press enter to choose it. If you do not wish to select anything then press function 9 (BACK). Use w_position to position the popup window in a position other than the default, only the top row and left column is looked at as w_popup will decide what the bottom row and right column should be depending on the number of options and the maximum width of those options.

Returns:

The option you selected from 1 to the maximum options you have specified or 0 if function BACK was entered. If w_raw is set to 1 then any character pressed will be returned, see w_raw for more details.

Object File Location:

WPOPUP

Example:

```
/* Program for the w_popup function */

#include "dsk1.windowh"

extern w_popup();

main()
  {
    int opt;

    w_init();
    opt = w_popup("OPTIONS"," option 1 \n option 2 \n");
     .
     .
  }
```

w_position function

Positions the next window to be opened.

Calling Sequence:

```
#include "dsk1.windowsh"

int top_row,bottom_row,left_column,right_column;

w_position(top_row,bottom_row,left_column,right_column);
```

where

top_row	is the top row of the window from 2 to 22.
bottom_row	is the bottom of the window from 3 to 23.
left_column	is the left of the window from 1 to 39.
right_column	is the right of the window from 2 to 40.

Description:

Use w_position to reposition the next window to be opened. The top_row may not be less than 2 and greater than 22, bottom_row may not be less than 3 and greater than 23, left_column may not be less than 1 and greater than 39 and right_column may not be less than 2 and greater than 40.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_position function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_position(10,15,10,15);
    w_open("A repositioned window");
  }
```

w_putchar function

Puts a character to the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

char c, ch;

ch = w_putchar(c)
```

where

c	is the character to put.
ch	is the character just put to the window.

Description:

Use w_putchar to put a character to the current open window. The following special characters do the following; \n (newline) will cause the cursor to move down one line and to start in column one, \r (return) will move the cursor to column one, \f (formfeed) will clear the screen and position the cursor at the top left hand corner of the window and \t (tab) will move the cursor to the right five spaces. This function will display any other character to the window if the current column is less than the window width but will continue to put characters into the virtual buffer until the 80th column is reached then it will force a \n (newline).

Returns:

Character that is being put.

Object File Location:

WINDOWS

Example:

```
/* Program for the w_putchar function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Windows");
    w_puts("Information in window");
    w_putchar('\f');  /* clear the window */
     .
     .
  }
```

w_puts function

Puts a string to the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

char *string;

w_puts(string);
```

where
string is a pointer to a null terminated string.

Description:

Use w_puts to put a string to the current window. This function will call w_putchar for each character in the string until it reaches a NULL (ascii 0) in the string.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_puts function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Windows");
    w_puts("This is one line\nThis is another line.");
     .
     .
  }
```

w_raw variable

Specifies whether raw characters will be passed through w_cmdbar or w_popup.

Calling Sequence:

```
extern int w_raw;
```

Description:

If w_raw is set to 1 then any character other than the valid movement or selection ones will be returned back from w_cmdbar or w_popup as negative values, setting w_raw back to 0 will reset w_cmdbar and w_popup to normal operations.

Example.

```
/* Example of w_raw */

#include "dsk1.windowsh"

extern int w_raw;

main()
  {
    int opt;

    w_raw = 1;
    while((opt = w_cmdbar(" opt1 \n opt2 \n")))
        if(opt < 0)
          {
            /* then do special case */
            switch(-opt)
              case ...
          }
```

```
        else
          {
            /* do normal options */
            switch(opt)
              case ...
          }
  }
```

w_rmargin variable

Determines the rightmost margin within the virtual buffer.

Calling Sequence:

```
extern int w_rmargin;
```

Description:

This variable determines what the rightmost margin within then 80 column virtual buffer. The default for this variable is 80.

Example.

```
/* Example of w_rmargin */

#include "dsk1.windowsh"

extern int w_rmargin;

main()
  {
    w_rmargin = 20; /* margin will wrap at 20th char. */
    .
    .
    .
  }
```

w_scroll function

Scrolls the current window up by one line.

Calling Sequence:

```
#include "dsk1.windowsh"

w_scroll();
```

Description:

Use w_scroll if you wish to scroll the information in the current window up by one line.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_scroll function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("Windows");
    w_puts("Line 1\nLine2\n");
    w_scroll();
  }
```

w_title function

Displays a new title to the current window.

Calling Sequence:

```
#include "dsk1.windowsh"

char *title;

char  just_type;

w_title(title,just_type);
```

where

	title	is a pointer to a string.
	just_type	is a character to specify justification type.

Description:

Use w_title to display a new title to the current opened window. The just_type is one of; 'C' center title, 'L' left justify title and 'R' right justify title.

Returns:

Nothing

Object File Location:

WINDOWS

Example:

```
/* Program for the w_title function */

#include "dsk1.windowsh"

main()
  {
    w_init();
    w_open("old title");
    w_title("new title",'R');
     .
     .
  }
```

window variable

Contains the current open window.

Calling Sequence:

```
extern int window;
```

Description:

This variable contains the current window that is open, in this case it is also the maximum number of windows that are open at this point. The windows start at 0, while a value of -1 means that there are no open windows. Under no circumstances should this variable be altered since it would cause unpredictable results. It is made accessible for validation purposes only.

Example.

```
/* Example of window variable */

#include "dsk1.windowsh"

extern int window;

main()
  {
    w_open("Windows");
    if(window == 0)
      w_puts("This is the first window.");
    w_close();
  }
```

Disk 98. Contents of file WINDOWSDOC

```
.co Use the RUNOFF program to print the c99 Windows Document.
.af dsk1.windows1dc
.af dsk1.windows2dc
```

Disk 99. c99 Windows — Source

Version:
Requires: c99
Author: Tom Bentley
Language: c99
Updated: 3/14/88

Source code to c99 Windows libraries which allow the C programmer to utilize a complete windows environment.

```
dskdir. v2.0. 12-dec-96
Disk name               = C99WINDOWS
Sectors total           = 360
Sectors used            = 178
Sectors available       = 180
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 MAKEWDEMO     2 DIS/VAR 80   >022 001
002 >003 MAKEWIND      2 DIS/VAR 80   >023 001
003 >004 WCMDBARC     10 DIS/VAR 80   >024 009
004 >005 WCMDC        14 DIS/VAR 80   >02d 013
005 >006 WDEMOC       10 DIS/VAR 80   >03a 009
006 >007 WGETSC       12 DIS/VAR 80   >043 011
007 >008 WINDOWC      46 DIS/VAR 80   >04e 045
008 >009 WINDOWS1G     2 DIS/VAR 80   >07b 001
009 >00a WINDOWS2G     2 DIS/VAR 80   >07c 001
010 >00b WINDOWS3G     7 DIS/VAR 80   >07d 006
011 >00c WINDOWSC     40 DIS/VAR 80   >083 039
012 >00d WINDOWSG      6 DIS/VAR 80   >0aa 005
013 >00e WINDOWSH      3 DIS/VAR 80   >0af 002
014 >00f WINDOWSI      5 DIS/VAR 80   >0b1 004
015 >010 WMSGC         4 DIS/VAR 80   >0b5 003
016 >011 WPOPUPC      13 DIS/VAR 80   >0b8 012
```

Disk 99. Contents of file MAKEWDEMO

```
LOAD DSK*.CSTARTO
LOAD DSK*.WDEMOO
LIBRARY DSK1.WINDOWS_LB
LIBRARY DSK1.C99_LB
LIBRARY DSK1.RAGLIB
LOAD DSK1.WINDOWSGO
```

Disk 99. Contents of file MAKEWIND

```
LOAD DSK*.CSTARTO
LOAD DSK*.WINDOWO
LIBRARY DSK1.WINDOWS_LB
LIBRARY DSK1.C99_LB
LIBRARY DSK1.RAGLIB
LOAD DSK1.WINDOWS3GO
LOAD DSK1.WINDOWS2GO
LOAD DSK1.WINDOWS1GO
```

Disk 99. Contents of file WCMDBARC

```
/**
 ** w_cmdbar (c99 Windows 1.0)
 **
 ** by Tom Bentley (July 19, 1987)
 **
 **/

#include "dsk1.windowsh"
#include "dsk1.windowsi"

/* external functions */
extern wputs(),hchar(),key();

/* external variables */
extern int w_raw;

/* entry point */
entry w_cmdbar;

w_cmdbar(cmd_str)
    char *cmd_str;       /* command string */
  {
    char cmd[41];        /* local command string */
    char c;
    int  cmd_pos[21][2]; /* contains start of command and size of string */
    int  l,s,cno,cmdno,x;

    hchar(BAR_ROW,1,' ',MAX_COL);
    l = YES;
    for(cno=0;cno<20;++cno)
        cmd_pos[cno][0]=cmd_pos[cno][1] = 0;

    /* parse cmd_str \n is used as a separator */
    s = cno = 0;
    cmd_pos[cno][0] = s;
    while(*cmd_str)
      {
        if(*cmd_str == '\n')
          {
            cmd_pos[cno][1] = s - 1;
            ++cmd_str;
            ++cno;
            cmd_pos[cno][0] = s;
            continue;
          }
        cmd[s++] = *cmd_str++;
      }
    cmd[s] = NULL;
```

```
wputs(BAR_ROW,1,cmd);

/* prompt for command */
--cno;
cmdno = 0;
while(l)
  {
    w_noinv(cmd);
    s = cmd_pos[cmdno][0];
    while(s <= cmd_pos[cmdno][1])
      {
        cmd[s] = cmd[s] | 128;  /* turn on high bit */
        ++s;
      }
    wputs(BAR_ROW,1,cmd);
    do
      {
        for(x=1600;x;--x);
        c = key(5,&s);
      }
    while(!s);
    switch(c)
      {
        case LEFT:
          --cmdno;
          if(cmdno < 0)
              cmdno = cno;
          break;
        case RIGHT:
          ++cmdno;
          if(cmdno > cno)
              cmdno = 0;
          break;
        case BACK:
          x = NULL;
          l = NO;
          break;
        case ENTER:
          x = cmdno + 1;
          l = NO;
          break;
        default:
          if(w_raw)
            {
              x = -c;
              l = NO;
            }
          break;
      }
  }
hchar(BAR_ROW,1,32,MAX_COL);
```

```
    return(x);
  }
```

Disk 99. Contents of file WCMDC

```
/**
 ** w_cmd (C99 Windows 1.0)
 **
 ** by Tom Bentley (August 17, 1987)
 **
 **/

#include "dsk1.windowsh"
#include "dsk1.windowsi"

/* external functions */
extern w_ibox(),w_restore(),hchar(),gchar(),key();

/* entry point */
entry w_cmd;

/* variables for windows */
extern int  w_t_row[];
extern int  w_b_row[];
extern int  w_l_col[];
extern int  w_r_col[];
extern int  window;       /* active window                */
extern int  buf_col;      /* column for virtual screen */
extern int  cur_row;      /* current row within active window */
extern int  cur_col;      /* current col within active window */
extern int  sav_row;
extern int  sav_col;
extern int  use_vbuf;
extern char w_adj_buf[];

w_adjust(n,size)
    int *n,size;
  {
    int  y;

    sav_row = W_T_ROW;
    if(!*w_adj_buf & gchar(sav_row,W_L_COL+1) != 134)
      {
        for(y=0,sav_col=W_L_COL+1;
            sav_col<W_R_COL;++sav_col)
            w_adj_buf[y++] = gchar(sav_row,sav_col);
        w_adj_buf[y] = NULL;
      }
    use_vbuf = NO;
    w_clear();
    use_vbuf = YES;
    w_ibox(window,1);
    *n = *n + size;
```

```
    w_restore();
    w_ibox(window,0);
    sav_row = W_T_ROW;
    if(*w_adj_buf)
      {
        hchar(sav_row,W_L_COL+1,160,W_R_COL-W_L_COL-1);
        for(y=0,sav_col=W_L_COL+1;
            w_adj_buf[y] & sav_col<W_R_COL;++sav_col)
            hchar(sav_row,sav_col,w_adj_buf[y++],1);
      }
    w_flush();
  }


w_cmd(c)
    char c;
  {
    char action;
    int  x,status;

    if(!c)
        do
          {
            action = key(5,&status);
          }
        while(!status);
    else
        action = c;

    switch(action)
      {
        case C_UP: /* adjust window up */
          if(W_T_ROW > MIN_ROW)
              w_adjust(&W_T_ROW,-DEC_SIZ);
          else if(W_B_ROW > W_T_ROW + DEC_SIZ)
              w_adjust(&W_B_ROW,-DEC_SIZ);
          break;
        case C_DOWN: /* adjust window down */
          if(W_B_ROW < MAX_ROW)
              w_adjust(&W_B_ROW,DEC_SIZ);
          else if(W_T_ROW < W_B_ROW - DEC_SIZ)
              w_adjust(&W_T_ROW,DEC_SIZ);
          break;
        case C_LEFT: /* adjust window left */
          if(W_L_COL > MIN_COL)
              w_adjust(&W_L_COL,-DEC_SIZ);
          else if(W_R_COL > W_L_COL + DEC_SIZ)
              w_adjust(&W_R_COL,-DEC_SIZ);
          break;
        case C_RIGHT: /* adjust window right */
```

```
        if(W_R_COL < MAX_COL)
            w_adjust(&W_R_COL,DEC_SIZ);
        else if(W_L_COL < W_R_COL + DEC_SIZ)
            w_adjust(&W_L_COL,DEC_SIZ);
        break;
      case NXT_SCR: /* display next window */
        sav_row = cur_row;
        x = (W_R_COL - W_L_COL) / 2;
        buf_col = buf_col + x;
        if(buf_col > BUF_SIZ)
            buf_col = 0;
        w_flush();
        cur_row = sav_row;
        cur_col = x;
        break;
    }
  return(action);
}
```

Disk 99. Contents of file WDEMOC

```
/**
 ** DEMO PROGRAM FOR C99 WINDOWS
 **
 **      by Tom Bentley
 **
 **/

#include "dsk3.windowsh"
#include "dsk3.grf1rf"
extern w_msg(),w_cmdbar(),w_popup();

#define YES 1
#define NO  0

main()
  {
    int x;

    w_init();
    tscrn(15,5);
    w_open(0);
    w_close();
    w_position(2,10,1,20);
    w_open("Window 1");
    w_puts("\nPresenting...");
    delay();
    w_position(6,14,5,25);
    w_open("Window 2");
    w_puts("\nA New Tool...");
    delay();
    w_position(10,18,10,30);
    w_open("Window 3");
    w_puts("\nFor C99 Programmers!!");
    delay();
    w_position(8,22,3,38);
    w_open("*** Window 4 ***");
    for(x=10;x;--x)
        w_puts("$$ c99 Windows by Tom Bentley $$\n\n");
    delay();
    w_clear();
    w_puts("Features of c99 Windows.\n\n");
    w_puts("1. Up to five opened windows.\n");
    w_puts("2. Titled windows.\n");
    w_puts("3. Message line.\n");
    w_puts("4. Command Bar.\n");
    w_puts("5. Popup Menus.\n");
    w_puts("6. 80 character virtual buffer.\n");
    w_puts("7. Easy but powerful interface.\n");
```

```
    delay();
    w_clear();
    w_puts("\n** This is a new title **");
    w_title("c99 Windows",'L');
    delay();
    w_title("c99 Windows",'R');
    delay();
    w_title("c99 Windows",'C');
    delay();
    w_clear();
    w_puts("\n** This is the message line **");
    w_msg("Plain message",NO);
    delay();
    w_msg("Inverted message",YES);
    delay();
    w_msg("",NO);
    w_clear();
    w_puts("\n** This is a command bar **\n");
    w_puts("\n\n** Press <Enter> to select\n");
    w_cmdbar(" Option 1 \n Option 2 \n Option 3 \n");
    w_clear();
    w_puts("\n** This is a popup menu **");
    w_puts("\n\n** Press <Enter> to select\n");
    w_position(15,55,7,9);
    w_popup("OPTIONS"," Option 1 \n Option 2 \n Option 3 \n");
    delay();
    w_close();
    delay();
    w_close();
    delay();
    w_close();
    delay();
    w_close();
  }

delay()
  {
    int x;

    for(x=32700;x;--x);
    for(x=10000;x;--x);
  }
```

Disk 99. Contents of file WGETSC

```
/**
 ** w_getchar & w_gets (C99 Windows 1.0)
 **
 ** by Tom Bentley (July 14, 1987)
 **
 **/

#include "dsk1.windowsh"
#include "dsk1.windowsi"

/* entry point */
entry w_getchar, w_gets;

/* variables for windows */
extern int  window;       /* active window                */
extern int  cur_row;      /* current row within active window */
extern int  cur_col;      /* current col within active window */
extern int  w_drum[];     /* buffer drum */
extern int  bcol;
extern int  use_vbuf;
extern int  w_t_row[];
extern int  w_b_row[];
extern int  w_l_col[];
extern int  w_r_col[];

w_getchar()
  {
    locate(cur_row,cur_col);
    return(getchar());
  }

w_gets(s)
    char *s;
  {
    char *b;
    int p,c;

    p = cur_row * cur_col;
    while(1)
      {
        c = w_getchar();
        switch(c)
          {
            case LEFT:
              if(p < cur_row * cur_col)
                {
                  *s-- = NULL;
                  /* put it into virtual buffer */
```

```
                if(use_vbuf)
                  {
                    b = w_drum[cur_row - W_T_ROW - 1] + bcol--;
                    *b = NULL;
                  }
                if(cur_col == W_L_COL + 1)
                  {
                    --cur_row;
                    cur_col = W_R_COL - 1;
                    bcol = W_R_COL - W_L_COL - 2;
                    w_putchar(' ');
                    --cur_col;
                  }
                else
                  {
                    --cur_col;
                    w_putchar(' ');
                    --cur_col;
                  }
              }
            break;
          case '\n':
            *s = NULL;
            return;
          default:
            *s++ = c;
            /* put it into virtual buffer */
            if(use_vbuf)
              {
                b = w_drum[cur_row - W_T_ROW - 1] + bcol++;
                *b = c;
              }
            if(cur_col >= W_R_COL - 1)
              {
                if(cur_row >= W_B_ROW - 1)
                  {
                    *s = NULL;
                    return;
                  }
                else
                  {
                    ++cur_row;
                    cur_col = W_L_COL + 1;
                    bcol = 0;
                  }
              }
             else
                 ++cur_col;
            break;
        }
    }
}
```

Disk 99. Contents of file WINDOWC

```
/*************************************
**        T H E   W I N D O W        **
*************************************
**                                   **
** Written by: Tom Bentley           **
**             P.O. BOX 346          **
**             Osgoode, Ont.         **
**             K0A 2W0               **
**                                   **
** Updated   : August 15, 1987       **
** Version   : 2.0                   **
**                                   **
*************************************/

/*************************************
** Change History
** --------------
** Version 1.2
**
** 87-03-22 Fixed problem with end of file
**          closing down browse file.
** 87-03-29 Fixed find termination, now uses
**          space bar.
**
** Version 2.0
**
** 87-07-14 Converted to use new c99
**          Windows library.
*************************************/

#include "dsk3.stdio"
#include "dsk3.windowsh"
#include "dsk3.grf1rf"

extern printf(),sprintf(),itoa(),atoi();
extern w_cmd(),w_gets(),w_cmdbar(),w_msg(),w_popup();

#define MAX_WIN  5
#define MX_WIN   4
#define DEC_SIZ  5
#define MIN_ROW  1
#define MAX_ROW  20
#define MIN_COL  1
#define MAX_COL  40
#define BUF_SIZ  82

/* define storage area */
int  p_fil = 0;        /* file pointer for paste file */
```

```
/* variables for windows */
extern int use_vbuf;
int win_fil[MX_WIN] = { 0,0,0,0 };
int win_lin[MX_WIN] = { 0,0,0,0 };
int  fnd_act    = 0;
char *fnd_ptr;

/* variables for windows */
extern int  window;       /* active window                */
extern int  buf_col;      /* column for virtual screen */
extern int  w_raw;
extern int  w_drum[];
extern int  sav_ptr[];
extern char scrn_buf[];
extern int  w_t_row[];
extern int  w_b_row[];
extern int  w_l_col[];
extern int  w_r_col[];

#define W_T_ROW  w_t_row[window]
#define W_B_ROW  w_b_row[window]
#define W_L_COL  w_l_col[window]
#define W_R_COL  w_r_col[window]
#define W_WIDTH  W_R_COL - W_L_COL - 1

#define FINISH_ACT  0
#define START_ACT   1
#define ESC_ACT     159

#define LEFT        8
#define RIGHT       9
#define DOWN        10
#define UP          11
#define C_LEFT      147
#define C_RIGHT     132
#define C_DOWN      152
#define C_UP        133
#define NXT_SCR     14
#define ROLL_UP     12
#define ROLL_DOWN   2
#define BACK        15

int action;
char  buf[BUF_SIZ];
char  fname[30];

main()
{
  int x;

  w_init();
```

```
  tscrn(15,5);
  fnd_act = 0;
  p_fil = 0;
  w_open("The WINDOW  by Tom Bentley v2.0");
  action = START_ACT;
  while(action)
      get_command();
  w_close();
  clear();
}

get_command()

{
  int act,x;

  if(window >= 0)
      status();

  w_raw = YES;
  action = w_cmdbar(" Options \n Exit \n");
  w_raw = NO;

  switch(action)
    {
      case 1: /* options */
        act = w_popup("Options",
      " Browse \n Open \n Close \n Find \n Show \n Dev \n Klose \n Paste \n");

        switch(act)
          {
            case 1: /* browse a file */
              if(window < 0)
                  w_open(NULL);
              if(!get_fn())
                  break;
              w_title(fname,'C');
              win_lin[window] = 0;
              put_window(2);
              break;
            case 2: /* Open window */
              w_open(NULL);
              win_lin[window] = 0;
              break;
            case 3: /* Close window */
              w_close();
              break;
            case 4: /* Find */
              find();
              break;
```

```
          case 5: /* Show */
            show(0);
            break;
          case 6: /* Paste Device */
            get_dev();
            break;
          case 7: /* Klose file */
            if(win_fil[window])
              {
                fclose(win_fil[window]);
                win_fil[window] = 0;
              }
            break;
          case 8: /* Paste */
            if(p_fil <= 0)
              {
                w_msg("The Paste File is not open!",YES);
                break;
              }
            paste();
            break;
        }
      break;
    case -C_UP: /* adjust window up */
    case -C_DOWN: /* adjust window down */
    case -C_LEFT: /* adjust window left */
    case -C_RIGHT: /* adjust window right */
    case -NXT_SCR: /* display next window */
      w_cmd(-action);
      break;
    case 0: /* exit window */
    case 3:
      /* make sure all files are closed */
      for(x=0;x<MAX_WIN;++x)
          if(win_fil[x] > 0)
              fclose(win_fil[x]);

      /* close paste file if open */
      if(p_fil > 0)
          fclose(p_fil);

      action = FINISH_ACT;
      break;
    case -ROLL_DOWN: /* show next page */
    case -32:
      put_window(2);
      break;
    case -ROLL_UP: /* show previous page */
      show(-(W_B_ROW - W_T_ROW - 1));
      break;
  }
```

```
}


status()

{
  char b[40];

  sprintf(b, "Line # %d  [%d %d]", win_lin[window],buf_col + 1,
          buf_col + W_WIDTH);
  w_msg(b,NO);
}


show(roll_line)
  int roll_line;
{
  int  line,num_to_read;
  char cline[10],b[2];

  if(!win_fil[window])
    {
      w_msg("You must BROWSE a file first!",YES);
      return;
    }

  if(!roll_line)
    {
      get_inp("Show Line#",cline,12);
      if(!*cline) return;
      line = atoi(cline);
    }
  else
    {
      line = roll_line;
      itoa(line,cline);
    }

  if(cline[0] == '+' | cline[0] == '-')
      /* relative positioning */
      line = win_lin[window] + line -
        (W_B_ROW - W_T_ROW - 2);


  if(line < 1)
      line = 1;
  /* if line < current line rewind file */
  if(line < win_lin[window])
    {
      rewind(win_fil[window]);
```

```
      win_lin[window] = 0;
    }

  if(win_lin[window] == 0)
      num_to_read = line - 1;
  else
      num_to_read = line - win_lin[window] - 1;

  while(num_to_read--)
    {
      if(feof(win_fil[window])) break;  /* if eof then stop */
      fgets(b,2,win_fil[window]);
      ++win_lin[window];
    }
  put_window(2);
}


find()
{
  char fstr[40],fbuf[81];
  int  more,fcol;

  if(!win_fil[window])
    {
      w_msg("You must BROWSE a file first!",YES);
      return;
    }

  get_inp("Find",fstr,40);
  if(!*fstr) return;
  w_msg("Press <SPACE BAR> to terminate find.",YES);
  more = YES;
  rewind(win_fil[window]);
  win_lin[window] = 0;
  while(more == YES)
    {
      if(feof(win_fil[window])) return;
      fgets(fbuf,BUF_SIZ - 1,win_fil[window]);
      ++win_lin[window];
      if((fcol=index(fbuf,fstr))>=0) /* found it */
        {
          buf_col = fcol;
          fnd_ptr = fbuf;
          fnd_act = YES;
          put_window(2);
          fnd_act = NO;
          status();
          more = w_cmdbar(" Continue \n Stop \n");
        }
      if(poll(NO) == ' ')
```

```
          break;
    }
}


buf_fill()

{
  int x,r;

  r = 0;
  if(fnd_act)
    {
      w_puts(fnd_ptr);
      w_putchar('\n');
      ++r;
    }

  for(x = W_T_ROW + 1 + fnd_act;
      x < W_B_ROW-1;
      ++x, ++r, ++win_lin[window])
    {
      if(feof(win_fil[window])) break;
      fgets(buf,BUF_SIZ  - 1,win_fil[window]);
      w_puts(buf);
      w_putchar('\n');
    }
}

put_window(d_type)
  int d_type;   /* display type */
{

  switch(d_type)
    {
      case 2:
        w_clear();
        buf_fill();
      case 3:
        break;
    }
}

cnt_files()
  {
    char fil_cnt,w;

    /* only 3 disk file can be open at the same time */
    for(fil_cnt=0,w=0;w<window;++w)
        if(win_fil[w] != 0)
```

```
            ++fil_cnt;
    return(fil_cnt);
  }

get_fn()

{

  if(cnt_files() > 2)
    {
      w_msg("Too many files opened!",YES);
      return;
    }

  if(win_fil[window])
    {
      fclose(win_fil[window]);
      win_fil[window] = 0;
    }

  while(!win_fil[window])
    {
      get_inp("File:",fname,28);
      if(!*fname)
          return(NO);
      if((win_fil[window] = fopen(fname,"r")) <= 0)
          win_fil[window] = fopen(fname,"R");
    }
  return(YES);
}


get_dev()
  {
    char p_nam[27];

    /* get device for paste file */
    get_inp("Paste File:",p_nam,28);
    if(!*p_nam)
        return;
    /* if it's a disk file check limit */
    if(*p_nam == 'D' | *p_nam == 'W')
        if(cnt_files() > 2)
          {
            w_msg("Too many files opened!",YES);
            return;
          }

    /* if paste file is already open close it */
    if(p_fil > 0)
        fclose(p_fil);
```

```
    if((p_fil = fopen(p_nam,"a")) <= 0)
        w_msg("Can't open Paste File!",YES);
  }

paste()
  {
    char b[40],p_win[41],*p,*d,ch;
    int r,c,i,w;

    if(window < 0)
      {
        w_msg("No Window to Paste!",YES);
        return;
      }

    get_inp("Window(s):",p_win,40);
    if(!*p_win)
        return;

    p = p_win;
    while(*p)
      {
        /* A is entered then the active virtual buffer is pasted */
        if(win_fil[window] > 0 & (*p == 'A' | *p == 'a'))
          {
            w_msg("Pasting Active Buffer.",NO);
            for(i=0,r=w_t_row[window]+1;r<w_b_row[window];++r,++i)
              {
                d = w_drum[i];
                fputs(d,p_fil);
              }
            ++p;
            continue;
          }

        /* if it's a bad window skip it */
        w = *p - '1';
        if(w < 0 | w > MX_WIN)
          {
            ++p;
            continue;
          }

        sprintf(b,"Pasting Window (%d)",w+1);
        w_msg(b,NO);
        i=sav_ptr[w] + (W_WIDTH);
        for(r=w_t_row[w]+1;r<w_b_row[w];++r)
          {
            for(c=w_l_col[w]+1;c<w_r_col[w];++c)
              {
```

```
                /* if its the current window its not in scrn_buf yet */
                if(w == window)
                    ch = gchar(r,c);
                else
                    ch = scrn_buf[i++];
                putc(ch,p_fil);
              }
            putc('\n',p_fil);
          }
        ++p;
      }
  }

get_inp(title,buf,siz)
    char *title,*buf;
    int  siz;
  {
    w_position(21,23,1,siz);
    use_vbuf = NO;
    w_open(title);
    w_gets(buf);
    w_close();
    use_vbuf = YES;
  }


/* name: index(string,substring)
 * function: Returns the starting
 * location of the substring in
 * the string, or the value -1 if
 * the substring was not found.
 * ex. index("This is it","it");
 *       returns 8
 *     index("This is it","IT");
 *       returns -1
 */
index(string,substring)
  char string[],substring[];
  {
    int i,j,k;

    i=0;
    while(string[i]!=0) {
      j=i;
      k=0;
      while(substring[k]==string[j]) {
        if(substring[1+k]==0)
          return(i);
        else {
          j++;
          k++;
```

```
          }
        }
      i++;
      }
    return(-1);
    }

/*
 * String copy
 *
 *  strcpy(s,t) - copy t to s
 *
 */

strcpy(s,t)
  char *s,*t;
  {
    while(*s++ = *t++);
  }
```

Disk 99. Contents of file WINDOWS1G

```
/**
 ** Globals for C99 Windows
 **
 **/

#include "dsk1.windowsi"

entry w_buf;

char w_buf[NUM_ROWS][BUF_SIZ]; /* virtual buffer */
```

Disk 99. Contents of file WINDOWS2G

```
/**
 ** Globals for C99 Windows
 **
 **/

#include "dsk1.windowsi"

entry scrn_buf;

char scrn_buf[SBUF_SIZ];  /* contains old windows */
```

Disk 99. Contents of file WINDOWS3G

```
 /*
 ** Globals for C99 Windows (August 17, 1987)
 **
 **/

#include "dsk3.windowsi"

entry w_drum;
entry sav_ptr,w_t_row,w_b_row;
entry w_l_col,w_r_col;
entry window,buf_col,bcol,scrn_nxt;
entry cur_row,cur_col,sav_row,sav_col;
entry use_vbuf,wtitle;
entry w_rmargin,w_raw,w_adj_buf;

char *wtitle;          /* pointer to window title */
int  w_drum[NUM_ROWS]; /* scroll drum */
int  sav_ptr[MAX_WIN]; /* pointers into scrn_buf */
int  w_t_row[MAX_WIN]; /* top row position for windows */
int  w_b_row[MAX_WIN]; /* bottom row position for windows */
int  w_l_col[MAX_WIN]; /* left column position for windows */
int  w_r_col[MAX_WIN]; /* right column position for windows */
int  window;           /* active window                */
int  buf_col;          /* column for virtual screen */
int  bcol;             /* column in virtual buffer */
int  scrn_nxt;         /* next scrn_buf location */
int  cur_row;          /* current row within active window */
int  cur_col;          /* current col within active window */
int  sav_row;          /* used to store row */
int  sav_col;          /* used to store column */
int  use_vbuf;         /* use virtual buffer ? */
int  w_rmargin;        /* right margin in virtual buffer */
int  w_raw;            /* don't perform validation on entry */
char w_adj_buf[MAX_COL]; /* used for window adjustments */
```

Disk 99. Contents of file WINDOWSC

```
/************************************
**                                **
**  Kernal Library for c99 Windows  **
**                                **
************************************
**  Author: Tom Bentley           **
**  Date  : August 17, 1987       **
**          Box 346               **
**          Osgoode, Ont.         **
**          K0A 2W0               **
************************************
**  Version : 1.0                 **
************************************/

#include "dsk1.grf1rf"
#include "dsk1.windowsi"
extern printf();

entry w_init,w_locate,w_title;
entry w_clrbuf,w_putchar,w_puts,w_clear;
entry w_open,w_close,w_box,w_inv;
entry w_position,fillb,strlen;
entry w_ibox,w_restore,w_flush;
entry w_scroll,wputs,w_noinv;

extern char w_buf[NUM_ROWS][BUF_SIZ];
extern char scrn_buf[];
extern int  w_drum[];  /* buffer drum */
extern int  sav_ptr[]; /* pointer to screen save buffer*/
extern int  w_t_row[];
extern int  w_b_row[];
extern int  w_l_col[];
extern int  w_r_col[];

/* variables for windows */
extern int  window;       /* active window                */
extern int  buf_col;      /* column for virtual screen */
extern int  bcol;         /* column in virtual buffer */
extern int  scrn_nxt;     /* next scrn_buf location */
extern int  cur_row;      /* current row within active window */
extern int  cur_col;      /* current col within active window */
extern int  sav_row;
extern int  sav_col;
extern int  use_vbuf;     /* use virtual buffer ? */
extern int  w_rmargin;
extern int  w_raw;
extern char *wtitle;      /* pointer to window title */
extern char w_adj_buf[];
```

```
w_init()
{
  int x;

  for(x=0;x<MX_WIN;++x)
    {
      w_t_row[x] = MIN_ROW;
      w_b_row[x] = MAX_ROW;
      w_l_col[x] = MIN_COL;
      w_r_col[x] = MAX_COL;
    }
  w_clrbuf();
  for(x=0;x<NUM_ROWS;++x)
      w_drum[x] = &w_buf[x][0];
  window = -1;
  buf_col = 0;
  scrn_nxt = 0;
  use_vbuf = YES;
  w_rmargin = BUF_SIZ - 1;
  w_raw = NO;
  *w_adj_buf = NULL;
  text();
  /* copy patterns to second set */
  for(x=0;x<128;++x)
      patcpy(x,x+128);
  invert();
  set_lines();
}

w_locate(r,c)
    int r,c;
  {
    cur_row = W_T_ROW + r;
    if(cur_row >= W_B_ROW)
        cur_row = W_B_ROW - 1;
    cur_col = W_L_COL + c;
    bcol = c - 1;
  }

/*
** w_position sets up the location of
** the next window that will be opened.
** tr is the top row, br is the bottom
** row, lc is the left column and rc
** is the right column.
*/
w_position(tr,br,lc,rc)
  int tr,br,lc,rc;
{
  int w;
```

```
  w = window + 1;
  w_t_row[w] = tr;
  w_b_row[w] = br;
  w_l_col[w] = lc;
  w_r_col[w] = rc;
}

w_clrbuf()
  {
      fillb(w_buf,NULL,NUM_ROWS * BUF_SIZ);
  }

w_flush()
  {
    char *b;
    int  row,c;

    for(row=0,sav_row=W_T_ROW+1;sav_row < W_B_ROW;++sav_row,++row)
      {
        b = w_drum[row] + buf_col;
        sav_col = W_L_COL + 1;
        while(*b & sav_col < W_R_COL)
            hchar(sav_row,sav_col++,*b++,1);
        if((c = W_R_COL - sav_col) > 0)
            hchar(sav_row,sav_col,32,c);
      }
  }

w_putchar(c)
    char c;
  {
    char *b;

    switch(c)
      {
        case '\n':
          if(cur_row >= W_B_ROW - 1)
            {
              w_scroll();
              break;
            }
          else
              ++cur_row;
        case '\r':
          cur_col = W_L_COL + 1;
          bcol = 0;
          break;
        case '\t':
          if(bcol + TAB < BUF_SIZ)
            {
```

```
            cur_col = cur_col + TAB;
            bcol = bcol + TAB;
          }
        break;
      case FF:
        w_clear();
        break;
      default:
        if(use_vbuf)
          {
            /* see if we need to scroll */
            if(bcol >= w_rmargin)
              {
                if(cur_row >= W_B_ROW - 1)
                  {
                    w_scroll();
                  }
                else
                    ++cur_row;
                bcol = 0;
                cur_col = W_L_COL + 1;
              }

            /* put it into virtual buffer */
            b = w_drum[cur_row - W_T_ROW - 1] + bcol++;
            *b = c;
          }

        /* put it to the screen */
        if(cur_col < W_R_COL)
            hchar(cur_row,cur_col++,c,1);
    }
  return(c);
}

w_puts(s)
  char *s;
{
  while(*s)
      w_putchar(*s++);
}

w_scroll()
{
  char *sav;
  int  x;

  sav = w_drum[0];
  for(x=0;x < W_B_ROW - W_T_ROW -2;++x)
      w_drum[x] = w_drum[x+1];
  fillb(sav,NULL,BUF_SIZ);
```

```
    w_drum[x] = sav;
    w_flush();
    cur_col = W_L_COL + 1;
    bcol = 0;
  }

w_clear()
  {
    int x,wid;

    w_locate(1,1);
    wid = W_R_COL - W_L_COL - 1;
    for(x=W_T_ROW+1;x<W_B_ROW;++x)
          hchar(x,W_L_COL+1,32,wid);
    if(use_vbuf)
      {
        w_clrbuf();
        buf_col = 0;
      }
  }

w_open(title)
  char *title;
{
  if(window < MX_WIN)
    {
      if(window >= 0)
          w_save();
      ++window;
      w_locate(1,1);
      wtitle = title;
      w_clear();
      w_ibox(window,0);
      wtitle = *w_adj_buf = NULL;
      return(YES);
    }
  return(NO);
}

w_close()

{
  if(window >= 0)
    {
      w_ibox(window,1);
      w_clear();
      w_restore();
      w_t_row[window] = MIN_ROW;
      w_b_row[window] = MAX_ROW;
      w_l_col[window] = MIN_COL;
```

```
      w_r_col[window] = MAX_COL;
      --window;
      if(window >= 0)
        {
          w_locate(1,1);
          scrn_nxt = sav_ptr[window];
        }
      else
          scrn_nxt = 0;
    }
}

w_save()

{
  sav_ptr[window] = scrn_nxt;
  for(sav_row=W_T_ROW;sav_row<W_B_ROW;sav_row++)
      for(sav_col=W_L_COL+1;sav_col<W_R_COL;sav_col++)
          scrn_buf[scrn_nxt++] = gchar(sav_row,sav_col);
}


w_restore()

{
  int i,w;

  if(window < 1)
      return;

  for(w=0;w<window;++w)
    {
      w_ibox(w,0);
      for(i=sav_ptr[w],sav_row=w_t_row[w];sav_row<w_b_row[w];++sav_row)
          for(sav_col=w_l_col[w]+1;sav_col<w_r_col[w];++sav_col)
              hchar(sav_row,sav_col,scrn_buf[i++],1);
    }
}


w_ibox(w,clear)

int w,clear;

{
  w_box(w_t_row[w],w_l_col[w],w_b_row[w],w_r_col[w],clear);
}


/*
** set_lines sets up the definitions for a line
```

```
** drawing character set.
**
** the grf1 graphics library must be loaded.
**
*/

set_lines()
{
  /* 128 - Top Left corner  */
  chrdef(128,"FF848484848484FF");
  /* 129 - Top Right corner */
  chrdef(129,"80C0E0F0E8E4E4E4");
  /* 130 - Bottom left corner */
  chrdef(130,"FF7F3F1008040404");
  /* 131 - Bottom right corner */
  chrdef(131,"E4E4E404040404FF");
  /* 132 - Left vertical */
  chrdef(132,"8484848484848484");
  /* 133 - Right vertical */
  chrdef(133,"E4E4E4E4E4E4E4E4");
  /* 134 - Top horizontal */
  chrdef(134,"FF0000000000000FF");
  /* 135 - Bottom horizontal */
  chrdef(135,"FFFFFF00000000FF");
}


/* invert goes through the pattern
** descriptor table and inverts all
** character patterns so there is an
** inverted character set from 128-255.
*/
invert()
  {
#asm
       REF  VSBR,VSBW
PAT    EQU  >0C00
       LI   2,128*8      NUMBER OF CHARS
       LI   0,PAT        START OF PATTERNS
INVL   BLWP @VSBR        READ A CHARACTER
       INV  1            INVERT BIT PATTERN
       BLWP @VSBW        WRITE THE CHARACTER
       INC  0
       DEC  2
       JNE  INVL
#endasm
  }

w_inv(s)
    char *s;
```

```
  {
    while(*s)
        *s = *s++ | 128;
  }

w_noinv(s)
    char *s;
  {
    while(*s)
        *s = *s++ & 127;
  }


wputs(r,c,s)
  int  r,c;
  char *s;
{
  while(*s & c <= MAX_COL)
      hchar(r,c++,*s++,1);
}

w_title(t,c)
  char *t,c;
{
  int len,width,mid;

  len = strlen(t);
  width = W_R_COL - W_L_COL - 1;
  if(len > width)
      return;
  switch(c)
    {
      case 'C':  /* center */
        mid = (width - len) / 2 + W_L_COL + 1;
        break;
      case 'R':  /* righ justify */
        mid = W_R_COL - len;
        break;
      case 'L':  /* left justify */
      default:
        mid = W_L_COL + 1;
        break;
    }
  *w_adj_buf = NULL;
  w_inv(t);
  hchar(W_T_ROW,W_L_COL + 1,160,width);
  wputs(W_T_ROW,mid,t);
}

/* w_box draws a box specified by top row (tr), top column (tc),
** bottom row (br) and bottom column (bc).
```

```
*/
w_box(tr,tc,br,bc,clear)

  int tr,tc,br,bc,clear;
{
  int cnt,c1,c2,c3,c4,c5,c6,c7,c8;

  /* set up character numbers */
  if(clear)
      c1=c2=c3=c4=c5=c6=c7=c8 = ' ';
  else
    {
      c1 = 128;
      c2 = 129;
      c3 = 130;
      c4 = 131;
      c5 = 132;
      c6 = 133;
      c7 = 134;
      c8 = 135;
    }

  /* draw horizontal lines */
  cnt = (bc - tc) + 1;
  hchar(tr,tc,c7,cnt);
  hchar(br,tc,c8,cnt);

  /* draw vertical lines   */
  cnt = (br - tr) + 1;
  vchar(tr,tc,c5,cnt);
  vchar(tr,bc,c6,cnt);

  /* fill in corners       */
  /* top left              */
  hchar(tr,tc,c1,1);
  /* bottom left           */
  hchar(br,tc,c3,1);
  /* top right             */
  hchar(tr,bc,c2,1);
  /* bottom right          */
  hchar(br,bc,c4,1);

  /* put in title */
  if(wtitle != NULL)
      w_title(wtitle,'C');
}


/* fill s with t for n times */
fillb(s,t,n)
```

```
  char *s,t;
  int   n;
  {
  while(n--)
    *s++=t;
  }

strlen(s)
  char *s;
{
  char *t;

  t=s-1;
  while(*++t);
  return(t-s);
}
```

Disk 99. Contents of file WINDOWSG

```
/**
 ** Globals for c99 Windows (August 17, 1987)
 **
 **/

#include "dsk3.windowsi"

entry w_buf,w_drum,scrn_buf;
entry sav_ptr,w_t_row,w_b_row;
entry w_l_col,w_r_col;
entry window,buf_col,bcol,scrn_nxt;
entry cur_row,cur_col,sav_row,sav_col;
entry use_vbuf,wtitle;
entry w_rmargin,w_raw,w_adj_buf;

char *wtitle;      /* pointer to window title */
char w_buf[NUM_ROWS][BUF_SIZ];
char scrn_buf[SBUF_SIZ];
int  w_drum[NUM_ROWS];
int  sav_ptr[MAX_WIN];
int  w_t_row[MAX_WIN];
int  w_b_row[MAX_WIN];
int  w_l_col[MAX_WIN];
int  w_r_col[MAX_WIN];
/* variables for windows */
int  window;       /* active window                */
int  buf_col;      /* column for virtual screen */
int  bcol;         /* column in virtual buffer */
int  scrn_nxt;     /* next scrn_buf location */
int  cur_row;      /* current row within active window */
int  cur_col;      /* current col within active window */
int  sav_row;
int  sav_col;
int  use_vbuf;     /* use virtual buffer ? */
int  w_rmargin;    /* right margin in virtual buffer */
int  w_raw;        /* don't perform validation on entry */
char w_adj_buf[MAX_COL]; /* buffer for window adjustments */
```

Disk 99. Contents of file WINDOWSH

```
/************************************
**                                  **
**  Run Time Library for WINDOWS    **
**                                  **
************************************/

extern w_init(),w_locate(),w_title();
extern w_clrbuf(),w_putchar(),w_puts();
extern w_open(),w_close(),w_box(),w_inv();
extern w_position(),fillb(),strlen();
extern w_noinv(),w_clear();
extern w_flush(),w_scroll();
```

Disk 99. Contents of file WINDOWSI

```
/**
 ** Defines for C99 Windows
 **
 **/

#define YES      1
#define NO       0
#define NULL     0
#define FF       12
#define TAB      5
#define SBUF_SIZ 2922 /* ROWS * COLS * WINDOWS-1 (21*38*4) */
#define MAX_WIN  5
#define MX_WIN   4
#define DEC_SIZ  5
#define MIN_ROW  2
#define MAX_ROW  23
#define MIN_COL  1
#define MAX_COL  40
#define NUM_ROWS 22
#define NUM_COLS 38
#define BAR_ROW  1
#define MSG_ROW  24
#define BUF_SIZ  81
#define W_T_ROW  w_t_row[window]
#define W_B_ROW  w_b_row[window]
#define W_L_COL  w_l_col[window]
#define W_R_COL  w_r_col[window]
#define LEFT        8
#define RIGHT       9
#define DOWN        10
#define UP          11
#define C_LEFT      147
#define C_RIGHT     132
#define C_DOWN      152
#define C_UP        133
#define NXT_SCR     14
#define ROLL_UP     12
#define ROLL_DOWN   2
#define BACK        15
#define ENTER       13
```

Disk 99. Contents of file WMSGC

```
/**
 ** w_msg (c99 Windows 1.0)
 **
 ** by Tom Bentley (May 28,1987)
 **
 **/

#include "dsk1.windowsh"
#include "dsk1.windowsi"

/* external functions */
extern wputs(),hchar();

/* entry point */
entry w_msg;

w_msg(msg,inv)
    char *msg;    /* message text */
    int   inv;    /* invert msg YES or NO */
  {
    if(inv)
      {
        hchar(MSG_ROW,1,160,MAX_COL);
        w_inv(msg);
      }
    else
      {
        hchar(MSG_ROW,1,32,MAX_COL);
        w_noinv(msg);
      }
    wputs(MSG_ROW,1,msg);
  }
```

Disk 99. Contents of file WPOPUPC

```
/**
 ** w_popup (c99 Windows 1.0)
 **
 ** by Tom Bentley (July 19, 1987)
 **
 **/

#include "dsk1.windowsh"
#include "dsk1.windowsi"

/* external functions */
extern gchar(),key();

/* entry point */
entry w_popup;

/* variables for windows */
extern int  w_t_row[];
extern int  w_b_row[];
extern int  w_l_col[];
extern int  w_r_col[];
extern int  window;       /* active window                */
extern int  cur_row;      /* current row within active window */
extern int  cur_col;      /* current col within active window */
extern int  sav_row;
extern int  sav_col;
extern int  use_vbuf;     /* use virtual buffer ? */
extern int  w_raw;

w_popup(title,cmd_str)
    char *title,*cmd_str;
  {
    char cmd[41];        /* local command string */
    char *cs,c;
    int  l,s,cno,cmdno,x,w,r,col;

    l = YES;
    /* parse cmd_str \n is used as a separator */
    s = cno = 0;
    w = strlen(title);
    cs = cmd_str;
    while(*cs)
      {
        if(*cs == '\n')
          {
            if(s>w)
                w = s;
            s = 0;
```

```
        ++cno;
      }
    else
        s++;
    cs++;
  }

--cs;
*cs = NULL;  /* get rid of last \n */
w_b_row[window+1] = w_t_row[window+1] + cno + 1;
w_r_col[window+1] = w_l_col[window+1] + w + 1;
r = cur_row;
col = cur_col;
use_vbuf = NO;
w_open(title);
w_puts(cmd_str);
*cs = '\n'; /* put back newline for next time around */
/* prompt for command */
cmdno = 1;
while(l)
  {
    /* get command from window */
    x = W_T_ROW + cmdno;
    sav_row = cmdno;
    sav_col = W_L_COL + 1;
    cs = cmd;
    while(sav_col < W_R_COL)
      {
        c = gchar(x,sav_col++);
        *cs++ = c;
      }
    *cs = NULL;
    w_inv(cmd);
    w_locate(sav_row,1);
    w_puts(cmd);
    do
      {
        for(x=900;x;--x);
        c = key(5,&s);
      }
    while(!s);
    switch(c)
      {
        case UP:
          --cmdno;
          if(cmdno < 1)
              cmdno = cno;
          break;
        case DOWN:
          ++cmdno;
          if(cmdno > cno)
```

```
                cmdno = 1;
              break;
            case BACK:
              x = NULL;
              l = NO;
              break;
            case ENTER:
              x = cmdno;
              l = NO;
              break;
            default:
              if(w_raw)
                {
                  x = -c;
                  l = NO;
                }
              break;
          }
        w_noinv(cmd);
        w_locate(sav_row,1);
        w_puts(cmd);
      }
    w_close();
    cur_row = r;
    cur_col = col;
    use_vbuf = YES;
    return(x);
  }
```

Disk 100. Telco — Program Disk

Version: 2.1 *Author*: Charles Earl
Requires: EA or XB *Language*: AL *Updated*: 10/28/88

The most user friendly and perhaps powerful terminal emulator for the 99/4A. Includes many modes of terminal operation including VT100, supports 80 columns, full catalog, print spooling, macros, on-line editor, and much more. Highly acclaimed.

```
dskdir. v2.0. 12-dec-96
Disk name               = TELCO2*1
Sectors total           = 360
Sectors used            = 346
Sectors available       = 12
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >01d *README       3 DIS/VAR 80   >159 002
002 >01e CHARA1        5 PROGRAM      >15b 004
003 >002 LOAD          6 PROGRAM    Y >022 005
004 >003 T>ADM3A      10 PROGRAM      >027 009
005 >004 T>ANSI       11 PROGRAM      >030 010
006 >005 T>ASCII       6 PROGRAM      >03a 005
007 >006 T>CATALOG    15 PROGRAM      >03f 014
008 >007 T>CISB       17 PROGRAM      >04d 016
009 >008 T>D410       10 PROGRAM      >05d 009
010 >009 T>DIALER     17 PROGRAM      >066 016
011 >00a T>EDITOR     13 PROGRAM      >076 012
012 >00b T>HP2392     11 PROGRAM      >082 010
013 >00c T>HREVIEW     3 PROGRAM      >08c 002
014 >00d T>MAINMENU   16 PROGRAM      >08e 015
015 >00e T>PCPDIAL    12 PROGRAM      >09d 011
016 >00f T>SETCOLOR    5 PROGRAM      >0a8 004
017 >010 T>SETFILEX   11 PROGRAM      >0ac 010
018 >011 T>SETHARD    12 PROGRAM      >0b6 011
019 >012 T>SETMACRO    7 PROGRAM      >0c1 006
020 >013 T>SETMODEM    9 PROGRAM      >0c7 008
021 >014 T>SETTERM    16 PROGRAM      >0cf 015
022 >015 T>VT100      12 PROGRAM      >0de 011
023 >016 T>VT52       11 PROGRAM      >0e9 010
024 >017 T>XMODEM     16 PROGRAM      >0f3 015
025 >018 T>YMODEM     16 PROGRAM      >102 015
026 >019 TELCO        33 PROGRAM      >111 032
```

```
027 >01a TELCP        26 PROGRAM      >131 025
028 >01b TELCQ        11 PROGRAM      >14a 010
029 >01c TOS/CONFIG    6 PROGRAM      >154 005
```

Disk 100. Contents of file *README

For the people who have downloaded version 2.0 of Telco this package has been provided to update the user to the 2.1 level. Note that this update can not be used as a "Quick Fix" for 1.x users. These users must download the complete 2.1 package.

Correction to the manual in 2.0.

On page 14 of the documentation, near the bottom of the page "PCDIAL" should read "PCPDIAL".

Charles Earl

Disk 101. Telco — Documentation

Version: 2.1 *Author*: Charles Earl
Requires: *Language*: *Updated*: 10/28/88

Complete documentation for disk 100. You will need this to really make use of Telco.

```
dskdir. v2.0. 12-dec-96
Disk name               = TELCO2*1
Sectors total           = 360
Sectors used            = 358
Sectors available       = 0
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >003 PRINTDOC      2 PROGRAM      >0db 001
002 >004 READMEPR      2 DIS/VAR 80   >0dc 001
003 >002 TELCOD1     186 DIS/VAR 80   >022 185
004 >005 TELCOD2     168 DIS/VAR 80   >0dd 139 >006 028
```

Disk 101. Contents of file READMEPR

The Telco documentation is not a TI-Writer file. It should be printed out with the enclosed XB Printdoc program. If your printer's form feeds are not character 12, on line 150 of the printdoc program, change 12 to the appropriate character.

Disk 101. Contents of file TELCOD1

TELCO Terminal Emulator

Version 2.0

Reference Manual

Specifications within this manual are subject to change without notice.

Most of the hardware names in this manual are trademarks or trade names of specific manufacturers.

LICENSE

No version of TELCO is public domain software, nor is it free software. TELCO is user-supported software, as explained in this license. TELCO is copyright (c) 1988 by Charles Earl.

Non-registered users are granted a limited license to use TELCO on a trial basis for the purpose of determining whether or not TELCO is suitable for their needs. Use of TELCO, except for this purpose, requires registration. Use of non-registered copies of TELCO by any person on a continuing basis or in a business or professional office is strictly forbidden.

Registration permits a user to use TELCO on a single computer; a registered user may use the program on a different computer, but may not use the program on more than one computer at the same time.

No user may modify TELCO in any way, including but not limited to decompiling, disassembling or otherwise reverse engineering the program.

All users are granted a limited license and are encouraged to copy TELCO for the trial use of others subject to the above limitations, and provided the following conditions are met:

1. TELCO must be copied in unmodified form.
2. The documentation must be included.
3. No fee, charge or other compensation may be accepted or requested by any licensee.
4. TELCO may not be distributed in conjunction with any other product.

Sysops of BBS's are encouraged to make TELCO available for downloading provided the above conditions have been met.

No individual or organization is permitted to collect the *registration fee* for the author unless express written permission is obtained and that permission is displayed to those paying the registration fee in such a manner. Non-commercial users' groups are encouraged to distribute TELCO through their libraries, provided that only minimal fees are charged to recover media and copying costs.

A TELCO registration licenses you to use TELCO on a regular basis. Registration includes mailed notification of the next update, and further updates at the author's discretion.

The registration fee is $20.00. To order copies of TELCO, register, or obtain written permission, contact:

Charles Earl
34 McLeod Street
Ottawa, Ontario
Canada K2P 0Z5

TELCO Registration Form Version 2.0

Name of User: _____

Address: _____

_____

_____

_____

# of copies to register @ $20.00 ea.: _____

Obtained copy of TELCO from:
_____

If you are registering a copy of TELCO for which the registration fee was paid in advance, i.e. to a group collecting it for the author, please enclose a copy of the receipt (if possible), and fill out the following information: (Note: This form MUST be returned to the author to complete your registration.)

Name of organization collecting fee:
_____

Event at which it was collected:
_____

Date of payment: _____

Make check or money order payable to Charles Earl, and send it with this form to:

Charles Earl
34 McLeod Street
Ottawa, Ontario
Canada
K2P 0Z5

Thank you for contributing to a user-supported product. If there are features you would like to see in a future release, please list them on the back of this form or on a separate sheet.

TABLE OF CONTENTS

INTRODUCTION

TELCO has been designed to provide the TI user with easy and convenient access to a variety of telecommunications tasks. This program has been written in TMS 9900 assembly language for optimum performance. Features include:

The ability to emulate various terminals.

A Dialer with 99 entries with automatic redialing of up to 15 selections.

A Conference mode, available in all terminal emulations.

Xmodem and ASCII transfer capability.

Macros, Device logging, Print spooling etc...

NEW IN VERSION 2.0

PC Pursuit Dialer
Ymodem transfers
CompuServe B transfers (binary and ASCII)
Significantly faster Xmodem transfers
Two new terminal emulations: VT-52 and HP2392
Terminal modes improved
Select options directly from help screens
Enhanced editor

Improved Conference mode
Module capacity increased for Geneve and Super Space II users
80-column card support
Print spooler now supports RS232 and Myarc PIO
512-character buffer in print spooler

I would like thank the following people and organizations (in alphabetical order) for their support and the assistance they provided, in the form of hardware loans, access time, and information. While many people have contributed to the development of TELCO, these deserve recognition, and I apologize for any possible omissions.

Chris Bobbitt (Asgard)
Barry Boone
Bob Boone (Computer Download Unlimited)
Jeff Guide and Delphi
Jim Horn and CompuServe
Walt Howe
Jane Laflamme (Laflamme & Wrigley)
The Ottawa TI-99/4A Users' Group

I would also like to thank all those who have already registered their copies of TELCO, both for their support and their helpful suggestions.

HARDWARE REQUIREMENTS

TELCO has been designed to operate on both the TI-99/4A and the Geneve 9640. The minimum system required to use TELCO is:

TI-99/4A console or Geneve 9640
32K Memory Expansion (TI-99/4A only)
RS232 Card
1 SSSD Disk Drive
Modem (Hayes compatible preferred)

E/A, TI-Writer, Minimem, or Extended Basic cartridge

Additional Hardware (optional)

Horizon Ramdisk
Modified RS232 Cable
Printer
Additional Disk Drive(s)
Supercart or Super Space
Super Space II
80-column card

TELCO has been developed using the overlay concept. This means that TELCO will, on occasion, load a function from diskette. It is advisable to keep the TELCO diskette in the drive it was loaded from for easy access to the functions. If for any reason you removed the diskette (as single-drive users will for file transfers, etc.), and TELCO requires a function from disk, just reinsert the TELCO disk and select the function over again. TELCO will not damage your diskette.

On a basic system, up to 3 overlays will reside for use within memory. These will be the last 3 overlays used. If a minimem is used, 4 overlays are available. If a supercart or Super Space is used, 5 overlays are possible. With a 32K Super Space II, as many as 11 modules are available, a TI with an 80-column card will support 27 modules, and a Geneve will support 29. TELCO will automatically detect which type of cartridge is in use and configure the system accordingly.

Xmodem and CompuServe B transfers do not currently work on the Myarc Hard-disk controller. This may change when the final version of the card is available, but is difficult to predict with any degree of accuracy.

TELCO FILES

TELCO consists of three types of files. The main program is called TELCO, TELCP and TELCQ. These files contain the basic functions and utilities required. The second type of file is the module file. These files start with T> and contain the logic required to perform their specific function.

T>ADM3A ADM3A terminal emulation
T>ANSI ANSI terminal emulation
T>ASCII ASCII upload
T>CATALOG Catalog a disk
T>CISB CompuServe B transfer protocol
T>D410 D410 terminal emulation
T>DIALER Auto redialer
T>EDITOR Message editor
T>HP2329 Hewlett Packard 2329 emulation
T>HREVIEW Review buffer help screen
T>MAINMENU Main menus
T>PCPDIAL PC Pursuit dialer
T>SETCOLOR Set the program colors
T>SETFILEX File transfer setup
T>SETHARD Hardware setup
T>SETMACRO Macro editor
T>SETMODEM Modem setup
T>SETTERM Terminal setup
T>VT52 VT52 terminal emulation
T>VT100 VT100 terminal emulation
T>XMODEM Xmodem transfer protocol
T>YMODEM Ymodem transfer protocol

No files of the third type have been provided with the package, as TELCO makes them if they are needed.

TOS/CONFIG Configuration of TELCO
TOS/PCP PC Pursuit Dial directory
TOS/PHONE Phone directory

The documentation is contained in the files TELCOD1, TELCOD2, TELCOD3, etc., and may be printed out using the included PRINTDOC utility. The READMEPR file is simply instructions for printing out the documentation. Please note that the documentation is not meant to be printed through TI-Writer. If you do this, it may not read properly, which could result in some confusion.

An extended basic loader called LOAD has been provided, which will work with the TI extended basic cartridge but may not work with the newer extended basics. A CHARA1 file has also been included, although TELCO will work with any CHARA1 file. If no CHARA1 file exists on the diskette, the current character set will be used.

Users of TELCO V 1.x may use their old TOS/PHONE file with TELCO 2.0, but the TOS/CONFIG file must be recreated. This means that macros and other settings from version 1.3 must be re-entered when you first start using version 2.0.

GETTING STARTED

To run TELCO from the E/A environment:
Select option 5 and enter:
DSK1.TELCO

To run TELCO from the TI-Writer environment:
Select option 3 and enter:
DSK1.TELCO

A program called LOAD has been provided to auto-load TELCO when the Extended Basic cartridge is used.

Once the program is loaded and begins execution, it will search for a CHARA1 file. If no CHARA1 file is in the drive from which TELCO was loaded, the program will use the default character set in memory. Any CHARA1 file may be used.

After loading the CHARA1 file, TELCO will check the drive for the TOS/CONFIG file. This file did not come with your package; it is created by TELCO once the program has been loaded for the first time or if the config file has been erased.

At this point, the Title screen will be displayed. After the program has been booted thirty times, the title screen will not be displayed. The title screen may, however, be called up from the Main Menu at any time.

For users of Myarc Disk Controller cards and the CorComp 512K ramdisks, the drive access detection scheme may not work properly. If this scheme fails to detect the drive number, TELCO will not be able to load modules, and the title screen will not be displayed. If this happens reboot TELCO while holding down the [Enter] key. You will be prompted to enter a drive number. Enter the number of the drive you want TELCO to run from. (CorComp Ramdisk users may also enter the letter "R".) Should you wish to return to the last drive access scheme at any point, enter "0". Once any of these selections has been chosen, the file TELCO will be saved to the drive specified. Be sure that there is no write- protect tab present.

The Status Line

On the bottom line of your screen there is a status line which looks like this:

```
00:00:00 1200 8N1 F LA PA RW MAINMENU
```

The first item on the status line, from left to right, is the elapsed time clock. This clock will be reset to 00:00:00 when the dialer connects to a BBS via the autodialer, and may be reset manually within the terminals by pressing **FCTN Q**.

The second item is the current system baud rate. This example is set to 1200 bps. Following baud rate is parity. In the example above, this is set to 8 data bits, no parity checking, and 1 stop bit.

Next is the full/half duplex flag. When TELCO is set to full duplex, the system expects the remote system to echo all transmissions back to the user. When it is set to half duplex, TELCO will automatically echo transmissions back to the user.

After the duplex flag is the log flag. There are four settings for this flag. Once the user opens the log, the (LW) flag will appear. This flag indicates that the log is open and waiting for the current line, the one the cursor was on when the log was opened, to scroll off the screen. Once that happens, the (LA) flag will appear. This flag is used to indicate that all lines scrolling off the top of the screen will appear in the log. When the user closes the log, the (LC) flag will appear. This flag indicates that the log is closing. If the log hold feature is selected, the (LC) flag will appear until the line at which you selected hold scrolls off the top of the screen. At this point, the (LH) flag will appear until such time as the user toggles the log hold off. When this is done, the (LW) flag will appear.

After the log flag is the print spooler flag. This flag has three possible settings, similar to those for the log. (PW) indicates Print spool Waiting to start, (PA) indicates Print spool Active, and (PC) indicates Print spool Closing.

This unusual buffer design may seem strange at first, but this will allow the log and print spooler to duplicate the lines of text in environments where full cursor movement is used. For those who have no need of this feature, or who are not comfortable with it, the spooler may be set to "immediate" (IMM) in the Terminal setup. The default setting is "delay", which acts as described above.

The next flag that appears is the (R) flag for remote echo on. If this flag is set, all incoming characters are echoed back to the sender.

The last flag, (W), indicates that window lock is on. If this flag is set and the local screen size is smaller than the remote system's screen size, the local window will not scroll right or left automatically to keep the cursor on the screen.

Common Keypresses

In most cases, except where noted later, **FCTN 9** will return the user to the previous function. Geneve users may also use the [ESC] key for the same purpose in many cases. **FCTN =** will offer the user the option of quitting. If "no" is selected, the user will be returned to the current function. **FCTN 0** will send the user to the main menu. For help, press **FCTN 7**. If help is available, this key is active, and options may now be selected from within the help screens.

All menus in TELCO follow the same pattern for ease of option selection. There are two ways in which the user may select an option. The first is by using the arrow keys (**FCTN X** and **FCTN E**) to move the bar to the desired choice, then pressing **ENTER**. The second is by pressing the capital letter of the option.

Whenever the user is prompted by TELCO to enter a string, these keypresses are available for editing the string:

FCTN 1	Delete character at cursor
FCTN 2	Insert character at cursor
FCTN 3	Delete from cursor to end of line
FCTN 5	Clear input
FCTN S	Left Cursor
FCTN D	Right Cursor

In the Terminal mode, the following keypresses are available for quick access to functions:

FCTN 1	Autodialer
FCTN 2	Print Spooler toggle
FCTN 3	Window left
FCTN 4	Download files (Pgdn on Geneve)
FCTN 5	Window right
FCTN 6	Upload (Pgup on Geneve)
FCTN 7	Help
FCTN 8	Review buffer
FCTN H	Hangup
FCTN M	Macro select
FCTN B	Terminal setup options
FCTN Y	Screen setup options
FCTN N	Full/Half duplex toggle
FCTN L	Log open/close

FCTN /	Log hold
FCTN J	Window lock toggle
FCTN V	Status line toggle
FCTN .	(function-period) Conference Mode
FCTN Q	Reset clock
CTRL 2	Clear screen locally

The list of available keypresses may be called up by pressing the help key (**FCTN 7**).

Note: In conference mode, screen controls such as cursor up or down are not transmitted to the remote system.

Terminal Emulations

Most keyboard and screen functions operate in the same manner with all of the terminals. In some cases, a special function has been included for keymapping, etc.

The emulations currently available are:

ANSI Graphics driver used with IBM
D410 Data General 410
ADM3A Common terminal in TI community
VT100 (Limited implementation this release, but improved over TELCO 1.3)
VT52 (Limited implementation this release)
HP2392 Hewlett Packard 2392 (limited implementation this release)

MAIN MENU

Most of the functions of TELCO are menu driven. The main menu provides access to these functions. The main menu has the following options:

Terminal
Review buffer
Auto dialer
Manual pc pursuit
Upload file
Download file
Catalog
Editor
Log open/close
Setup options
Intro screen
Quit program

Terminal

This function will load the current terminal. The default terminal is the ANSI terminal unless this is changed by the user or set up by the dialer.

Review Buffer

This function allows the user to page back through the last 8K of received data. The review buffer mode is very powerful, and is not destroyed by file transfers. The following keypresses are available to use:

FCTN 1	Top of review buffer
FCTN 2	Bottom of review buffer
FCTN 3	Window left
FCTN 4	Window down
FCTN 5	Window right
FCTN 6	Window up
FCTN 7	Help
FCTN 8	Screen Dump to a device
FCTN P	Purge review buffer
FCTN E	Line up
FCTN X	Line down
FCTN S	Column left
FCTN D	Column right

The screen dump function will dump a 23- or 24-line by 80-column copy of the screen in view, depending on whether or not the status line is visible. This function may be used with the log open and print spool active. A screen may be dumped to any device.

Auto Dialer

The dialer uses a file called TOS/PHONE on disk to keep a record of up to 99 numbers to dial. This file is not on disk when you receive the package. TELCO will create the file automatically, and does not create 99 numbers until the user pages down that far.

To view the list of numbers use:

FCTN 4	Page down
FCTN 6	Page up
FCTN X	Line down
FCTN E	Line up

To modify an entry, press 'M', then enter the number of the entry you wish to modify. The following information must be entered for a dial record.

A) Name: Title or name of BBS
B) Phone: Up to 14 characters for phone number
C) Term/mod: If connected to BBS load this terminal or module (name of module must be typed in by user)
D) Baudrate: Will set system to desired baud rate before dialing
E) Parity: Will set system to desired parity before dialing
F) Duplex: Will set system to full or half duplex after dialing
G) Width: Terminal width for BBS you are calling.

Dialing

To dial a number, type the number of the entry and press [Enter]. To dial more than one number in the directory, type the numbers, separated with spaces, and press [Enter]. TELCO will dial a list of up to 15 numbers (as many as fit on the line) until it connects on one of the numbers.

EXAMPLE: to dial numbers 1, 15 and 63 enter: "1 16 63".

While dialing, pressing **FCTN 4** will remove the current number from the redial list. The space bar will abort to the next number in the redial list. **FCTN 9** will abort the entire dial and return to the Dialer menu.

For PC Pursuit dialing, autodial modem users may link to the PC Pursuit dialer with the Term/mod option. Set up a dialer entry for the local Telenet node, with "PCDIAL" under Term/mod. The rest of the procedure is the same as for those using the manual PC Pursuit dialing option.

Manual PC Pursuit

Once you are connected to Telenet, you may redial a city with this dialer. You may have up to 99 cities in the dialing directory, and as many as 15 may be redialed. The list of numbers may be viewed with the same commands as the autodialer, and the entries are modified in a similar fashion. The following information must be entered for each dial record:

Name: Name of city
PCP: City i.d. string

Once TELCO is connected to a city, the autodialer will be reloaded, and the redial feature may be applied to any BBS within dial range of the PC Pursuit node. It is not necessary to have an autodial modem to use this feature.

File Transfer

The transfer protocols currently available are ASCII, Xmodem, Ymodem and CompuServe B. The Xmodem, Ymodem, and CompuServe B protocols send files with the header Paul Charlton designed unless the file is a display/fixed 128 unprotected file. They will receive files with or without the header, and any file received without the header will be in display/fixed 128 format. TELCO's CompuServe B protocol supports both binary and ASCII transfers, and will work with the Quick B option on CompuServe. If the file being received does not have a header, the number of bytes to receive will display "unknown" both for the total byte count and the estimated transfer time. The default error checking method is CRC. This error checking scheme is one of the most reliable ones. If the remote system fails to understand a request for CRC, TELCO will switch to Checksum. This process will take about 1 minute and the program will usually display 5 errors.

To abort a transfer in progress, press **FCTN 9**. This will abort both the local and the remote system in most cases. (Aborting a transfer during uploading may be an exception, since the request to abort the transfer may be taken in as data by the remote system. In such cases, the transfer would be aborted locally, but not at the remote system.) By default, aborted download files will be erased from your diskette. You may elect to keep aborted downloads through the file transfer setup.

The ASCII transfer is used to send any display/variable 80 text file to the remote system. TELCO has various options which allow the user to control how fast a file is transmitted, or to send line-by-line. Carriage returns (CR) and linefeeds (LF) may be removed or altered to make the file more readable to the remote system, leading spaces can be stripped, and at the end of the line, it is possible to add CR, CR/LF, LF, a space, or nothing. If the local echo is on, the file will scroll by on the screen. The screen will be restored after the transfer is complete.

File transfer options on the main menu have been divided into "Upload file" and "Download file" for faster access.

Catalog

The catalog function provides the user with the ability to delete, protect, unprotect and view files on a diskette. Once the user has selected the drive to catalog, the manager will build the directory. Once it is built, the display will show the diskname, free sectors, used sectors and number of files followed by up to ten file descriptions.

Although this manager will support several functions, only one function may be performed per catalog. This is done by marking files. To mark a file, move the bar to the file and press 'M'. To unmark, press 'U'. It is possible to mark all the files on the diskette by pressing 'A'. To clear all of the marks, press 'C'.

After marking all the files to be altered, pressing the 'P' key will allow the user to proceed to the functions menu. The functions available are:

Delete
Protect
Unprotect
Abort

All marked files will be affected by the function selected.

To view a D/V 80 file, move the highlighted bar to the desired file and press "V". While viewing a file, any key will pause the display, and any key will restart it. **FCTN 9** will abort the view.

Editor

The editor may be used to edit or create small D/V 80 files (e.g. messages) to save to disk. The limit is 50 lines by 80 columns. The following keys are available:

FCTN D	Cursor Right CTRL-A Clear All Tabs
FCTN S	Cursor Left CTRL-B Set Bell
FCTN E	Cursor Up CTRL-C Clear Tab
FCTN X	Cursor Down CTRL-P Place Tab
FCTN 1	Delete a Character CTRL-R Set Right Margin
FCTN 2	Insert a Character CTRL-S Show Tab Line
FCTN 3	Delete a line CTRL-T Tab
FCTN 8	Insert a line

The Set Bell, Clear Tab, Place Tab, and Set Right Margin keypresses act on the current cursor position, and the Bell sounded is the sound effect currently in use. These settings are saved in the Config file if save Changes is selected after exiting the editor.

Log Open/Close

This feature will trap incoming data and dump it to a device. While the user is in a terminal, this function may also be accessed directly by pressing **FCTN L**. Unwanted text may be filtered out with the log hold function. The log hold may be toggled on and off with **FCTN /**. *Note*: In order to prevent the log file from becoming excessively large, after approximately 18K of data is dumped to disk, TELCO closes the log file and opens a new one with the last character of the filename incremented by 1. This is the default setting for the log feature. This feature may be turned off in the Terminal setup menu, so that a continuous log is kept. By default, multiple blank lines are compressed to one blank line in both the log and the review buffer. If multiple blank lines must be preserved, the "Squeeze blank lines" option in Terminal Setup should be set to "OFF".

Setup Options

The setup options menu provides the user with easy access to the TELCO operations which may be altered. The setup options menu consists of:

Terminal setup
Screen setup
Hardware setup
Modem setup
File transfer setup
Edit macros
Save Changes

TERMINAL SETUP

The following options are available in the terminal setup menu:

A) Emulation mode: ANSI
B) Terminal width: 40
C) cr translation in: CR
D) cr translation out: CR
E) Destructive Backspace: Off
F) Line wrap: Auto
G) Screen scroll: Auto
H) Screen wrap: Off
I) Duplex toggle: Full
J) Remote echo: Off
K) Spooler mode: Delay
L) Xmit on/off mode: Logic

Disk 101. Contents of file TELCOD2

M) Xmit on chr :017
N) Xmit off chr :019
O) Log mode :Block
P) Squeeze blank lines :On
Q) Baud rate :1200
R) Parity :8N1

Emulation Mode

Currently there are five emulation modes available. These are:

ADM3A
ANSI
D410
HP2392
VT52
VT100

More information about these emulations is at the end of this document.

Terminal Width

The terminal width may be any value from 20 to 80. This value represents the screen size of the remote system. This allows the TI-99/4A user to view an 80-column screen through a scrolling window.

CR Translation In

If this toggle is set to CR then all incoming CRs are left alone. If it is set to CR/LF then all incoming CRs are translated in CR/LF sequences.

CR Translation Out

If this toggle is set to CR, a one-character CR is sent when [Enter] is pressed. If it is set to CR/LF, a CR/LF sequence is sent.

Destructive Backspace

Some remote systems treat all backspaces received as destructive backspaces, while others simply move one position to the left without erasing the character, relying on the terminal to erase characters which are backspaced over. The destructive backspace option should be set to "On" for the latter type, if you wish to erase as you backspace.

Line Wrap

If line wrap is set to AUTO, all lines which are longer than the terminal screen width are automatically continued onto the next line. If it is set OFF, any data beyond the terminal screen width is truncated.

Screen Scroll

If screen scroll is set to AUTO, screen scrolling is automatic when the cursor exceeds the bottom line. If it is set to OFF, the cursor will stay at the bottom line, causing data to be overwritten.

Screen Wrap

If the Screen scroll setting is "off", this option will allow the cursor to wrap through the bottom of the screen to the top, and vice versa if Screen Wrap is set "on". Some terminals use this option as a quick way to move the cursor from the bottom to the top of the screen.

Duplex Toggle

This toggle will switch the system between full and half duplex. If TELCO is set to half duplex, all characters transmitted will be echoed back to the user locally. If it is set to Full duplex, TELCO assumes that the remote system will echo the user's transmissions. The duplex may also be toggled directly from the keyboard with **FCTN N**.

Remote Echo

If this feature is turned on, all received data will be echoed back to the sender.

Spooler Mode

The spooler mode may be toggled between "delay" and "imm" (immediate). In Delay mode, data is not sent to the printer until it scrolls off the screen, allowing full-screen graphics to be spooled to the printer.

Xmit On/Off Mode

This may be toggled between "Logic" and "Single". Logic mode means that TELCO will attempt to send Xoff until transmission stops, up to ten times. If it is set to single, TELCO will send only one Xoff.

Xmit On Chr

This is the decimal value of the character that is transmitted for Xon. Usually this is control-q, or 17.

Xmit Off Chr

This is the decimal value of the character that is transmitted for Xoff. Usually this is control-s, or 019.

Log Mode

This is toggled between "Block" and "Cont" (Continuous). In Block mode, the log will close approximately every 18K of received data, and open a new log with the filename incremented.

Squeeze Blank Lines

By default, TELCO will compress multiple blank lines received to one blank line in the review buffer and in the log. If this option is set to "Off", multiple blank lines will be preserved as received.

Baud Rate

The baud rate options allow the user to select from 5 different preset baud rates:

300bps
1200bps
2400bps
4800bps
9600bps

There is also an "other" option, which allows the user to enter any baud rate supported by the modem in use.

Parity

The default setting for TELCO is 8 data bits, no parity check and 1 stop bit, which is often written "8N1". This option allows the user to select from the three most common parity settings of 8N1, 7E1, and 7O1, or build any combination from: 7 or 8 data bits; Even, Odd, or No parity; and 1 or 2 stop bits.

SCREEN SETUP

This menu provides the user with the following options:

A) foreground color
B) background color
C) menu inverse toggle
D) 40/80 Display (Geneve or 80-column card ONLY)
E) status line toggle
F) sound effects: Beep
G) alarm sound: Chime
H) window width: 40
I) left column: 00

Menu Inverse Toggle

This toggle will switch all menus and windows used in TELCO from normal to inverse display or back again.

40/80 Display

If TELCO is being used on a Geneve 9640, or with an 80-column card, the user may select an 80-column display. This toggle switches between 40 and 80 columns. To have TELCO boot into 80 columns, simply save the changes to disk at the setup menu. On a 99/4A without an 80-column card, this option will have no effect.

Status Line Toggle

This toggle switches the terminal height from 23 to 24 lines and back. Note: in 24-line mode the status line will disappear. For Geneve or 80-column card users, this option changes the interlace and switches between 24- and 26-line mode. Geneve users should use this option instead of Ctrl-Alt-Shift, since TELCO will save the status line position properly only if **FCTN v** or the menu option has been used.

Sound Effects

TELCO provides the user with three options for sound effects when a bell character is received: none (sound off), chime or beep. The default is beep. The bell character may vary depending on the emulation mode being used.

Alarm Sound

The alarm is sounded upon successful completion of file transfers and successful completion of a dial. It may be set to any of Chime, Beep, or Off. You may find it useful to set it to a different sound from the bell character sound effect, or it may be set to the same sound.

Window Width

This function allows the user to specify the size of the viewing window. The terminal screen that is being emulated will be slid through this window. Note that the maximum window width on a TI-99/4A is 40 columns, and on a Geneve in 80-column mode or on a TI-99/4A with an 80-column card, it is 80. The 40/80 Display toggle will adjust the maximum value automatically.

Left Column Shift

This feature allows the user to shift the left edge of the terminal "window" to correct problems with monitors or televisions. Note that the window width plus the left column shift should not exceed a total of 40 on a TI-99/4A, or 80 on the Geneve in 80-column mode.

HARDWARE SETUP

The following options are available:

Spooler port
Baud Rate
Parity
Modem port
Hangup type

Spooler Port

Print spooler support is available for the TI, CorComp, and Myarc PIO ports 1 and 2, as well as RS232 ports 1 through 4. Additional hardware support may be added if demand from registered users warrants. The default setting for this option is "no printer", which will prevent TELCO from locking up if the spooler option is accidentally selected when no printer is attached to the system. Otherwise, TELCO may take up to 10 seconds to determine that no printer is in use. Since some terminals support remote requests for printer on/off, this setting should be left at "no printer" unless a printer is in use. If the PIO port is in use, the Baud Rate and Parity options, used for the RS232 ports, will instead display N/A. For the RS232 port, the same Baud Rate and Parity settings are available as in the terminals. A 512 character buffer has been added to the print spooler for additional speed.

Modem Port

The modem may be on port 1, 2, 3 or 4.

Hangup Type

TELCO will support a modified cable. This cable arrangement will allow the user to utilize the DTR line hangup option. Connect the DTR line from your modem to the CTS (Clear To Send) line of the TI RS232 port being used (pin 5 for ports 1 & 3 or pin 13 for 2 & 4). Check to make sure that the dip switch for the DTR line on the modem is set so that the TR light is off if your TI is off. These pinouts should be correct for Hayes-compatible modems, but check your modem manual to verify this.

Modem		*RS232:*	*Ports 1&3*	*Ports 2&4*
Ground	1	Ground	1	1
Transmit	2	Receive	3	14
Receive	3	Transmit	2	16
Ground	7	Ground	7	7
DTR	20	CTS	5	13

The other hangup method does not require cable modification but instead uses the hangup string (see Modem Setup). This method is not, however, as reliable as the DTR line method. The default string setting is for Hayes compatible modems. The Myarc RS232 card cannot support the DTR hangup method, so Myarc users must select the "string" method.

FILE TRANSFER SETUP

The following options are available:

A) Aborted downloads: Discard
B) Default error check: CRC
C) Echo locally: Off
D) Blank lines: Off
E) Character pacing: 00
F) Line pacing: 00
G) Pace character: 000
H) Strip leading space: Off
I) Line by line send: Off
J) Send at end of line: CR
K) CR translation: None
L) LF translation: None

Aborted Downloads

When a download is aborted, the system will either discard the partial file or, if this toggle is set to Save, the file will be kept.

Default Error Check

The default error checking with Xmodem transfers set by TELCO is CRC. This is the best method for error checking. The Xmodem transfer protocol will switch to Checksum in about one minute if the remote system still can not understand which error checking protocol is in use. Switch the default to Checksum if you know that the remote system is not capable of using CRC.

Echo Locally

When an ASCII file is being sent, the ASCII transfer protocol may be set to echo what it is sending back to the user. Note that echoed data while an ASCII transfer is in progress will not be put into the review buffer, nor will it be formatted in any way except for CR/LF characters. On the TI-99/4A, the terminal screen width will be ignored - all data is displayed in 40 columns. On the Geneve, all data is displayed in either 40 or 80 columns, depending on your screen setting.

Blank Lines

Some systems will assume a CR/LF CR/LF sequence (i.e. a blank line) to be an end of text marker. If this toggle is set to Expand, a CR/LF (space) CR/LF sequence will be sent for blank lines.

Character Pacing

This is the pause time between characters transmitted during an ASCII transfer in 60ths of a second. This may be used for ASCII transfers to systems that are unable to receive characters at the full speed of the transmission.

Line Pacing

This is the pause time between lines sent during an ASCII transfer in 60ths of a second and is used for reasons similar to those for using character pacing.

Pace Character

Some systems send a character after each line to indicate that the system is ready for the next line. To have the ASCII transfer do this, set the pace character to the appropriate value. If it is not required, set the pace character to zero.

Strip Leading Space

If this option is selected, during an ASCII send all leading spaces will be stripped before transmission.

Line by Line Send

If this is turned on, when an ASCII upload is started, the first line of the file will be sent, and all subsequent lines will be sent one at a time as the spacebar is pressed.

Send at End of Line

This option allows the user to select what will be sent at the end of each display/variable 80 record. If this option is set to nothing, nothing will be transmitted. It may be set to send nothing, cr, cr/lf, lf only, or a space.

CR Translation

When an ASCII file is being sent, a CR may be translated into a CR/LF sequence or even STRIPped from the file.

LF Translation

When an ASCII file is being sent, a LF may be translated into a CR/LF sequence or even STRIPped from the file.

MODEM SETUP

Strings: All strings and macros that are transmitted may take advantage of special characters. These are:

!	Transmits a carriage return (use !! to send "!")
~	Pauses transmission for 1 second
*#	Will repeat character immediately prior to the asterisk the number of times specified by #
^	Subtracts 64 from the ASCII value of character which follows ^ symbol. e.g. ^A sends a CTRL-A or ASCII 1.
|	Sends Character number 27, the escape character
^j	Replaces the "*" in strings (e.g. *70 should be ^j70 to disable call waiting in some areas.)

The following options are available:

Initialization string
Dial string
Hangup string
Abort redial chr :!
Modem echo time :120
Redial pause time:120

Initialization String

This string will be sent to your modem whenever the program is booted.

Dial String

This string is sent to the modem by the dialer as a prefix to the phone number string within each dial record.

Hangup String

If the hangup type (see Hardware setup) is set for "string", this string will be used to hang up the modem.

Abort Redial Character

Some modems may require a special character for aborting a redial. Usually the default '!' (carriage return) will work.

Modem Echo Time

Some modems may echo the number being dialed back to the user quite slowly. On occasion, it is too slow for the dial to react properly. If this happens, lengthen the modem echo time until the dialer is acting properly.

Redial Pause Time

This is the time period between redials. Some modems require a pause to settle down.

Note: The preceding settings are suitable for Hayes-compatible modems. Other modems may require some experimentation to find the best settings. For the Volksmodem 12, for example, the abort character is "A", and the modem echo time should be set to 280.

EDIT MACROS

Up to 26 36-character macros are available. When a macro is used, the data is transmitted to the remote system. For viewing the available macros, the following keys are active:

FCTN 4	Page Down
FCTN 6	Page Up
FCTN X	Line Down
FCTN E	Line Up

To edit a macro, select the corresponding letter (A-Z) (*Note*: The macro must be on screen before it can be edited.) The cursor is placed beside the letter of the macro to be edited, ready for the user to enter the string. The string is accepted once the user presses [Enter]. The same special characters that are used in the modem setup strings are available for use in macros. A macro to log a user on to a board may be set up, for example, by using the pause and carriage return characters (~ and !) with the text.

An additional feature of macros is the ability to link more than one macro together. Use an "&" (ampersand) at the end of the macro, followed by the letter of the macro to link to. While a bug in earlier releases of TELCO prevented this from working properly, this is corrected in V 2.0.

e.g. John Doe&B will link to macro B.

Note: While it is possible to link a macro to itself, this is unwise, as the macro will be in a permanent loop. It is possible to abort the execution of a macro by pressing **FCTN 9**, but this will not break out of a looping macro, nor will it break a larger loop of linking macros.

The maximum length of a macro string is 36 characters. If long macros are required, however, the multiplication feature (*#) and macro link feature can help to defeat this limitation.

To invoke a macro from the terminal, press **FCTN m**, followed by the letter of the macro desired.

CAUTION: After editing any of the setup options or macros, be sure to select "save Changes" in the Setup options menu if you do not want your new settings or macros to be lost on exiting TELCO.

TERMINAL EMULATION MODES

ADM3A Terminal

The following incoming ADM3A commands are processed:

ESC =rc	Row/Column position
CTRL G	Bell
CTRL H	Backspace
CTRL J	Linefeed
CTRL K	Cursor Up
CTRL L	Cursor Right
CTRL M	Carriage Return
CTRL Z	Home and Clear

The following TI keys transmit special functions:

FCTN E	Cursor Up
FCTN D	Cursor Right

ANSI Terminal

The following incoming ANSI commands are processed:

ESC[pl;pc H	Cursor Position
ESC[pl;pc F	Horizontal Position
ESC[pn A	Cursor Up
ESC[pn B	Cursor Down
ESC[pn C	Cursor Forward
ESC[pn D	Cursor Backward
ESC[6 n	Device Status Report (sends RCP sequence)
ESC[s	Save Cursor Position
ESC[u	Restore Cursor Position
ESC[2 J	Erase Display
ESC[K	Erase Line
CTRL G	Bell
CTRL H	Backspace
CTRL J	ine Feed
CTRL L	Clear screen and Home cursor
CTRL M	Carriage Return

The following TI keys transmit special functions

FCTN S	Backspace
FCTN D	ESC[C
FCTN E	ESC[A
FCTN X	ESC[B

D410 Terminal

The following incoming commands are processed:

CTRL G	Bell
CTRL H	Home cursor
CTRL J	Linefeed
CTRL K	Delete from cursor to end of line
CTRL L	Clear screen
CTRL M	Carriage Return
CTRL P r c	Cursor position (row/column)
CTRL W	Cursor Up
CTRL X	Cursor Right
CTRL Y	Cursor Left
CTRL Z	Cursor Down
>1E,>46,>45	Clear screen
>1E,>46,>47	Home cursor

The following TI keys transmit special codes:

ENTER	>0A
FCTN E	CTRL-W
FCTN X	CTRL-Z
FCTN S	CTRL-Y
FCTN D	CTRL-X

Keyboard mapping feature for the D410:

The D410 terminal uses a 2-byte sequence for the special function keys. To simulate the function keys, press **FCTN K** followed by the key whose ASCII value is the same as the second byte in the sequence desired. e.g. for the sequence equivalent to F1 you would type **FCTN K Q**

HP2392 Terminal

The following incoming commands are processed:

ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Right
ESC D	Cursor Left
ESC H	Home cursor
ESC F	Home cursor Down
ESC G	Move cursor to left margin
ESC K	Erase from cursor to end of line
ESC J	Erase from cursor to end of screen
ESC &acrY	Cursor positioning

To transmit the codes which represent the function keys on the HP2392, press **FCTN K**, then one of the following keys:

1	F1
2	F2
3	F3
4	F4
etc.	

VT52 Terminal

The following incoming commands are processed:

ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Right
ESC D	Cursor Left
ESC H	Home Cursor
ESC Y r;c	Direct Cursor Addressing
ESC K	Erase from cursor to end of line
ESC J	Erase from cursor to end of screen

To transmit the codes which represent the VT52 keypad and function keys, press **FCTN K**, then one of the letters below:

0-9	Numerics
-	Dash
,	Comma
.	Period
Enter	Enter
A	PF1
B	PF2
C	PF3
D	PF4

VT100 Terminal

The following incoming commands are processed:

ESC[n A	Cursor Up
ESC[n B	Cursor Down
ESC[n C	Cursor Right
ESC[n D	Cursor Left
ESC[r;c H	Direct Cursor Addressing
ESC[r;c f	"
ESC[H	Home
ESC[f	"
ESC[K	Erase from cursor to end of line
ESC[0J	Erase from cursor to end of screen
ESC[2J	Clear Screen
ESC[?2l	Enter VT52 mode (loads VT52 terminal)
ESC[?5i	Printer on (spooler)
ESC[?4i	Printer off (spooler)

To transmit the codes which represent the VT100 keypad and function keys, press **FCTN K**, then one of the letters below:

0-9	Numerics
-	Dash
,	Comma
.	Period
Enter	Enter
A	PF1
B	PF2
C	PF3
D	PF4

A SPECIAL NOTE ABOUT PARITY

Since TELCO offers a truer implementation of the 8N1 protocol in the ANSI terminal emulation, 8N1 will not work on all boards in this emulation. Some programs strip the 8th bit from incoming data, but since TELCO is able to interpret a subset of IBM graphics, it requires the more complete implementation of the protocol. Currently, TELCO strips the 8th bit in all terminals except ANSI. It is always a good idea to use the correct parity for the board you are calling. If you seem to get random characters, or only some of the characters when you call a board, this indicates that your system is not set to the correct parity. Change your parity to match that of the system you are calling, and all will be well. During Xmodem transfers, you may notice that parity switches to 8N1 even if you are on a 7E1 or 7O1 board. Xmodem is an 8-bit binary transfer protocol, so both systems switch to 8N1, then back to their original settings after the transfer is complete. TIBBS and Techie BBSs usually run at 7E1 or 7O1, while TEXLINK BBS runs at 8N1.

EXAMPLES

SETTING UP

The following is a step-by-step example of how to set up TELCO for the following system: TI-99/4A with CorComp RS232 card; Volksmodem 12 modem; pulse telephone line; a string for hangup; line-by-line ASCII uploads which send a space at the end of each line transmitted, using character pacing; Xmodem transfers with the option of keeping incomplete downloads turned on; a 38-column screen offset 2 columns towards the right, print spooler changed to immediate; and the log to disk closing every 18K, incrementing the file name: (Note: These are not necessarily recommended settings, only examples of how to change settings. In most cases, the defaults will probably be adequate. To set your system up differently, simply follow the example below, making different choices.)

1. From the main menu, select "Setup options", either by moving the highlighted bar to the option and pressing [Enter], or by pressing the capital letter "S".

2. Select Terminal Setup.

3. Select Emulation mode, and choose option 1, ADM3A.

4. Select Terminal width, and type 38. Press [Enter].

5. Toggle the Spooler mode by pressing "K" or by pressing [Enter] while the bar is over that option, until it reads "imm" in the right-hand column.

6. Toggle the Log mode to "block" by pressing "O", or by pressing [Enter] while the option is highlighted.

7. Press **FCTN 9** to return to the setup menu.

8. Select Screen setup.

9. Select window width and type "38".

10. Select left column, and type "2".

11. Return to the setup menu with **FCTN 9**, and select hardware setup.

12. Select Hangup type, and toggle it to "String".

13. Select spooler port with the highlighted bar, and press [Enter]. Move the highlighted bar to "cc pio/1" and press [Enter].

14. Return to the setup menu, and select modem setup.

15. Select Dial string, and type "ATDP" in the window. Press [Enter].

16. Select Abort redial chr, and enter "A".

17. Select Modem echo time and enter 280. Return to the setup menu.

18. Select File transfer setup.

19. Select A, Aborted downloads, and toggle it to "KEEP".

20. Select I, and toggle line by line ON.

21. Select J, and choose option e, space.

22. Select E, Character pacing, and type 10. Press [Enter].

23. Be sure to select "C" for "save Changes" before exiting TELCO.

DIALER ENTRIES

The following is a step-by-step example of adding a new entry to the autodialer for a board with the name TEXLINK, and the number 999-9999, which requires a parity of 7E1, half duplex, and an ADM3A terminal emulation with a baud rate of 1200, using a terminal simulation of 80 columns.

1. Go to the autodialer, either directly from the terminal by pressing **FCTN 1**, or from the main menu.

2. Press M, then type the number of the entry that you wish to modify and press [Enter].

3. Select "A", then type TEXLINK and press [Enter].

4. Select "B" then type the number 999-9999, with or without the dash, depending on your modem, and press [Enter].

5. Select "C", then type ADM3A and press [Enter].

6. Select "D", then "B" to select 1200 bps.

7. Select "E", and the parity menu will appear. Select 7E1 and press [Enter].

8. Select "G" then enter the terminal width desired for this entry.

Any of the above steps may be skipped if a particular entry (e.g. baud rate) is already correct.

The editing keys are available for use in making entries to the autodialer. They are:

FCTN 1	Delete character at cursor
FCTN 2	Insert character at cursor
FCTN 3	Delete from cursor to end of line
FCTN 5	Clear input
FCTN S	Left cursor
FCTN D	Right cursor

SAMPLE MACROS

A) !~*10USER NAME!~PASSWORD!
B) This is neat!!*&E
C) |N5
D) |O
E) LOOK at tt ^ Hhis!!
F)
G)

Example A, above, will send a carriage return (!), followed by a pause times 10 (~*10), followed by the string "USER NAME", then a carriage return (!), a one-second pause (~), and finally the string "PASSWORD", ending with a carriage return (!). This type of macro is useful for logging onto a board where the pause between system prompts can be accurately anticipated. It may take some experimentation to create a workable macro for such conditions. Once it is set up, however, logging on to a particular board is as simple as pressing **FCTN M** and then pressing "A".

Example B will simply send the string "This is neat!*" and branch to macro E. The two exclamation marks will send one exclamation mark rather than carriage returns, and the asterisk will appear normally because it is not followed by a numeric value. The "&E" tells TELCO to do macro E after completing B.

Example E, which may be called separately, and will always be sent after B, transmits the string "LOOK at this!", by using control-H (^ H) to backspace over the extra t.

Macros C and D send escape sequences. These two macros may be used to set an Epson-compatible printer to skip perforations and back if the following method is used:

1. Set your system to HALF DUPLEX (**FCTN N**).
2. Set your print spooler to IMMEDIATE, with the printer turned ON.
3. Open the print spooler.
4. Press **FCTN M**, then press "C", and your printer will be set to skip perforations.

Following the same sequence, but with macro D, will set it back. Check your printer manual for escape codes that you can use this way.

ENTERING MACROS

The letter of the macro to be edited must appear on screen. You may page up and down in the macro list, or scroll up or down one line at a time, with **FCTN 6**, **FCTN 4**, **FCTN X**, and **FCTN E** respectively. Choose the letter of the macro to edit, and your cursor will be placed ready for you to edit the string. The maximum length of a single macro is 36 characters, which may be used most efficiently by using the multiplication (*#) wherever feasible. Chaining macros is also helpful. The editing keys are also available here as in the dialer. Once a macro has been edited to your satisfaction, press [Enter]. If you wish to leave a macro unchanged, press **FCTN 9** to abort the edit. After editing Macros, be sure to save the changes to disk from the setup menu.

TI AND GENEVE KEYPRESS TABLE

ASCII (Decimal)	*TI Keypress*	*Geneve Keypress*
0	Ctrl-, (comma)	Ctrl-, (comma)
1-26	Ctrl-a to z	Ctrl-a to z
27	Ctrl-. (period)	Escape
28	Ctrl-;	Ctrl-;
29	Ctrl-=	Ctrl-=
30	Ctrl-8	Ctrl-8
31	Ctrl-9	Ctrl-9
32-126	Normal	Normal
(i.e. space, alphabetic chars, etc.)		

Disk 102. TASS2001

Version: 3.00 *Author*: Gary Bowser
Requires: EA or XB *Language*: AL *Updated*: 3/14/88

"Tri Artist Slide Show" program is an extremely complete, powerful, and fast program for creating slide shows. Ability to load one picture while viewing another, can use RLE, TI-Artist, and GRAPHX files. Excellent documentation. Available by special arrangement with author.

```
dskdir. v2.0. 12-dec-96
Disk name               = RAM_DISK
Sectors total           = 360
Sectors used            = 220
Sectors available       = 138
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 LOAD          4 PROGRAM      >022 003
002 >003 P2001-LOAD    8 PROGRAM      >025 007
003 >004 TASS v3.00   97 PROGRAM    Y >02c 096
004 >005 TASSDOCS    111 DIS/VAR 80   >08c 110
```

Disk 102. Contents of file TASSDOCS

TASS 2001 TRI ARTIST SLIDE SHOW
Version 3.00.88
User's Guide

LICENSE FOR ALL PHOENIX 2001 SERIES SOFTWARE FROM GARY BOWSER

No version of any of the Phoenix 2001 series of software is public domain software, nor, is it free software. All the Phoenix 2001 series of software are user-supported software, as explained in the following license, for this version 3.00.88 of TASS2001 which is part of the Phoenix 2001 series of software from Gary Bowser.

Non-registered users are granted a limited license to use TASS2001 on a trial basis for the purpose of determining whether or not TASS2001 is suitable for their needs. Use of TASS2001, except for this purpose, requires registration. Use of non-registered copies of TASS2001 by any person on a continuing basis or in a business or professional office is strictly forbidden.

Registration permits a user to use TASS2001 on a single computer; a registered user may use the program on a different computer, but may not use the program on more than one computer at the same time.

No user may modify TASS2001 in any way, including but not limited to decompiling, disassembling or otherwise reverse engineering the program.

All users are granted a limited license and are encouraged to copy TASS2001 for the trial use of others subject to the above limitations, and provided the following conditions are met:

1. TASS2001 must be copied in unmodified form.
2. The documentation must be included.
3. No fee, charge or other compensation may be accepted or requested by any licensee.
4. TASS2001 may not be distributed in conjunction with any other product.

Sysops of BBS's are encouraged to make TASS2001 available for downloading provided the above conditions have been met. No individual or organization is permitted to collect the registration fee for the author unless express written permission is obtained and that permission is displayed to those paying the registration fee in such a manner.

A TASS2001 registration licenses you to use TASS2001 on a regular basis. Registration includes mailed notification of the next update, and further updates at the author's discretion.

The registration fee is $15.00. To order copies of TASS2001, register, or obtain written permission, contact:

Gary Bowser
432 Jarvis St. #502
Toronto , Ontario
Canada, M4Y 2H3

INTRODUCTION

The goal of this line of software (Phoenix 2001) is to try to make the best software ever for the TI99/4A and GENEVE and all future 9900 based computer systems, and try to keep the TI99 alive pass year 2001. (The reason for the number 2001 at the end of all the software). The `Phoenix' part comes from the idea of the fabled bird the `Phoenix' which would be reborn from it's own ashes, which I think was like the TI99, which got burned when TI decided to stop making TI99 systems, but the TI99 spirit was so great we can I guess almost survive anything, and in fact we have been reborn with lots of new hardware/software from dozens of third-party companies around the world.

This program "TASS2001" is the first step in a long line of new software from me or my future company. I hope also to start another line of software called "Phoenix 2010" which will be aimed at the super expanded TI'ers like ones with Geneves, 512K, etc. And hopefully be good enough to push them into more new hardware and to hold onto their TI's past year 2010.

TASS2001 is for making a slideshow out of your TI-ARTIST, GRAPHX, and DRAW-A-BIT II pictures. There is no need to make special disks or command files because TASS is fully automatic, but yet with dozens of great features, like Forward, Back, controllable delay per disk drive, to allow ramdisks to be the same speed as a floppy, or for making the pictures stay on a little longer, etc. It is also the only slide show program that I know of, which can load in the next picture from disk without either blanking the current screen, or giving the user a slow downward display of the new one as it loads in, since the TASS2001 uses new ways of reading info from disks, it will load a picture very quickly, plus it will keep the old picture on the screen until the loading of the new one is complete, and all delays are finished, it then pages in the new picture in a blink of a eye. Other little features are, screen dumps ,save/convert screen, left/right flips, up/down flips, plus many more little goodies.

HARDWARE REQUIREMENTS

TASS2001 has been designed to operate on both the TI-99/4A and the Geneve 9640. The minimum system required to use TASS2001 is: A TI-99/4A console with at least 32K Memory Expansion or a GENEVE Myarc 9640 and at least one SSSD (90K) disk drive. Plus one of the following modules, Editor/Assembler (EA),TI-Writer, Supercart Superspace, Extended Basic.

TASS can support any type of ramdisk, from Myarc to Horizon, plus it will in the near future support Hard disk drive access.

FILES THAT SHOULD BE ON THE `TASS2001' DISK

Filename	*Size*	*Comment*
LOAD	4	A quick Extended Basic file to load "P2001-LOAD"
P2001-LOAD	8	E/A #5 load file to load any Phoenix 2001 file
TASS v3.00	97	TASS2001 in P2001 format
TASSDOCS	111	This doc file you should be reading right now.

GETTING STARTED

To run TASS from the E/A environment: Select option #5 & type:

```
DSK1.P2001-LOAD
```

To run TASS from the TI-Writer environment: Select option #3 & type:

```
DSK1.P2001-LOAD
```

A program called LOAD has been provided to auto-load TASS when the Extended Basic cartridge is selected from the main TI menu. After the P2001-LOAD is file is loaded, either with Extended Basic or any E/A #5 loaders like Funnelweb, Horizon menu, etc. The program will display a Title Screen, with my address, etc. And on the top of the screen it will display "A P2001 file is loading....",it then will start searching for the "TASS v3.00" file starting at drive #5 (yep, drive 5,to allow ramdisk loading) until drive 9,if of course there is drives at these locations, after searching drive 5 to 9,if the file has not been find as yet, it will start searching from drive 1 to 4,after which if the file was not found it will display the error message "Error in Loading... Press FCTN-QUIT",if this happens press FCTN-QUIT (CTRL-ALT-DEL on GENEVE) and check your disk and/or ramdisks for the proper file, this error message will also happen if there is a bad sector on one of your floppies. Of course, most of the time the loader will find what it is looking for, and will then start loading in the big TASS file.

Note: If you have the hardware to edit or change your GROM 1 and 2 containing TI BASIC, you must disabled the hardware or have the normal TI BASIC gram files in memory, or the TASS2001 program will not run correctly, since it uses lots of GROM subprograms. This note only applies to owners with hardware like the GENEVE, MaxiMem, and other Gram devices. (On the GENEVE you must be in GPL mode with a version greater than V0.97, and not in MDOS mode. But you can run TASS2001 at any speed like #5.)

START-UP INSTRUCTIONS :

After the program is loaded it will display a title screen; press any key to begin. Now the program will display the startup screen with the cursor next to the words `Disk Order :'. Enter the order of the disk drives in your slide-show. Ex. 142 , would show drive 1, then drive 4, then drive 2.

Whenever TASS comes to a zero in the disk order, it will run a very fast graphic line demo program, which displays about 40 drawings before going to the next drive. All keys are still in operation when the demo is running.

Next, the cursor will be next to the words `Save Filename :'. This is the filename that will be used when saving a slide from the SAV SCR menu. Enter a different name if you wish.

Next, the cursor will be below the words `Print Devicename :'. In this version, the print routine supports the CANON PJ-1080A color printer and some other color printers; if you do not have a color printer, then just press enter. If you do have a color printer, then enter the correct device-name for your printer.

And, finally the cursor will be in the Disk Delay table, this is where you set your delays between slides, for each drive. The numbers show seconds in hexadecimal. Enter your delay for each drive. The number before the cursor is the drive number. The first one in the table is reserved for future use. Delay ranges from >00 (0 secs.) to >FE (254 secs.) and >FF is infinite delay.

Note: Whenever the cursor is on the screen, you have full-editing, INS, DEL, ERASE, and cursor forward, back movement.

MENU INSTRUCTIONS :

After the startup screen is done, or when the menu key is pressed, the `MENU' will be displayed on top of the slide in memory. You have these options :

Options	What you see on the menu.	
Name of program	TASS2001	
Version and year of prog.	V3.00.88	
B/W fore, background color	COLOR	17
Current disk drive number	DRIVE	0
Repeat disk order yes or no	REPEAT	Y
Magnify printing yes or no	MAG	N
The command being executed, if any		

FCTN	*description*	*name of key*
1,2,3	are not used	
4	Next disk drive	NEXT
5	Menu on/off	MENU
6	Picture on/off	PICTURE
7	not used	
8	Goes to start-up screen	RESET
9,0	are not used	
=	Quit to TI title screen	QUIT

CTRL	*description*	*name of key*
1	Change B/W color	COLOR
2	Change Repeat	REPEAT
3	Change Zoom/Mag	ZOOM
4	Up or down flip	U/D FLIP
5	Left or right flip	L/R FLIP
6	Print the picture	CANON
7	Go to the save menu	SAV SCR
8,9,0	are not used	

The words `REPEAT',`ZOOM/MAG' will be displayed at the bottom of the menu, when the keys `REPEAT',`ZOOM' are pressed. Then type `Y' for yes and `N' for no. To exit repress the first key. A version is in the works that will be able to zoom in on a part of the picture.

When the keys 1-F are pressed and the display at the bottom of the menu is blank. Then the screen color will change.

When the key 'COLOR' is pressed, the words `BW COLOR' is displayed. Then type in the fore, then background. The first digit in the menu is the fore. To exit repress the color key. The color will not change until either a color picture is displayed or the picture is toggled on and off with the `PICTURE' key.

Color table :	0 Screen color			
1 Black	4 Dark Blue	7 Cyan	A Dark Yellow	D Magenta
2 Medium Green	5 Light Blue	8 Medium Red	B Light Yellow	E Gray
3 Light Green	6 Dark Red	9 Light Red	C Dark Green	F White

When the `PICTURE' key is pressed, the picture being displayed will either be turned off or on. When the picture is turned on, the current color is erased and the current B/W color is used. The picture off feature is useful when the menu is hard to see due to a slide in the background.

The `RESET' allows you to go back to the start-up screen, to change disk order, save filename, print devicename, disk delay, then returns you back to exactly where you were in the menu.

If the current disk drive number is changed then when you exit the menu the slide-show will start at the beginning of the new disk. Ctrl 4 does an UP or DOWN flip. Just press it and the slide will be flipped up-side down. This will take about 2-3 seconds before you see the new slide.

Ctrl 5 does a LEFT or RIGHT flip. Just press it and the slide will be mirrored. You will see the new slide scroll down the screen very fast. Ctrl 6 does a color screen dump of the display to a color printer like the CANON PJ-1080A or Radio Shack CGP-220 (26-1268) printers. Ctrl 7 displays the SAVE screen menu. You will then have these options, any other key not listed will return you to the main MENU.

Options	*What you see on the menu*
Name of program	TASS2001
TI-ARTIST	1 ARTIST
GRAPHX	2 GRAPHX
Run Length Encoded (DF128)	3 RLE
	4
	5
4-8 are for new save	6 .....
formats in a future ver.	7 .....
	8 .....
Name of the menu screen	SAVE SCR

The ARTIST option saves just the Black/White part of the slide displayed, in a format compatible with the TI-ARTIST program. The GRAPHX option saves the slide displayed, in a format compatible with the GRAPHX program.

The RLE option saves the slide displayed, to a DIS/FIX128 Run Length Encoded format used a lot to transmit or store pictures between other computers. This option produces a 100% true RLE file not like MAX/RLE. When you press a key from 1-8 the phrase SAVING appears on the bottom of the menu then and TASS saves the slide on the screen, with the current SAVE filename entered at the start-up screen. After the file is saved it will auto-increment the filename as the examples below show.

SAVE name	*next name that will be used*
PICTURE	PICTURG
PICTURG	PICTURH
AZ	BA
BA	BB
ZZ	AA

Note: You can go back to the start-up screen and change the SAVE filename by using the RESET key from the main MENU. Also if any errors happen when saving a slide, it will display IO ERR y or ERROR, just press enter to go back to the main MENU. If it is a IO ERR y then the error type is listed in the TI *User's Reference Guide* on page III-11 to III-12. The first digit X of the I/O ERROR is always 6 in the SAV SCR section so it is not used.

RUNNING INSTRUCTIONS :

The program will start executing the slideshow, when the menu is turn off. You have some options when the slideshow is running.

Options :

FCTN 4	Next disk drive in order.
FCTN 7	Back one picture: When it reaches the beginning of the disk, it will start at the end of the previous disk in order. If there is no previous disk, then it will start at the last disk in order.
FCTN 5	MENU on/off
FCTN 8	Reset to beginning of disk.
FCTN 9	Forward one picture, cancels delay.
CTRL 4	Up or down flip
CTRL 5	Left or right flip
0	Pause the picture until another key is pressed.

Note: You must hold down the key until it takes effect because the program only scans the keyboard when not loading a picture. On the GENEVE, since there is a key buffer, there should be no need to hold down the key. In the next ver. I hope to have a software key buffer in the TASS program, so you would not have to hold down the key until it takes effect.

SPECIAL INSTRUCTIONS :

Because TASS remembers the DISKNAME the diskettes must have different names. If there is no disk in the drive or no pictures on the disk, then TASS will goto the next drive very quickly.

TASS automatically skips over any incompatible files. But due to its program structure, this example would be in error.

Disk catalog :

TARDIS_P	ARTIST	Would display as B/W
RICK1_C	ARTIST	Would load color
RICK1_H	GRAPHX	Would display
RICK1_P	ARTIST	Would display as B/W

As you can see there can't be a compatible file in between a color `TI-ARTIST' picture. Also TASS will load a TI-ARTIST B/W picture with for without the filename ending in `_P'; so this is also compatible with DRAW-N-PLOT pictures.

TASS looks at the joystick port for the peripheral like the `ATARI: Video Touch Pad' which I like to call the `THE RAT'. This is a device with 12 keys on it that has full control over TASS program just like the keyboard. So with one of these you could control the slideshow from a remote location.

The plans for converting an ATARI: Video Touch Pad which is normally sold with the STARAIDER game for the ATARI 2600 machine, to work on the TI99/GENEVE Joystick ports follow the TASS2001 docs and also a layout of the keys for the TASS2001 program on the VTP.

Phone: (416) 960-0925 voice
(416) 921-2731 TI-TOWER BBS

The BBS that I run is 24 hours 300/1200 (8N1) baud and is for any TI or GENEVE user. I am also currently the librarian of the local 9T9 UG.

Atari: Video Touch Pad Conversion V1.10.87.OPA by Gary Bowser

INSTRUCTIONS :

1) Unscrew the case of the Video Touch Pad, then remove the bottom part of the case, then remove the board from the case.

2) Unplug all the wires from the board then plug them back in this order:

COLOR OF THE WIRE	MARKINGS ON THE BOARD	COMMENTS
GREEN	ORN	
WHITE	RED	
	BLU	Left unconnected
ORANGE	BRN	
	VIO	Left unconnected
VIOLET	GN	>>>>>> These two wires for the GENEVE must be
YELLOW	YEL	
RED	WHT	>>>>>> reversed, VIOLET to WHT and RED to GN.
BLUE		Not used leave unconnected
BROWN		Not used leave unconnected

3) Remove the two resistors and solder two diodes (1N914/4148 Radio Shack Cat. # 276-1122) where the resistors were. When looking from the top side of the board (no printed circuit) with the left most wire being green, both diodes must have the orange part on the left.

4) Solder a short piece of wire between the two connections on the board that were left unconnected (BLU & VIO).

5) Now put the board back in the top part of the case with the printed circuit facing to the rubber keys. And close up the case.

6) Now type and run this small XB Demo sub-program. The values returned from the sub-program are as follows:

```
+--+--+--+
|1 |2 |3 |
+--+--+--+
|4 |5 |6 |
+--+--+--+
|7 |8 |9 |
+--+--+--+
|10|20|30|
+--+--+--+
```

If you hold down one of the bottom row of keys (10,20,30) and press one of the 1 to 9 keys the values will be added so you can have values from 1 to 39. The bottom row of keys works like shift keys so you can have four different values for each of the top keys 1 to 9. Zero is returned when there is no key pressed or when there is no new key pressed.

Sample Tester for the VTP in Extended BASIC:

```
1 CALL CLEAR
2 CALL VTP(K):: PRINT K :: GOTO 2
32000 SUB VTP(K)
32001 CALL JOYST(1,A,B):: CALL JOYST(2,C,D):: CALL KEY(1,E,X):: CALL KEY(2,F,K):
: K=A+C :: K=ABS((A+B)/4+(C+D)/2)-3*(K>0)-6*(K<0)+(F*2+E+3)/1.9 :: IF K<>G THEN
G=K ELSE K=0
32002 SUBEND
```

Since the VTP unit plugs into the Joystick port, the Alpha Lock will have to be up for full operation of the VTP unit in any program. VTP keyboard layout for the TASS program :

```
+---------+---------+---------+
|B/W COLOR|REPEAT   |ZOOM/MAG | CTRL
|         |         |         | FCTN
| 1     A | 2     B | 3     C | normal / SHIFT
+---------+---------+---------+
|U/D FLIP |L/R FLIP |CANON    | CTRL
|NEXT DSK |MENU     |PICTURE  | FCTN
| 4     D | 5     E | 6     F | normal / SHIFT
+---------+---------+---------+
|SAVE SCR |         |         | CTRL
|BACK     |REDO     |FORWARD  | FCTN
| 7   Yes |8        | 9    No | normal / SHIFT
+---------+---------+---------+
|  SHIFT  |  CTRL   |  FCTN   |
|         |         |         |
|         |   0     |         |
+---------+---------+---------+
```

Disk 103. GEE

Version: 1.0 *Author*:
Requires: EA *Language*: Assembly *Updated*:

A graphics programming language (not GPL!) out of Australia. Allows creation of BASIC programs that can do full bitmap graphics. Has capability to load and save TI-Artist screens, and has some animation effects. Neat program for anyone into BASIC programming and/or graphics.

```
dskdir. v2.0. 12-dec-96
Disk name               = GEE
Sectors total           = 360
Sectors used            = 152
Sectors available       = 206
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 ARTICLE       8 DIS/VAR 80   >022 007
002 >003 CLOCK         5 DIS/VAR 80   >029 004
003 >004 CURSOR        4 DIS/VAR 80   >02d 003
004 >00e CURVES        4 DIS/VAR 80   >0a1 003
005 >00f FOURIER       4 DIS/VAR 80   >0a4 003
006 >005 G-DOC1       46 DIS/VAR 80   >030 045
007 >006 G-DOC2       12 DIS/VAR 80   >05d 011
008 >007 GEE          33 PROGRAM      >068 032
009 >008 GEF          10 PROGRAM      >088 009
010 >009 GLOAD         3 DIS/VAR 80   >091 002
011 >00a GLOAD1        7 DIS/VAR 80   >093 006
012 >00b HELLO         2 DIS/VAR 80   >099 001
013 >00c LOAD          7 PROGRAM      >09a 006
014 >00d PRINTDOC      2 DIS/VAR 80   >0a0 001
015 >010 SPHERES       5 DIS/VAR 80   >0a7 004
```

Disk 103. Contents of file ARTICLE

G — THE GRAPHICS LANGUAGE
from Adelaide South Australia

Bob Warren

G can stand for many things including GRAPHICS, GREAT, and GOOD FUN. The graphics language G is all of these things. It was developed by Gene Krawczyk, one of the original members of ATICC, and he has made it available to the Club. It was written in Assembly Language, and so requires a disc and memory expansion to run.

G is a powerful graphics language with simple commands, which makes it ideal for children, but it is also sophisticated enough to provide a challenge for adults. Unlike TI-ARTIST, which only produces still pictures, G allows a form of animation, and can rapidly change screens. A screen can be STOREd at any time, a new screen drawn, and then the old screen RESTOREd when required. A screen can also be SAVEd to or LOADed from a disk when needed.

I have edited Gene's instructions and commands to produce a beginners version of G which everyone can use. Details of this version are given below. Gene also included some commands which allow an experienced Assembly Language programmer to execute Assembly routines, including sprites(I think). When I have sorted these out and written instructions for their use, I will present an ADVANCED G.

I hope there will be sufficient interest for a regular column in our bulletin to be devoted to G, both to answer queries and to exchange programing ideas.

Disk 103. Contents of file CLOCK

```
:START
 BGND 11 SCREEN 3 COLOR 13 SIZE 1 PRINT 60 0 "THE TIME
SIZE 0
BGND 15 5376 780
 COLOR 14 BOX 15 30 50 120
 FILL 20 40
 COLOR 8 PRINT 15 155 "PRESS FCTN BACK TO GO TO MENU.
 COLOR 1 PRINT 105 30 "ANALOG
COLOR 10 PRINT 20 175 "DIGITAL
 COLOR 15
 PRINT 120 46 "12" PRINT 124 138 "6"   PRINT 80 94 "9" PRINT 170 94 "3"
 P=32 LET L=360/60 T=360/12
 ARC 128 96 52 52 0 361
 TRACE 4 BCOLOR 1
 FOR I=0 TO 32 STEP 1 ARC 128 96 I I 0 361 NEXT I
 TRACE 1
 COLOR 12 FOR I=0 TO 40 STEP 1 ARC 128 96 I I 0 361 NEXT I
 SIZE 1 COLOR 12 PRINT 115 170 ":"
:HERE TRACE 2
 FOR Y=270 TO 360+270 STEP T
 ANGLE Y SET 128 96 DRAW 20
 G=Y-270/30
 COLOR 4 PRINT 100 170 G
FOR I=270 TO 360+270 STEP L
COLOR 12 SET 128 96
ANGLE I
DRAW P
J=I-270/L
COLOR 6 PRINT 130 170 J
 FOR C=0 TO 10 STEP 1 NEXT C
COLOR 12 SET 128 96 DRAW P
COLOR 7 PRINT 130 170 J
NEXT I
COLOR 12 ANGLE Y SET 128 96 DRAW 20
PRINT 100 170 G
COLOR 12
NEXT Y
CLS GOTO :START
```

Disk 103. Contents of file CURSOR

```
 SET 0 91 DRAW 255 ANGLE 90 SET 125 0 DRAW 191
 TRACE 1 X=100 Y=X
 SIZE 1 FORMAT 7
 PRINT 10 170 "PRESS AN ARROW KEY
 SIZE 0
 PRINT 5 0 "X=
 PRINT 5 10 "Y=
 O=X P=Y
:HERE DISPLAY 21 0 X-125 DISPLAY 21 10 Y-91
:HERE1
 KEY$ A
 IF A=68  THEN X=X+1 GOSUB :DRAW GOTO :DRAW1
 IF A=83 THEN X=X-1 GOSUB :DRAW GOTO :DRAW1
 IF A=69 THEN Y=Y-1 GOSUB :DRAW GOTO :DRAW1
 IF A=88 THEN Y=Y+1 GOSUB :DRAW GOTO :DRAW1
 GOTO :HERE1
:DRAW SET T R T=X R=Y PRINT 21 0 0-125 PRINT 21 10 P-91 O=X P=Y
 RETURN
:DRAW1 SET X Y
 GOTO :HERE
 STOP
```

Disk 103. Contents of file CURVES

```
REM CURVE PLOTTER
REM PLOTS SEVERAL SIN & COS CURVES
REM AND THE RESULTANT CURVE
REM JAMES N. STRICHERZ
REM 16 JUNE 1988
LET Z=90
LET F=2
LET G=4
LET H=7
LET J=8
LET K=10
FOR I=1 TO 1275 STEP 1
    LET A=Z
    LET B=Z
    LET C=Z
    LET D=Z
    SIN I A
    COS I B
    LET X=I/5
    LET Y=95+A
    COLOR F
    SET X Y TO X Y
    LET Y=95+B
    COLOR G
    SET X Y TO X Y
    LET Q=180+I
    COS Q C
    LET Y=95+C
    COLOR H
    SET X Y TO X Y
    SIN Q D
    LET Y=95+D
    COLOR J
    SET X Y TO X Y
    LET E=A+B+C+D
    LET Y=95+E
    COLOR K
    SET X Y TO X Y
NEXT I
:GO GOTO :GO
```

Disk 103. Contents of file FOURIER

```
REM EXAMPLE OF GENERAL FOURIER SERIES
REM Y=A0 + A(N)*COS(N*PI*X/L) + B(N)*
REM SIN(N*PI*X/L)
REM WHERE D=A(N); E=B(N); P=PI; L=L;
REM N=N; Q=A0; B,C ARE THE SIN AND COS
REM RESULTS; A IS THE COS/SIN ARGUMENT
REM JAMES N. STRICHERZ
REM 26 JUNE 1988
COLOR 7
LET D=1
LET E=2
LET P=360
LET N=4
LET L=2*P
LET Q=15
LET Z=50
REM Z MULTIPLICATION FACTOR IN SIN/COS
FOR I=0 TO 1275 STEP 1
REM 1275/5=255 WIDTH OF SCREEN
     LET B=Z
     LET C=Z
     LET X=I/5
     LET A=N*P*X/L
     SIN A B
     COS A C
     LET Y=-Q+B*D+C*E+45
     SET X Y
NEXT I
:GO GOTO :GO
STOP
```

Disk 103. Contents of file G-DOC1

G the high res language

from Adelaide South Australia

LOADING AND OPERATING G

To load G from E/A select option 5 from the E/A menu then type in

```
DSK1.GEE
```

and press **ENTER**

The program will load and then put you into the menu mode like so.

G the hi res language
(L)oad,(S)ave,(I)nit,(E)dit,(R)un.

Pressing the keys in parenthesis will put you into that mode.

(L)oad

Expects an input to be typed in e.g. DSK1.SAMPLE will load the file "SAMPLE" into memory. If you load another file the old file will be lost. If the load is successful then the program will be tokenised and then run.

(S)ave

Will save the file in memory. e.g. DSK1.SAMPLE will save what ever is in memory to the disk drive. If there is nothing in memory then it will save a blank line to the disk drive. You can also save to a printer or other peripheral by typing in the requirements e.g. TP will save the file to the thermal printer.

(I)nit

This will clear memory pointers and restart the program. This is similar to a cold start and the program will try to auto boot a file called GLOAD.

(E)dit

Will put you into the EDITOR. The key presses are the same as the ones TI uses. The cursor keys move the cursor on the screen like in TI-Writer. The main difference is that this editor does not window its screen, instead the 80 columns are in 2 rows of 40.

To delete a line press	FCTN 3
To insert a line press	FCTN 8
To scroll the screen up press	FCTN 4
To scroll the screen down press	FCTN 6
To delete a character press	FCTN 1
To insert a character press	FCTN 2
To go back to the menu press	FCTN 9
To run the file in memory press	FCTN 5

(R)un

Same as in BASIC.

Now a few simple programs to get your feet wet: anything in parentheses is the key press you have to do. From the menu mode with the original disk in drive one you press (L) to load a file.

Type in

`DSK1.GLOAD` **ENTER**

If there are no errors with your system then the file should be loaded into memory. When the file has loaded press (R) or FCTN 5 and a small demonstration will start. To stop the demo press and hold FCTN 9 this will take you back to the menu. To look at the demo press (E) to enter the editor.

All commands will be typed in uppercase on the left side of the screen. A small example will be shown first and then explanations.

SET 10 20 TO 100 150

This will draw a line starting at location 10 across and 20 down as the starting point. The line will then be plotted to position 100 across and 150 down. All coordinates are given as x y, the x is columns and the y is rows. The value of x can be from 0 to 255, and the value of y is 0 to 191. Notice also there are no commas or parentheses in this language to separate variables or commands. The separator is a space (the same as in FORTH).

SET 10 24 TO 134 180
STOP

This will put a line on the screen and then it will just stop. When you press a key you will be taken back to the command mode. The actual command SET puts a dot on the screen and then command TO draws the line from the old position to the new coordinate. This allows you to have as many TO's as you like to draw shapes. So the command TO all ways draws from the last plotted point to the new one. If you just want to put a dot on the screen you would use

SET 100 100

VARIABLES

Variables are the single letters of the alphabet.

CONSTANTS

Constants can be integer decimal or hex.

ARRAYS

One array is allowed @@(a) e.g. @@(200)

LET

assigns a variable a value e.g LET A=10 gives A the value of 10. LET A=10*3+4/2+C*B will give A the value of the calculation, the value is worked out left to right and there is no precedence among the mathematical operators as there is in BASIC. Also you can't use parentheses to change the way you want the program to calculate the value.

As in BASIC the LET command is optional, but if you want speed use it, as LET quickens the actual time for processing. Without LET there is a lot of checking.

CLS

Clears the graphic screen. The screen is not saved and so once this is done there you would have to design this screen again.

GOTO :LABEL

Does a branch to the label in the file e.g.

:SAM GOTO :SAM

This is the same as BASIC 100 GOTO 100 All labels must be preceded with the colon and must not be more then 6 characters in length, 7 counting the colon. Also the label must be the first word on a line otherwise the search routine will not find your label. If you put a space in front of the label the word will also be missed.

GOSUB :LABEL

Same as GOTO except a return is stored so you can do subroutines. There are only 40 levels of subroutines allowed in G.

RETURN

Returns from a GOSUB same as in BASIC.

REM

As in BASIC it allows you to type in remarks in the program. When the interpreter comes to a REM statement it automatically goes to the next line. Additionally, the program does not alter your file in any way so any extra spaces you type in are left there.

COLOR c

Changes the foreground pen color to a new value. The value is from 0 to 15. Any value larger then this is truncated.

BCOLOR c

Same as COLOR but changes the background color. Default is clear. Seems only to be used in TRACE 4.

SCREEN c

Alters the color of the screen to the value of c, which can be from 0 to 15.

BOX x y a b

Draws a box on the screen whose top left hand corner is x,y. The box is (a) amount wide and (b) amount tall. e.g. BOX 100 100 50 20 puts a box on the screen which is 50 pixels wide and 20 pixels tall.

FOR NEXT STEP

Same as in BASIC, but only 10 levels of nesting allowed. FOR I=1 TO 1000 NEXT I for a delay.

ANGLE

Sets a new angle for the DRAW command can be between 0 to 360 degrees Other numbers can be used but if larger values are used they are just cut down. Also larger numbers may slow down execution time.

ANGLE 250 sets the angle to 250 degrees

DRAW

Draws a line from last point plotted at the angle specified in the angle command.

IF THEN

Similar to BASIC IF expression. If true then do rest of line, else go to next line. You can not do this though IF A=10 THEN :LABEL is not correct, you must do this instead IF A=10 THEN GOTO :LABEL

LOADS

Loads a bit map screen from disk same as Graphx does. LOADS DSK1.SAMPLE loads a screen.

SAVES

Saves a screen to disk Graphx format.

LOAD

Load a new file into memory and run it.

KEY$

KEY$ A puts key press in variable A. No key equals 255.

FORMAT

Sets up distance between characters for use in the print statement. The normal distance is 8 and in the 40 column mode is 6

FGND

Sets the foreground color FGND 6 sets the fgnd color of ALL "ON" pixels to 6 (red) FGND 6 0 255 sets the "ON" pixels from 0 to 255 to red

BGND

Sets the background color Same rules as FGND but works on the background.

TRACE a

Changes the drawing mode of the pen.

TRACE 0 delete:	From now on all pen commands will erase.
TRACE 1 draw:	This is the default all pen commands now to the obvious and allow drawing.
TRACE 2 invert:	All pixels that are on are now turned off and all pixels that are off are turned on. A very useful mode.

TRACE 3 (special color mode) will not alter any pixels but will just alter the foreground color of the pixels. this will only be noticed if you have a pixel on.
TRACE 4 (special color mode) This mode will show up straight away as you do not draw with pixels but with background colors. The background color has had to have been set up already with the BCOLOR command, now all drawing will be 8 pixels wide by 1 pixel tall. A strange mode but useful for some types of work, particularly for clearing backgrounds.

FILL x y

Fills in a area on the screen. this is not a very exact type of fill but it may do the job. It might have to be used a few times however to fill in awkward shaped polygons . You may have problems with this command.

PRINT x y a or
PRINT X Y "SKFFK"

Prints on the screen at location x y the value of a which can be a variable or a number. You can have as many variables as you like in one print statement. e.g. PRINT 10 10 A B C D 100 prints on the screen the value of a b c d and the number 100. If you put a quote after the coordinates then the characters after the quotes are displayed on the screen. The string of characters can be terminated by a quote or by leaving the rest of the line blank.

RND

Gives a random number

RND 255	A generates a random number between 1 and 255 and puts it in A

SIZE

Alters the size of the characters in the print statement.

SIZE a	alters the size of the characters in the print mode .
SIZE 0	default same size as normal.
SIZE 1	double size characters these are the only two sizes available at the present time.

PATTERN

Alters the fill pattern.

RESTORE

Puts up the old screen.

ARC

Allows you to draw circles and arcs. ARC 100 100 25 25 0 360 draws a circle at 100 100 with a diameter of 25.

SIN

Gives the sin of a number.

COS

Gives the cos of a number.

WRAPON

Allows screen wrap.

WRAPOFF

Does not allow screen wrap

STORE

Saves a screen to memory recalled by the restore command.

CLEAR

Clears an area of the screen.

INVERT

Inverts an area of the screen.

LINESTYLE

Changes the type of line being used.

DISPLAY

Prints data to the screen but erases what ever is underneath.

STOP

Waits for a key press and then goes to main menu.

SUMMARY OF COMMANDS

ANGLE A (0-360)
ARC A B C D E F
ARRAYS @@(10) @@(B) can be between 0 to 500 numbers can be in hex or decimal same as E/A
BCOLOR A (0-15)
BGND 11 0 2000
BOX 10 10 110 110
CLEAR 10 10 100 110
CLS
COLOR 6
COS 32 B
DISPLAY 10 10 "STRING
DISPLAY A B VARIABLE VARIABLE
DRAW 50
FILL 10 10
FGND 11 100 200
FOR I=TO 1000 NEXT STEP
FORMAT 8
GOTO :LABEL
GOSUB :LABEL
IF A=10 THEN continue the rest of the line
INVERT 10 10 110 110
KEY$ A
LET A = 100

LINESTYLE A
LOADS DSK1.SCREEN
LOAD DSK1.PROGRAM
PATTERN
PRINT
RETURN
REM
RESTORE
RND 200 A
SCREEN 3
SET A B TO A B
SIN 23 A
SIZE 0
STEP optional
SAVES DSK1.SCREEN
STOP
STORE saves a screen to temporary memory
THEN optional
TRACE 1-4
VARIABLES are single letters A-Z.
WRAPOFF
WRAPON

Disk 103. Contents of file G-DOC2

THE G CORNER

Last time I gave a list of the commands for G and a general guide to running G. However, the instructions for 2 commands, SIN and COS, were wrong. This time I intend to show the correct way of using SIN and COS, and to illustrate them in some simple, but effective programs. Those who are a little rusty on trigonometry can skip straight to the programs.

The problems with SIN and COS arise because the values of sin and cos vary between +1 and -1. Since G only uses whole numbers, SIN and COS would only return the numbers -1,0, or +1, which would not be of much use. In most programs sin and cos are usually multiplied by some other number anyway, so Gene set up the commands as: SIN A B, and COS C D, where A and C are the angles, B and D are the numbers multiplying sin and cos. B and D are also the variables to which the values are returned. For instance,

```
LET B=10 SIN 30 B
```

sets B equal to 5, since sin 30 = 0.5, so 0.5*10=5.

Try this little program, it is as easy as ABC.

```
A=128 B=96
V=90 W=60 F=1 G=3
SCREEN 1
C=6
FGND C
FOR I = 1 TO 5
A=A+1 C=C+1
FOR T = 0 TO 360
LET M=V LET N=W
COS F*T M
SIN G*T N
SET A+M B+N
NEXT T
FGND C
NEXT I
STOP
```

Also try other values for the variables F and G and see what happens. In this program SIN and COS are immediately multiplied by the pixel locations on the screen, and so the results are always large whole numbers. In other situations where you may wish to use SIN and COS you may have to multiply them by a large number, say 100, do the calculation, and then divide by 100 to get the answer to the required accuracy. Now try this program.

You can use the same technique of using a variable in different functions to plot all sorts of patterns.

As an example of simple animation, try this little program.

```
REM MOVING HELLO
FOR I=8 TO 128 STEP 4
DISPLAY I 20 " HELLO"
REM NOTE THE SPACE NEXT I
STOP
```

If anyone has any problems or suggestions(or even reads this column), I would be glad to hear from them.

Disk 103. Contents of file GLOAD

```
SCREEN 14
WRAPON
TRACE 4
FOR I=0 TO 64
BCOLOR I
SET I*16 0 TO I*16+90 191
NEXT I
TRACE 1
WRAPOFF
STORE
FOR I=1 TO 5
CLS
RESTORE
NEXT I
COLOR 5
SIZE 1
A=26 B=234 K=24 GOSUB :SHOW
PRINT   28 K "DEMONSTRATION"
A=80 B=190 K=60 GOSUB :SHOW
PRINT 82 K "OF 'G'"
A=56 B=206 K=96 GOSUB :SHOW
PRINT 58 K " BY ATICC"
SIZE 0
LOAD DSK1.GLOAD1
STOP
:SHOW FOR J=A TO B STEP 8
GOSUB :CLR
NEXT J
RETURN
:CLR
BCOLOR 15
TRACE 4
SET J K TO J K+16
TRACE 1 RETURN
```

Disk 103. Contents of file GLOAD1

```
LET B=48
:START
CLS
RESTORE
TRACE 4
BCOLOR 15
 FOR I = 50  TO 180 STEP 8
 SET I 10 TO I 120
 NEXT I
 TRACE 1
 COLOR 8
DISPLAY 60 20 "SELECT ACTION"
PRINT   60 40 "1. STARS"
PRINT   60 60 "2. LINES"
PRINT   60 80 "3. SNAIL"
PRINT   60 100 "4. CIRCLES"
:QUEST KEY$ A
IF L=1000 THEN GOTO :DEMO
:LOOP
IF A = 49 THEN GOTO :STARS
IF A = 50 THEN GOTO :LINES
IF A = 51 THEN GOTO :SNAIL
IF A = 52 THEN GOTO :CIRCL
IF A = 54 THEN GOTO :PROG
IF A = 53 THEN LOAD DSK1.GLOAD
L=L+1
GOTO :QUEST
:DEMO
LET B=B+1
LET L=1
IF B>54 THEN LET B=49
LET A=B
GOTO :LOOP
REM random lines
:STARS CLS
SCREEN 1
FOR I=0 TO 1000
 RND 255 A RND 191 C
  SET A C
NEXT I
FOR I=0 TO 2000
NEXT I
GOTO :START
REM lines
:LINES
CLS SCREEN 1
FOR I=0 TO 255 STEP 10
 SET 0 0 TO I 191
```

```
NEXT I
FOR P=191 TO 0 STEP -10
 SET 0 0 TO I P
NEXT P
FOR I=0 TO 2000
NEXT I
GOTO :START
REM snail
:SNAIL
CLS     COLOR 6 A=1         REM RED
FOR I=0 TO 360 STEP 5
SET 125 100
ANGLE I
DRAW 10+A
A=A+1
NEXT I
FOR I=0 TO 2000
NEXT I
GOTO :START
REM random circles
:CIRCL
CLS
FOR I=0 TO 90 STEP 4
ARC 127 95 I 90 0 360
NEXT I
FOR I=0 TO 90 STEP 4
ARC 127 95 90 I 0 360
NEXT I
FOR I=0 TO 1000
NEXT I
CLS
FOR I=0 TO 100
RND 255 A RND 191 D RND 20 C RND 20 E
ARC A D C E  0 360
NEXT I
FOR I=0 TO 2000
NEXT I
GOTO :START
REM load a screen
:PROG
LOADS DSK1.PIC
SCREEN 4
FOR I=0 TO 3000
NEXT I

GOTO :START
```

Disk 103. Contents of file HELLO

```
REM MOVING HELLO
FOR I=8 TO 128 STEP 4
DISPLAY I 20 " HELLO"
NEXT I
STOP
```

Disk 103. Contents of file PRINTDOC

```
.LM 0;RM 79;FI
.IF DSK2.G-DOC1
.IF DSK2.G-DOC2
```

Disk 103. Contents of file SPHERES

```
REM ARC: CIRCLES AND A LOT MORE!
REM ARC X Y A B C D
REM WHERE X,Y IS THE CENTER OF THE ARC;          C IS THE BEGINNING ANGLE;
REM       D IS THE ENDING ANGLE;                 A IS THE RADIUS ALONG X AXIS
REM       B IS THE RADIUS ALONG Y AXIS
REM ELLIPSOIDS BECOME A MATTER OF SPECIFYING DIFFERENT A AND B RADII.

REM JAMES N. STRICHERZ
REM 26 JUNE 1988
REM
LET Z=10
LET X=100

COLOR 7
FOR L=1 TO 3
FOR I=60 TO 140 STEP 10
    IF I>100 THEN A=A-10
    IF I<100 THEN A=A+10
    IF I=100 THEN A=A+10
    LET B=A/2
    IF L=1 THEN GOSUB :ARC1
    IF L=2 THEN GOSUB :ARC2
    IF L=3 THEN GOSUB :ARC3
NEXT I
    FOR Q=1 TO 100
    NEXT Q
    CLS
NEXT L
DISPLAY 10 10 "FINISHED! PRESS ANY KEY!"
STOP
END
:ARC1 ARC X I A B 0 360
      RETURN
:ARC2 ARC I I A B 0 360
      RETURN
:ARC3 ARC I X B A 0 360
      RETURN
```

Disk 104. PLUS! Disk 1

Version:
Requires: XB
Author: Jack Sughrue
Language: XB, AL
Updated: 4/5/88

This two-disk set is essentially an add on for the popular Funnelweb system. It contains templates and programs aimed at making your word processing more productive. Extensive documentation is provided. The user response has been excellent. PLUS! received an excellent review in *MICROpendium*.

```
dskdir. v2.0. 12-dec-96
Disk name               = PLUS-1
Sectors total           = 360
Sectors used            = 356
Sectors available       = 2
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 1             2 DIS/VAR 80   >022 001
002 >003 2             2 DIS/VAR 80   >023 001
003 >004 3             2 DIS/VAR 80   >024 001
004 >005 3/COL         8 PROGRAM      >025 007
005 >006 4             2 DIS/VAR 80   >02c 001
006 >007 5             2 DIS/VAR 80   >02d 001
007 >008 7             2 DIS/VAR 80   >02e 001
008 >009 8             3 DIS/VAR 80   >02f 002
009 >00a 9             2 DIS/VAR 80   >031 001
010 >00b C1            3 DIS/VAR 80   >032 002
011 >00c C2            3 DIS/VAR 80   >034 002
012 >00d C3            4 DIS/VAR 80   >036 003
013 >00e C4            3 DIS/VAR 80   >039 002
014 >00f C5            3 DIS/VAR 80   >03b 002
015 >010 C7            3 DIS/VAR 80   >03d 002
016 >011 C8            3 DIS/VAR 80   >03f 002
017 >012 C9            4 DIS/VAR 80   >041 003
018 >013 CAT          11 PROGRAM      >044 010
019 >014 DOCPACKS/1   67 DIS/VAR 80   >04e 066
020 >015 DOCPACKS/2   57 DIS/VAR 80   >090 056
021 >016 DOCPACKS/3   41 DIS/VAR 80   >0c8 040
022 >017 FNLSTRIP      7 DIS/VAR 80   >0f0 006
023 >018 G             2 DIS/VAR 80   >0f6 001
024 >019 G1            3 DIS/VAR 80   >0f7 002
025 >01a G2            3 DIS/VAR 80   >0f9 002
026 >01b G3            3 DIS/VAR 80   >0fb 002
```

```
027 >01c IG!PAY         8 PROGRAM       >0fd 007
028 >01d INSTADUMP      2 PROGRAM       >104 001
029 >01e INSTALABEL     4 PROGRAM       >105 003
030 >01f INSTAMAIL      4 PROGRAM       >108 003
031 >020 INSTAPRINT     2 PROGRAM       >10b 001
032 >021 L1            10 DIS/VAR 80    >10c 009
033 >115 L2             4 DIS/VAR 80    >116 003
034 >119 L3             4 DIS/VAR 80    >11a 003
035 >11d M1             2 DIS/VAR 80    >11e 001
036 >11f M2             3 DIS/VAR 80    >120 002
037 >122 PLUS!DOCS     58 DIS/VAR 80    >123 057
038 >15c TLTESTER       2 DIS/VAR 80    >15d 001
039 >15e teenypro       6 PROGRAM       >15f 005
040 >164 tinyteeny      2 PROGRAM       >165 001
```

Disk 104. Contents of file PLUS!DOCS

PLUS! DOCS

(PLUS! V.^ 1.0 -4/88)

This Fairware word-processing companion and utilities disk of templates, tutorials, articles, codes, programs for various versions of TI-Writer (particularly the Australian Funnelweb), was created, edited, adapted, rewritten, and/or collected especially by ME for the FAIRWARE marketplace. I hope you enjoy this disk, make backups, and pass it on to other TI people in this identical form. If you like PLUS! please send $10 to Jack Sughrue, Box 459, East Douglas, MA 01516. Thank you.

FIRST - Print out the four DOCPACKS files (which include 12 essential textfiles in this order):

1. ECODING, ALLCODING, IDEAL/SYS, QUICK/REF;
2. IMPLANT/TL, REDEF/TL, GRAPHS/TL;
3. MULTCOLDOC, SMALL/DOCS, MAX/DOC;
4. FWFLOW/1, FWFLOW/2.

DON'T FORMAT THE FOLLOWING FILES TO HARDCOPY:

1 2 3 4 5 7 8 9 C1 C2 C3 C4 C5 C7 C8 C9 M1 M2 G G1 G2 G3 T1 T2 T3

(as they are screen and format codes),

IFFING/TL

(as it is prepared to run through the XB program MULTCOLUMN / see docs), and

GEM/DOCS

(as those two files [BOX/DOC, BOX/REF], along with the T files, can be accessed ONLY by some STAR printers and REQUIRE THAT Cn CODES BE ON DSK1!).

All other DV80 files may be printed out (FNLSTRIP, L files, etc.)

But these tutorial docs are just a tiny part of the 70-file powerhouse of PLUS!, which includes instant DV/80 text/graphic printer; flexible banner program; personalized desk and yearly calendars; instant labelmaker; multiple-column printing; tinyteeny wordprocessing; envelope cataloguer; gothic printing; templates for TI's built-in graphics; codes that create IFfing with TransLiteration; Mnemonic Reference Chart for built-in TL codes; an active cataloguer and mini-DM for Drives 1, 2 & 3; mini screen dump; Pig-Latin writer, etc. All on ONE disk! PLUS single-stroke autocoding, the most popular feature of all PLUS-series disks, and many more features.

* G loads are TL graphing templates . (see tutorials)

* L loads are LETTER templates.

Using these, it is easy to design your own. L1 uses some graphic codes (which may have to be changed for your particular printer) and gives all the directions for creating L templates. L2 is a smaller, modified, very printer-compatible version of the same. Both G and L files include all the active TL keys described in the tutorial. L1 & 2 use the oft-requested Define Prompts, so the dates and names have to be typed in during FORMATting. These DPs can be eliminated if you have no use for this sort of thing. L3 is a business letter template without DPs.

* M loads prepare formatting structure for MULTCOLUMN.

* # & C loads are the autoloaders in the EDIT mode (see DECODING). Each number when used as the single-keystroke filename in LoadFile, automatically sets your TABs to 1 & 38 (screen size) to eliminate windowing and sets FORMATting "IF" codes. Keep these codes in the opening lines. When your text is run through the FORMATter, all hardcopies contain those particular code functions. For example, LF DSKn.4 loads C4. That gives you term-paper formatting (wide margins, double spacing, name headers, auto page numbering, NLQ, etc.) and, like ALL of these codes, complete access to the TL templates for the full range of fonts, size, graphics, and other printer commands through the TL key. (see tutorials)

Here is what the other autoload numbers will do:

C1 — the basic TL coding with normal margins;

C2 — condensed coding (118-columns wide) with built-in spacing;

C3 — normal mode (what I use 90% of the time) with graphics (Heart/N; Goose/Y) and some C9 coding;

C5 - full 80 columns;

C6 is empty for your own personalization;

C7 — full non-spaced TL that includes an arrow pointer and a copyright symbol;

C8 — loads up a fully-spaced, compressed, superscripted, narrow-lined text mode to type 134 columns by 134 rows, about 18,000 characters per page!;

C9 — (see ALLCODING).

On MOST of these templates the WXYZ keys are for my special redefinitions or are left open for redefining by you. G files redefine WXYZ into graphics, while maintaining the standard TLing.

* DOCPACKS files: — DECODING and ALLCODING are comprehensive tutorials with detailed examples on the use of all the codes and templates.

(IMPORTANT DOCS!)

QUICK/REF is a comprehensive, one-page quick-reference chart for any form of TI-Writer (best for Funnelweb). This chart also includes the standard TLing. Keep this handy guide near your computer.

FWFLOW is a marvelous flowchart reprinted here with permission of author Charles Good (of Lima Users) which step-by-steps the FWB processes (especially CONFIG) on V.

4. These files, the absolute best on the subject, were an immense help to me.

SMALL/DOCS explain how to use PLUS's SMALLIFICATION program which reduces XB programs considerably to save space and run faster and more efficiently. (Change drive # in Line 360 if Smallify 1 & 2 are not in Drive 1.) One SSSD disk of programs I wrote for school was reduced by 112 sectors when run through SMALLIFICATION!

MAX/DOC explains how to use the wonderful MAX/RLE. MICKEY is a graphic to test it out.

* CATPLUS's active cataloger lists disk contents of Drives 1, 2 or 3 (including RAM). Screen breaks files into program and other. To load program files (if in XB only [or BASIC that is supported by XB]) press the corresponding letter. Files list type and number of sectors used; disk title and total sectors used are also displayed. SCREEN OPTIONS:

> 1) CONTINUE — If there are too many files for the screen, pressing 1 displays the next batch of files.
>
> 2) QUIT — Quits back to TI screen.
>
> 3) AGAIN — Reloads to catalogue another disk in same or different drive or back to Funnelweb,
>
> 4) DELETE — Deletes by answering PROGRAM (P) or FILE (F) Then letter corresponding to line of File or Program. Computer double checks that you are still okay by asking if you are sure you want _____ deleted.
>
> 5) PRINT — Prints the catalogue as shown in condensed format on a PIO printer. These configurations may be changed by LISTing the XB program.

* INSTALABEL is FAST, prints up to six lines of ANY KIND of LABEL in condensed (default) or normal. Type what you want on the lines and number needed. When labels are done the program lets you immediately create another. Typing 99 quits.

* INSTAPRINT prints any DV80 text or graphic file instantly without using EDITOR or FORMATTER. It prints just as if you did PF in EDIT mode. Give DSKn.FILENAME as asked. Be sure your PRINTER IS ON! Like all tiny INSTA XB programs, this is FAST!

* INSTADUMP Is a 2-sector program that dumps any non-redefined characters directly from the screen to printer. Put your printer on and run the program. Easily adaptable to GOSUBs, for printing partial LISTs, for tiny wordprocessing duties, for printing character-graphic screens. Experiment. You'll be surprised what it can do.

* INSTAMAIL is a very quick hardcopy address maker, handy for people who need printouts for mail lists without going through a database.

* CALENDAR PROGRAMS let you create personalized desk and yearly calendars suitable for writing in daily notices (with additional blocks for other important monthly announcements). You are led through the processes and may print out text in upper or lower case. Be sure you have your printer on and ready to go with paper lined right up to printer head. After you have typed your heading and chosen year and month, you are asked (on DESK/CAL only) if you want messages for specific dates. If Y, the computer will ask which date. Type number. Then type message which is printed on that date. You must press ENTER after each 10-letter (maximum) line. There are 6 lines in each box. After the last line is ENTERed you are asked if you want any more messages (Y/N). If N the calendar prints instantly. If Y you can continue to put in messages (including in boxes before and after month by giving -3 or 35 [or whatever] as before or after dates). When the calendar has printed, you're asked if you want one exactly the same or if you wish to change some messages on an otherwise identical calendar or if you want to go on. Then you're asked if you want another month to print. An N here returns you to LOAD. ^ A Y asks you for title, year, month again and print another desk calendar out for you. And so on.

* BANNER PROGRAM operates off two menus. When the first screen comes up choose:

N — Normal. Normal TI character set.
R — Reformed. New character set (remains in effect even when returning to PLUS! unless you return to normal with the last menu [S - Start Over]).
C — Create. Design your own characters, numbers, or pictures by using the 8×8 grid. Type the Row # then the Column #. This will be automatic without a need to ENTER each time. S stores (for later retrieval or embellishment) and Print. You're given a magnification option to make your special creations up to 8 inches high.
Q — Quits you to LOAD. The second menu of each above set lets you select the height of the items you are going to print. (*Note*: All letters are enlarged using a collection of the small letters selected [except in C] and all are printed sideways.) Lastly, you're asked to print the words you want bannered.

* 3/COL — this modified PD program reduces catalog printout to a clean, tiny, dark level in 3 neat columns WITH DATE for perfect envelope catalogs. [Change "CHR$(11)" on Line 120 to "(17)" if printout overlaps.] [Change to YOUR name on line 220.] Thanks again, Charles Good.

* IG!PAY Type any words, phrases, or sentences you want in English. The computer converts English to Pig Latin. If you want multiple phrases continue to press Y when offered option. The percent sign lets you print directly. N quits program (still in memory for another RUN or LISTing).

* SETUP is a sophisticated STAR (EPSON)-compatible printer setup. Load up whatever codes you want in your printer. Then use your wordprocessor [or whatever] without shutting off the printer. What you established comes out in the hardcopy.

* tinyteeny is the world's smallest wordprocessor! Using subscript, this short program even lets you print on adding machine paper! Though many of the features of TIW are missing, tinyteeny does let you print your whole novel on a dozen pages (or half-a-dozen feet of adding machine paper). What other word processor can make that claim? tinypro has a tutorial.

* PLUS!VIEW is a workshop/presentation tool. This program (which is used here to explain the features of PLUS!) can be adapted to any business or user-group application. Make your point clearly and succinctly.

* FNLSTRIP makes a 4-line console strip that includes the double-line commands for TI-Writer, the single-line for DM-1000 (which is within Funnelweb), and another line for manually filling in commands for other software. This file can also be a template for any needed strips.

* MULTCOLUMN is a hybrid with lots of history. Years ago many programs were developed to convert TI-Writer files into two-column printouts. Probably the best of the lot is Jim Peterson's PRINTALL (which even prints up to five columns). The first I saw was an anonymous one from a New Jersey newsletter years ago. Then within months, it seems, there were dozens of variations, each with improvements and additions (such as more than two columns) *MICROpendium*, for example, recently published another, much-improved version by William Brown (which, itself, was REVISED in 1985!).

The LA Users (particularly Tom Freeman, Mike Dodd, and George Steffan) improved upon the earlier variations and added more selection screens (similar to Peterson's PRINTALL and Bartholomew Punch's peculiar PD version written specifically for my class to make a 5th-grade newspaper in 1983!) This small PLUS! program with auto-templating owes a debt of thanks to all these people and to all those unnamed persons whose efforts contributed to this "final" result. M1 & M2 are the autoloaders for setting up the structure for this conversion. MULTCOLDOC is essential for operating the three files. A newsletter editor's dream come true.

GOTHIC prints out beautiful Gothic lettering in upper and/or lower case. Great for awards, etc. This PD program has been in group libraries for a long time. I don't know the author and have asked lots of people. If anyone knows, please write me so I can give credit.

We assume no responsibilities for improper use or malfunction of any files contained on this disk. Please contact me with suggestions, comments, improvements. INCLUDING DOCPACKS FILES, THERE ARE ABOUT 70 FILES (and a lot of work!) ON THIS DSSD (OR FLIPPY SSSD) DISK. WHEN SHARING THIS DISK, PLEASE MAINTAIN THE INTEGRITY OF THIS WORD-PROCESSING COMPANION/UTILITY BY KEEPING ALL THE FILES INTACT.

I appreciate your support, Jack Sughrue.

Disk 105. PLUS! Disk 2

Version: *Author*: Jack Sughrue
Requires: *Language*: *Updated*:

Part two of the PLUS! system.

```
dskdir. v2.0. 12-dec-96
Disk name               = PLUS-2
Sectors total           = 360
Sectors used            = 356
Sectors available       = 2
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BANNER       22 PROGRAM      >022 021
002 >003 DESKCAL      18 PROGRAM      >037 017
003 >004 DOCPACKS/4   57 DIS/VAR 80   >048 056
004 >005 GEM/DOCS     52 DIS/VAR 80   >080 051
005 >006 GOTHIC       49 INT/VAR254   >0b3 048
006 >007 IFFING/TL    17 DIS/VAR 80   >0e3 016
007 >008 MAX-RLE      35 DIS/FIX 80   >0f3 034
008 >009 MICKEY        9 DIS/FIX128   >115 008
009 >00a MULTCOLUMN    9 PROGRAM      >11d 008
010 >00b PLUS!VIEW    20 PROGRAM      >125 019
011 >00c SETUP        22 PROGRAM      >138 021
012 >00d SMALLIFY/1    8 PROGRAM      >14d 007
013 >00e SMALLIFY/2    6 PROGRAM      >154 005
014 >00f T1           11 DIS/VAR 80   >159 010
015 >010 T2            4 DIS/VAR 80   >163 003
016 >011 T3            7 DIS/VAR 80   >166 002 >012 004
017 >016 YEARLY/CAL   10 PROGRAM      >017 009
```

Disk 106. c99 for MDOS

Version: *Author*: Clint Pulley
Requires: *Language*: *Updated*:

Clint Pulley's popular c99 compiler, modified to run under MDOS mode on the 9640. Allows creation of MDOS programs when used with QnD Linker (disk 107).

```
dskdir. v2.0. 12-dec-96
Disk name               = C99MDOS
Sectors total           = 360
Sectors used            = 322
Sectors available       = 36
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 C99C         58 PROGRAM    Y >022 057
002 >003 C99D         44 PROGRAM    Y >05b 043
003 >004 C99_DOC      39 DIS/VAR 80   >086 038
004 >005 CFIO         11 DIS/FIX 80 Y >0ac 010
005 >006 COMPILER_D   16 DIS/VAR 80   >0b6 015
006 >007 CONIO_H       3 DIS/VAR 80 Y >0c5 002
007 >008 CSUP         12 DIS/FIX 80 Y >0c7 011
008 >009 EXPLST       25 PROGRAM      >0d2 024
009 >00a EXPLST_C     13 DIS/VAR 80   >0ea 012
010 >00b MALLOC_O      3 DIS/FIX 80   >0f6 002
011 >00c MALLOC_S      9 DIS/VAR 80   >0f8 008
012 >00d POLLTST_C     3 DIS/VAR 80   >100 002
013 >00e POLLTST_O     5 DIS/FIX 80   >102 004
014 >00f PRINTF       13 DIS/FIX 80   >106 012
015 >010 RANDOM_C      5 DIS/VAR 80   >112 004
016 >011 README        5 DIS/VAR 80   >116 004
017 >012 RNDTST        9 PROGRAM      >11a 008
018 >013 RNDTST_C      4 DIS/VAR 80   >122 003
019 >014 RNDTST_CL     2 DIS/VAR 80   >125 001
020 >015 SCANF        15 DIS/FIX 80 Y >126 014
021 >016 STDIO_H       3 DIS/VAR 80 Y >134 002
022 >017 UNTAB_C      11 DIS/VAR 80   >136 010
023 >018 WHEN          7 PROGRAM      >140 006
024 >019 WHENTST_C     3 DIS/VAR 80   >146 002
025 >01a WHEN_C        4 DIS/VAR 80   >148 003
```

Disk 107. MDOS Quick and Dirty Development System

Version: *Author*: Clint Pulley
Requires: *Language*: *Updated*:

Includes an 80-column program editor, assembler, and linker for use under MDOS on the 9640. The assembler and linker can be run in batch mode for automated program builds.

```
dskdir. v2.0. 12-dec-96
Disk name               = MDOSQDDS
Sectors total           = 360
Sectors used            = 119
Sectors available       = 239
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 QDA           8 PROGRAM    Y >022 007
002 >003 QDA_DOC      11 DIS/VAR 80 Y >029 010
003 >004 QDE          43 PROGRAM    Y >033 042
004 >005 QDE_DOC      39 DIS/VAR 80 Y >05d 038
005 >006 QDL           9 PROGRAM    Y >083 008
006 >007 QDL_DOC       9 DIS/VAR 80   >08b 008
```

Disk 108. p-Code Utilities

Version: *Author*: Jerry Coffey, others
Requires: p-Code *Language*: Pascal *Updated*: 11/01/88

Programs which allow you to partition a p-System disk. Another allows you to transfer p-Systems files via modem. Documentation and source code is included.

```
dskdir. v2.0. 12-dec-96
Disk name               = PUTILSTRTM
Sectors total           = 360
Sectors used            = 298
Sectors available       = 60
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00c LOAD          8 PROGRAM      >0b6 007
002 >00e P-UTILDOCS   79 DIS/VAR 80   >0ef 078
003 >002 PASCAL      160 DIS/FIX128   >022 128 >003 030
004 >00d PUTIL        51 INT/VAR254   >0bd 050
```

[*Note*: This disk is not from a BCS original. It was kindly supplied by Jerry Coffey on August 9, 1999, and is a copy of his disk that he sent to the BCS library. CaDD Electronics would like to acknowledge Dr. Coffey's help and generosity in tracking down this material - Ed.]

Contents of file P-UTILDOCS

PASCAL TO TI TRANSFER SYSTEM

by Jerry Coffey

Instructions for Up/Downloading Pascal Files with any TI Terminal Emulator capable of TE2 or XMODEM transfers

Overview

This system uses a program called PAS>TI (XBasic) to copy images of p-System TEXT or CODE files onto TI disks in a format that can be routinely handled by most TI disk managers. The files appear to the TI system as normal Display/Fixed 128 files, however they are byte-for-byte images of p-System source code (or other TEXT) files or executable (compiled) CODE files. These files can be transferred as TI files using either TE2 or Paul Charlton's implementation of the XMODEM protocol. Using utilities of the UCSD p-System, the downloaded files can be recovered as standard TEXT files or executable code on disks formatted for the p-System.

Note: The technique used in PAS>TI can be readily adapted to select a discrete block of FORTH screens (comprising a FORTH program) and transferring it as a D/F128 file manageable by the TI disk system.

1. Preparing p-System files

Files to be copied to a TI disk should first be T)ransferred using the p-System Filer to a fresh Z)eroed p-System disk (SSSD preferred). This will be the source disk for the transfer to a fresh TI disk. The disk may use a single or duplicate directory. There should be *no gaps* on the disk — if you remove a file, the disk should be K)runched before using PAS>TI.

2. Copying Pascal to a TI disk

The program PAS>TI copies up to 30 p-System files into their D/F128 images on a TI disk. (*Note*: In the first version a maximum of 18 files was artificially imposed.) Select XBasic and load the program PAS>TI. Then place the p-System disk prepared as above in Drive #1. A fresh TI disk in SSSD format should be placed in the target drive. When the disks are in place, RUN the program. Messages at the bottom of the screen will indicate the progress of the transfer. *Note*: The first version of this program did not alphabetize the file pointers. This has been changed in the current version to make the program compatible with all disk management software. SYSOPs may use this program to directly prepare a download disk.

3. Handling D/F128 Files

The files created by PAS>TI can be treated as any other standard TI files. They can be copied to and from floppy or hard disks and transferred via modem using a binary (8-bit) protocol — either TE2 or XMODEM. The file names assigned by PAS>TI are created from the original Pascal file names by truncating to 9 characters (if necessary) and appending the first character of the Pascal type, e.g. <T>EXT or <C>ODE. Thus DISASMBLER.TEXT becomes DISASMBLET on the TI disk. This convention should be noted to avoid producing confusing or identical names on the TI disk.

4. Reversing the Transformation

Once the D/F128 Pascal images have been downloaded from a BBS (or transferred by modem between two TI99/4As) they can be restored to the p-System format using the standard p-System RECOVER utility.

First create a filler TEXT file six blocks long. Use the Editor to type in about one and one/half full screens of any text you choose. Check its length with the Filer and add or delete lines (from the Editor again) until its directory entry shows 6 blocks. Save this file for use in the recovery process. (More experienced users can skip the steps involving this filler file and use the M)ake command.)

Next Z)ero a fresh disk with a duplicate directory, T)ransfer the six block filler file to it and exit the p-System.

Using any TI disk manager delete the file "PASCAL" (D/F128) which appears to take up the whole disk. Then copy the D/F128 Pascal image files to this disk, writing down the order in which the files are copied (usually alphabetical).

Reboot the p-System, select the Filer and Z)ero the disk again with a duplicate directory. Return to the main Command prompt and e(X)ecute the utility program RECOVER. Answer the prompts, responding Y)es when it asks if important files have not yet been recovered. The program will then search the disk for the blocks of data that precede every p-System file and, if no errors have been made, will construct a valid directory corresponding to the files actually found on the disk. Text files (including your filler file) will be recovered as DUMMY##X which is why you need to know the order the files were copied to the disk. Code files may be recovered under their original names since this name is stored as part of the file data block.

After C)hanging the dummy file names from the FILER, you are ready to Edit, Compile, or print the TEXT files or e(X)ecute the files of compiled CODE.

```
NOTE ON CODE FILES ************************************************
*                                                                  *
*  While the technique described above will reliably recover text  *
*  files, the RECOVER utility often truncates code files.  The     *
*  P-system manual cautions that recovered files have to be relinked *
*  (the RECOVER routine apparently treats any block of embedded nulls*
*  as the end of the file).  However, the whole file IS on the disk  *
*  and can be recreated with the FILER's M)ake command as follows:   *
*                                                                  *
*  First print a directory of the files you downloaded.            *
*                                                                  *
*  filename     size  type   Your RECOVERed |BBSDL:       size      *
*  FASTCOPYC       7  DF128  disk will look |< UNUSED >     4 date 6 *
*  FASTCOPYT      13  DF128  like this ---> |DUMMY01X.TEXT  6  "  10 *
*  TEPC           31  DF128                 |FASTCOPY.CODE  2  "  16 *
*                                           |< UNUSED >     1     18 *
*  -- note that TEP is recovered with the   |TEPHOST.CODE   8  "  19 *
*     name TEPHOST, the name under which    |< UNUSED >   154     27 *
*     it was originally compiled...                                *
*                                                                  *
*  The full length of each Pascal file can be determined by        *
*  subtracting 1 for the TI file header sector and dividing the    *
*  result by 2 (there are 2 128-byte sectors in each 512-byte block).*
*                                                                  *
*  Thus FASTCOPY should be (7-1)/2= 3 blocks                       *
*   and TEPHOST should be (31-1)/2= 15 blocks                      *
*                                                                  *
*      BOTH files have been truncated in recovery.                 *
*                                                                  *
*  Use the M)ake command to add back the truncated blocks -----    *
*                                                                  *
*  Start filling the <UNUSED> gaps from the beginning of the disk. *
*                                                                  *
*    1) M)ake #9:FAKE[4] to fill the space normally used by the    *
*                        duplicate directory (blocks 6-9)          *
*                            ( #9: indicates drive 3 )             *
*                                                                  *
*    2) M)ake #9:FASTCOPY.CODE[3] to extend the recovered file the *
*                                 additional block -- use the SAME *
*                                 file name and respond Y)es to the *
*                                 'Remove old FASTCOPY.CODE?'prompt. *
*                                                                  *
*    3) M)ake #9:TEPHOST.CODE[15] to do the same for TEP.          *
*                                                                  *
*  TIP: If you display the directory before each M)ake operation   *
*       you can both check your progress and have the file names on *
*       the screen for reference as you type in the command.       *
*                                                                  *
*                  WHEN IN DOUBT CONSULT THE MANUAL!               *
*                                                                  *
*******************************************************************
```

ACKNOWLEDGMENTS: The sector read/write routines used in PAS>TI are available from *Genial TRAVelER* diskazine ($30/yr for six 720 sector flippies) — to subscribe, write Barry Traver, Editor, 835 Green Valley Drive, Philadelphia, PA 19128. The XB routine to read data from the p-System directory was adapted from a program by P. E. Schippnick.

PASTRN — PAScal text file TRaNslator

by Jerry Coffey and friends

This program was originally written in response to requests by members of the TI FORUM (CompuServe) for a means of reading p-System text files in a normal TI environment. Such files had been uploaded to the FORUM's Data Libraries using the XMODEM transfer protocol of Andy Cooper's excellent p-System terminal emulator, TEp. There they were stored in binary image form. When downloaded using Paul Charlton's Fast-Term, an exact binary image of these files was written on disk in TI's DISPLAY/FIXED 128 format. When this format is declared in the TI system, each byte is treated as a character — CHR$(0) to CHR$(255) — corresponding to the hexadecimal representation — '00' to 'FF' of the binary digits. Since the record length is fixed at an even divisor of the TI sector length — 128 bytes — no "end-of-record" or filler characters (which would corrupt the binary image) are needed.

PASTRN reads these records, discards the p-System "bookkeeping" information known as the "ZERO PAGE", breaks the records into "lines" at the p-System "end-of-line" character, inserts leading blanks for the p-System indentation codes and writes out the results in DISPLAY/VARIABLE 80 format — the standard TI format for text files.

The original program was fully commented in XBasic but ran very slowly. It was uploaded to the TI FORUM with a plea by the author for help in improving its speed. Andy Cooper responded with an elegant rewrite of the XBasic that provided a 4× improvement in speed. Working independently, Andy Dessoff wrote an assembly language routine which he called PSCAN to perform the critical but slow character handling operations. Finally the author combined PSCAN and the XBasic host program using Todd Kaplan's XBALSAVE technique so that the XBasic and machine language could be saved in a single file that loads and executes very quickly.

Note: Occasionally hybrid programs created with Todd Kaplan's original utility may lockup or give a spurious syntax error depending on the contents of memory when they are loaded and run. If a syntax error message is displayed, just type "RUN". If the display locks up, type **FCTN = (QUIT)**, reset with a widget, or turn the system off and back on. The program will usually run on the second try.

General Note on p-System Text Files

Text files are used for a variety of purposes in the UCSD p-System. They may be source code for programs in Pascal or other languages supported by the p-System, or articles, letters, or other documents. They are distinguished from other p-System file types by information stored in the ZERO PAGE and the special characters which begin and end each line. These special features are created by the p-System Editor and updated when a text file is edited. The p-System Editor and Compiler will not correctly process files in any other format.

SPLIT-P

Disk Partitioner

by Jerry Coffey

This program will partition a disk (in any format your disk controller can handle) into two portions, one of which will be recognized by the UCSD p-System as a valid p-System Volume. The rest of the disk can be used as a normal TI disk. The program requires a previously initialized disk which may be partitioned in any disk drive you choose. You should determine in advance the number of blocks you will need for p-System files. When you enter this number, SPLITP will calculate the number of sectors that will remain for TI files and ask for verification. Then you will be asked to enter the date in the format MM-DD-YY (all numbers e.g. 01 22 86). Be certain that a valid date is entered — it will be recorded in the p-System directory as the date of creation. From this point SPLITP will very rapidly mark out the p-System segment in the bitmap and create a valid ZEROed p-System directory. This directory will show a DATA file called TI-DIR which prevents the p-System from overwriting a block of 20 sectors usually reserved by the TI system for its own directory. Reserving directory space at this location prevents excessive head seeks when reading and writing files to the TI portion of the disk.

```
LIMITATIONS: ************************************************************
*                                                                          *
* Because SPLITP allocates space for P-system and TI directories at  *
* least 10 sectors (excluding sectors 0 & 1) will be allocated to    *
* the DF128 file "PASCAL".  Likewise the program will not accept an  *
* allocation of blocks for P-files that would leave less than 22     *
* sectors for TI files.  The P-directory is in sectors 2 through 11, *
* the TI directory appears in sectors 12 through 31 (after 20 files  *
* the directory is "fractured"), and the space reserved for P-files  *
* begins in sector 32 and runs for 2N consecutive sectors where N is *
* the number of blocks requested   If the CorComp MANAGER is already *
* on the disk, the program will skip over the space from sectors 48  *
* through 145, leaving the MANAGER intact (if requested).  The files *
* TI-DIR and/or MANAGER in the P-system directory and PASCAL in the  *
* TI directory should not be disturbed (use a "cloner" to copy the   *
* whole disk), but any other files in either directory may be handled*
* normally.  P E Schippnick's program to read P-directories from XB  *
* may be used after eliminating the tests for a standard P-system    *
* disk (i.e. "IF [3<>0 then ... and tests on FILENAME$ ).            *
*                                                                    *
**********************************************************************
```

ACKNOWLEDGMENTS: The sector read/write routines used in PAS>TI are available from *Genial TRAVelER* diskazine ($30/yr for six 720 sector flippies) — to subscribe, write Barry Traver, Editor, 835 Green Valley Drive, Philadelphia, PA 19128.

Disk 109. TI-Writer v4.2

```
dskdir. v2.0. 12-dec-96
Disk name               = TIW42/RAG
Sectors total           = 360
Sectors used            = 219
Sectors available       = 139
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 BUGDEMO       3 DIS/VAR 80   >022 002
002 >003 CHARA1        5 PROGRAM      >024 004
003 >004 CHARA2        5 PROGRAM      >028 004
004 >005 EDITA1       33 PROGRAM      >02c 032
005 >006 EDITA2        9 PROGRAM      >04c 008
006 >007 EDITINST     19 PROGRAM      >054 018
007 >008 FORMA1       33 PROGRAM      >066 032
008 >009 FORMINST     12 PROGRAM      >086 011
009 >00a INLOAD        6 PROGRAM      >091 005
010 >00b LOAD          6 PROGRAM      >096 005
011 >00c README        2 DIS/VAR 80   >09b 001
012 >00d TIWMM        12 DIS/FIX 80   >09c 011
013 >00e TIWSEA        9 PROGRAM      >0a7 008
014 >00f TIWV40       60 DIS/VAR 80   >0af 059
015 >010 TIWV41        3 DIS/VAR 80   >0ea 002
016 >011 TIWV42        2 DIS/VAR 80   >0ec 001
```

Disk 109. Contents of file BUGDEMO

This document demonstrates two bugs in the TI Writer Formatter fixed in Version 4.0.

1. The centered line below should have an at sign in it.

```
.SP;CE
Centered @@ At Sign
.SP
```

2. The line below

```
.SP
Formula 2*3*4.
.SP
```

should print as

```
2*3*4
```

and should not prompt for insert 3.

Disk 109. Contents of file README

TI Writer Version 4.0

Use the formatter to print file "TIWV40" on this disk. It fully describes the features of Version 4.0 of TI-Writer.

Disk 109. Contents of file TIWV40

TI WRITER VERSION 4.0

November 1988

Version 4.0 is a major new Fairware release of TI-Writer. The modifications have been extensive in order to remove many of the annoyances of the original package. At the same time, the new version is strictly compatible with the original.

Fairware contributions of $10 can be sent to:

RAG SOFTWARE
R. A. Green
1032 Chantenay Dr.
Gloucester, Ont. CANADA K1C 2K9

Editor Improvements

1. The editor runs independent from the cartridge (or with it if you like).

2. The performance (i.e. speed) of all features has been improved, with special attention to some features.

3. A dramatic improvement in the speed of move, copy and delete lines. Move lines is instantaneous and will never give the "out of memory" condition. Delete lines, in most cases, is also nearly instantaneous.

4. All cursor movement from line to line has been speeded up when no lines are changed.

5. All keyboard input has been speeded up in an attempt to prevent loss of characters when "wrapping" to the next line.

6. A new command, QQ, has been added to exit immediately from the editor without further prompting.

7. A number entered as a command is equivalent to Show line, that is, the line whose number is entered is positioned at the top of the screen.

8. In command mode, simply pressing **ENTER** will return you to edit mode.

9. Two new control codes have been added. **CTRL ,** positions to the top of the file (i.e. Show line 1). **CTRL .** positions to the last line of the file (i.e. Show line E).

10. The CHARA1 file is no longer required to define a new character set for the editor.

11. An install or configuration program is provided that allows you to tailor the editor for your environment. You can:

    a. define your printer for PF,

    b. set the initial screen colours,

    c. set the initial tabs,

    d. set word wrap initially on or off,

    e. set display of line numbers initially on or off,

    f. define the character set.

Editor Notes

1. No features of the Editor have been removed, although SD is different as it was done by code in the cartridge GROM.

2. The text buffer is exactly the same size.

3. As mentioned in the TIW manual but not stressed enough, it is IMPORTANT to immediately do a save file and then reload it after a Recover Edit.

Formatter Improvements

1. The Formatter now runs independent from the cartridge (or with it if you like).

2. The overall performance (i.e. speed) of the Formatter has been improved.

3. There has been a dramatic reduction in the size of the Formatter on disk which makes for faster loading.

4. The formatter's "format commands" can now be entered in upper, lower or mixed case.

5. When multiple format commands are used on a single line (separated by semi-colons) a period can precede all commands.

6. The bug in handling asterisks when not in mailing list mode and the bug in handling double at signs or double ampersands in cenetred lines have been fixed. Try your existing Formatter on the file BUGDEMO.

7. Eight new format commands have been added to the formatter. They are described below.

8. A new response to the prompt, "Pause at end of page?", is provided to make using letterhead or single sheet paper easier. The response "L" indicates that the formatter is to pause at the end of each page and that the normal spacing at the top of the page is not to be done so that letterhead paper can be positioned in the printer at the correct place for the first line of text to be printed. The formatter normally prints two blank lines, the HE line and another blank line at the top of pages. None of these will be printed when the "L" response is used, but the line number will still be set to five.

9. The disk number in the filenames for IF, ML and CH format commands may be specified as an asterisk to indicate the same drive as the main input file. This allows a document to be printed independent of the drive selected by the user.

10. An install or configuration program is provided that allows you to define your printer for the formatter.

Formatter Notes

No features have been removed.

Installing Version 4.0

First, make a working copy of the distribution disk. If you have a character set module that you want installed into the Editor then it must be made available on a separate drive or must be copied onto the working disk. You will be asked for the filename of the character set module during installation.

The two installation programs, EDITINST and FORMINST, can be loaded and run using: E/A Option 5, TIW Option 3, or XB with program INLOAD.

The installation programs prompt for their inputs.

Loading Version 4.0

The new Editor and Formatter can be loaded in a variety of ways. Both programs are now standard "E/A Option 5" programs. Of course, they can be loaded by the TI-Writer cartridge in the normal way, or by using Option 3. The Editor is completely independent and loads all VDP registers and tables. The Formatter requires the VDP set up as for E/A Option 5. The following special loaders are provided to simulate the TI Writer Cartridge Menu for loading the Editor and Formatter.

1. Extended BASIC "LOAD" program

   This program can be easily modified to add other programs to the menu. The first DATA statement has the menu heading and the number of menu items. The other DATA statements have the program environment type, the program name and the menu text for each program on the menu. Each item is in the form:

   "tnnnnnnnnnn ttttttt..."

   where "T" is the environment type the program requires, and is either "E" for E/A environment or "X" for Extended BASIC environment. "nnnnnnnnnn" is the name of the program file to be loaded and must be exactly 10 characters, padded if necessary on the right with blanks. "ttttttt" is the text for the menu display.

   The disk number from which the programs are loaded can also be changed by changing statement 7:

   ```
   CALL LOAD(-123,49)
   ```

   Where 49 is ASCII "1". *Note*: Do not change statements 1 or 2 as the loader is hidden between them.

2. E/A Supercart menu program.

   When program "TIWSEA" is loaded via Option 5 into an E/A Supercart (i.e. RAM at >6000->7FFF) it presents a menu for TI Writer Version 4.0. It also leaves itself as a ROM menu item for consoles that support ROM cartridges.

   Note that the file names entered for Options 3, 4 and 5 are retained and are available for later use. The drive number from which the Editor and Formatter are loaded can be easily patched in TIWSEA either on disk or when loaded into the E/A Supercart.

3. Mini-Memory resident loader.

When program "TIWMM" is loaded via Option 3 into the Mini-Memory and run with program name "TIW" it presents a menu for TI-Writer Version 4.0. Once loaded, the program can be reused until something else is loaded into the Mini-Memory.

Note that the file names entered for Options 3, 4 and 5 are retained and are available for later use. The drive number from which the Editor and Formatter are loaded can be easily patched in TIWMM using Easy Bug. Location >73D5 is the drive number for the Editor and location >73F1 is the drive number for the Formatter.

Distribution Disk Contents

BUGDEMO	Demonstrates two Formatter bugs
CHARA1	A true lower case character set
EDITA1	The Editor, segment 1
EDITA2	The Editor, segment 2
EDITINST	Installation program for the Editor
FORMA1	The Formatter
FORMINST	Installation program for the Formatter
INLOAD	Extended BASIC loader for the installation programs
LOAD	Extended BASIC loader for the Editor and/or Formatter
TIWMM	Mini Memory TI Writer Menu object text, program name: TIW
TIWSEA	E/A Supercart TI Writer Menu
TIWV40	This Version 4.0 writeup

New Format Commands

PRINTER CONTROL
coded as:

```
PC n1,n2,n3,...
```

which causes the control codes n1,n2,n3,... to be sent directly to the printer without changing the line count. This can be used for printer setup without the trouble of TL. Note that like most format commands, PC causes a break in the text and thus cannot be used in the middle of a line.

DEFINE UNDERSCORE CONTROL CHARACTER
coded as:

```
DU n
```

Where n is the number of the code to be used as the underscore begin character. The initial setting for DU is 38, the ampersand.

DEFINE BOLDFACE CONTROL CHARACTER
coded as:

```
DB n
```

Where n is the number of the code to be used as the boldface or overstrike begin character. The initial setting for DB is 64, the at sign.

DEFINE MAILING LIST CONTROL CHARACTER
coded as:

```
DM n
```

Where n is the number of the code to be used as the mailing list insert character. The initial setting for DM is 42, the asterisk.

DEFINE REQUIRED BLANK CHARACTER
coded as:

```
DR n
```

Where n is the number of the code to be used as the required blank character. The initial setting for DR is 94, the caret.

AS IS TEXT BEGIN
coded as:

```
AI
```

This format command is similiar to NF except that the left margin is still observed. As Is text is ended by an FI format command.

CONDITIONAL PAGE EJECT
coded as:

```
CP n
```

A page break will occur if there is less than n lines remaining on the current page; otherwise, the CP is ignored.

CHAIN FILES
coded as:

```
.CF filename
```

When this command is encountered, the main input file is closed, and the user is prompted to insert the disk for the named file. When the user presses **ENTER**, the named file is processed. This allows changing of disks and the processing of an unlimited length document. Note this command is invalid in an "included" file (just as IF commands may not be nested). In addition, only a single copy of the document will be printed when chained files are used.

REFERENCE SHEETS

The complete list of commands and function codes for both the Editor and the Formatter is given below.

Editor Commands

CMD	*FUNCTION*	*PARAMETERS*
C	Copy lines	start stop after
D	Delete lines	start stop
DF	Delete file	filename
E	Edit mode	--
F	Files help	--
FS	Find string	[startcol] [endcol] /string/
L	Lines help	--
LF	Load file	[after][start][end] filename
M	Move lines	start stop after
P	Purge text	Y|N
PF	Print file	[C][L][start][stop] printername
Q	Quit	E|P|S
QQ	Quick quit	--
RE	Recover Edit	Y|N
RS	Replace String	[startcol][endcol] /old/new/
S	Show line	number
SD	Show Directory	disknumber
SF	Save File	[start][stop] filename
SH	Search help	--
T	Tabs	I|T|L|R
n	Line n	--

Editor Codes

FCTN	*ACTION*
1	Delete Character
2	Insert Character
3	Delete Line
4	Roll Down
5	Next Window
6	Roll Up
7	Tab
8	Insert Line
9	Command Mode
0	Line Numbers Toggle
=	Command Mode
E	Cursor Up
D	Cursor Right
S	Cursor Left
X	Cursor Down

CTRL	*ACTION*
1	Oops
2	Reformat
3	Screen Colors
4	Next Paragraph
5	Duplicate Line
6	Last Paragraph
7	Word Tab
8	New Paragraph
9	New Page
0	Word Wrap Toggle
A	Roll Down
B	Roll Up
C	Command Mode
D	Cursor Right
E	Cursor Up
F	Delete Character
G	Insert Character
H	Last Paragraph
I	Tab
J	Next Paragraph
K	Delete to End of Line
L	Home Cursor
M	New Paragraph
N	Delete Line
O	Insert Line

P	New Page
Q	----
R	Reformat
S	Cursor Left
T	Back Tab
U	Special Character Mode
V	Beginning of Line
W	Word Tab
X	Cursor Down
Y	Left Margin Release
Z	Oops
comma	Show Line 1
period	Show Line E

Formatter Commands

CMD	*FUNCTION*
@@	Begin Boldfaceing (See DB)
&&	Begin Underscoring (See DU)
caret	Required Blank (See DR)
*	Mailing List Variable (See DM)
c/r	Break Text
p/a	Break Page
AD	Begin Right Margin Justification
AI	Begin As Is Text
BP	Break Page
BR	Break Text
CE n	Center n Lines
CH filename	Chain Files
CO text	Comment
CP n	Conditional Page Break
DB n	Define Boldface Character
DM n	Define Mailing List Variable Char
DP n:text	Define Mailing List Prompt
DR n	Define Required Blank Character
DU n	Define Underscore Character
FI	Begin Text Filling
FO text [%]	Page Footer
HE text [%]	Page Heading
IF filename	Include File
IN [+|-]n	Indent
LM [+|-]n	Left Margin
LS n	Line Spacing
ML filename	Mailing List File

NA	No Right Margin Justification
NF	No Text Filling
PA [+|-] n	Set Page Number
PC n1,n2,...	Printer Control
PL [+|-] n	Page Length
RM [+|-] n	Right Margin
SP n	Space n Lines
TL n:n1,n2,...	Transliterate Character

Disk 109. Contents of file TIWV41

Fixes for Version 4.1

1. Editor did not perform tabs.

2. Formatter did not respond to up arrow processing user input.

3. Formatter did not process end of file properly for .IF files.

4. Formatter did not process user input of page/letter numbers properly.

Note: The installation programs have also been modified for Version 4.1

Disk 109. Contents of file TIWV42

Fixes for Version 4.2

1. Editor in move lines when moving a single line to itself. For example: M 2,2,1

Disk 110. c99/9640.

Run c99 out of MDOS.

```
dskdir. v2.0. 12-dec-96
Disk name               = C99/9640
Sectors total           = 360
Sectors used            = 350
Sectors available       = 8
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       9 DIS/VAR 80   >022 008
002 >003 ASM           9 PROGRAM    Y >02a 008
003 >004 ASSM1        33 PROGRAM    Y >032 032
004 >005 ASSM2        20 PROGRAM    Y >052 019
005 >006 AUTOEXEC      2 DIS/VAR 80 Y >065 001
006 >007 C99C         58 PROGRAM    Y >066 057
007 >008 C99D         44 PROGRAM    Y >09f 043
008 >009 C99LIB       64 DIS/VAR 80 Y >0ca 063
009 >00a CB           45 PROGRAM    Y >109 044
010 >00b CLINT         2 DIS/VAR 80 Y >135 001
011 >00c CTYPE_H       2 DIS/VAR 80 Y >136 001
012 >00d LDR          12 PROGRAM    Y >137 011
013 >00e QDE          42 PROGRAM    Y >142 038 >00f 003
014 >012 STDIO_H       4 DIS/VAR 80   >013 003
015 >016 STDLIB_H      2 DIS/VAR 80 Y >017 001
016 >018 STRING_H      2 DIS/VAR 80 Y >019 001
```

Disk 110. Contents of file -README

This disk contains an almost complete set of functions for using c99 from MDOS. The programs and functions are almost all from Clint Pulley, but I have collected them piece by piece over the last several months.

1. QDE
   (Quick and dirty editor) loads from MDOS

   ```
   QDE {file name}
   ```

   To save, press <ESC> twice

2. C99C
   Compiler name files

   ```
   C99C {c file} {source file}
   ```

   By adding '/t' at end of line, you can include c.code in s.code

3. ASM
   ASSEMBLER. Uses TI ASSM1 and ASSM2

   ```
   ASM {s.code} {o.code} /{options}
   ```

4. CLINT
   This does these steps automatically!

   ```
   CLINT {c.code} {o.code}
   ```

   The c.code is loaded into the editor so you can check it, then compiled onto the 9640 RD, and assembled into the o.code!!

5. LDR
   This takes your o.code, picks up the functions you need from the lib C99LIB, and runs it!!

   ```
   LDR {o.code},C99LIB /*note commas!!*/
   ```

   If you add a program name at end, a program that will run directly from MDOS is created. IT IS NO LONGER NECESSARY TO LOAD CSUP; it is loaded routinely from LIB!!

   Ex.
   ```
   A:LDR B:prog_o,A:C99LIB B:program
   ```

6. CB
"c beautifier". Makes c code look properly indented and easier to follow

```
CB {c.code1} {c.code2}
```

7. STDIO_H, STRING_H, etc.
Header files; if you use them as "include files" use quotation marks, or they will be truncated!

STDIO_H	includes fopen,fclose,fread,sprintf,scanf,etc.
STDLIB_H	includes malloc,atoi,itoi,random,etc,
STRING_H	includes strlen,strcmp,stncmp,strcpy,etc

Remember, all this c99 material is FAIREWARE; if you use it, send a check to:

Clint Pulley
38 Townsend Ave.,
Burlington,Ont
CANADA L7T 1Y6

Donald L. Mahler, TIUG, BCS

Disk 111. Kirkwood Math Routines for c99

```
dskdir. v2.0. 12-dec-96
Disk name               = MATHFN
Sectors total           = 360
Sectors used            = 124
Sectors available       = 234
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >004 DOC          11 DIS/VAR 80   >091 010
002 >002 MATH        111 DIS/FIX 80   >022 110
003 >003 MATHI         2 DIS/VAR 80   >090 001
```

Disk 111. Contents of file DOC

DOCUMENTATION FOR MATHEMATICAL ROUTINES

Charles E. Kirkwood, Jr.
Box 1241
Clemson, SC 29633

merged with

'C' FLOATING POINT LIBRARY

Tom Bentley
324 Cambridge St. Apt. 207
Ottawa, Ont.
Canada K1R 7B5

This is an object file of mathematical routines, most of which have appeared in issues of *MICROpendium*. Several minor changes have been made to speed up execution. Since Tom Bentley's Floating Point Library is necessary for these routines, the routines are added to Tom's library and then compiled and assembled to obtain this object file called MATH.

The include file MATHI also includes Tom's FLOATI include file.

When the Mathematical Routines are used, it is not necessary FLOAT or FLOATI with your program.

All of the variables are floating-point unless stated otherwise. The routines that are included are:

`exp(x,r)`
: The natural base e to the x power, result is r.

`ln(x,r)`
: The natural logarithm of x, result is r.

`ax(a,x,r)`
: Float-point a to the floating-point x power, result is r.

`an(a,n,r)`
: Floating-point a to the integer n power, result is r.

`sqrt(a,r)`
: Square root of a, result is r.

`root(n,a,r)`
: The nth root of a, result is r.

`abs(x,r)`
: Absolute value of x, result is r.

`degrad(d,r)`
: Convert from degrees d to radians r.

`raddeg(r,d)`
: Convert from radians r to degrees d.

`sin(x,s)`
: Sine of x, result is s.

`cos(x,c)`
: Cosine of x, result is c.

Disk 111. Contents of file MATHI

```
/* MATH Include File */

#asm
 REF ITOF,FTOI,STOF,FTOS,FEXP,FCOM,FPGET,FPPUT,FINT,FCPY
 REF EXP,LN,AX,AN,ROOT,SQRT,DEGRAD,RADDEG,SIN,COS,ABS
#endasm

#define float     char
#define floatlen  8
```

Specialty Disks

These disks contain programs and files that are not of general interest and/or are not as "cleaned-up" as the materials we put on regular BCS disks. They do work and should prove useful to anyone interested in the areas they cover.

Specialty Disk 1. Load Interrupt Demos

Modifications to several routines from BCS disk #1, to the assembler on BCS disk #2, and some other programs to allow them to work with the load interrupt switch. By J. Peter Hoddie. Requires EA.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-LI
Sectors total           = 360
Sectors used            = 280
Sectors available       = 78
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >00f -README       7 DIS/VAR 80   >126 006
002 >002 DEBUG        32 DIS/FIX 80   >022 031
003 >003 DISASSM      20 DIS/FIX 80   >041 019
004 >004 DISASSM/S    80 DIS/VAR 80   >054 079
005 >005 DSRLNK/S      9 DIS/VAR 80   >0a3 008
006 >006 INIT         13 DIS/FIX 80   >0ab 012
007 >007 INIT/S       47 DIS/VAR 80   >0b7 046
008 >008 RECSCR        4 DIS/FIX 80   >0e5 003
009 >009 RECSCR/S     13 DIS/VAR 80   >0e8 012
010 >00a RECSCR/S1    12 DIS/VAR 80   >0f4 011
011 >00b SAVSCR        4 DIS/FIX 80   >0ff 003
012 >00c SAVSCR/S     13 DIS/VAR 80   >102 012
013 >00d SAVSCR/S1    13 DIS/VAR 80   >10e 012
014 >00e SCREEN       13 PROGRAM      >11a 012
```

Specialty Disk 2. GRAM Kracker Stuff

Contains version of Fast-Term and Disk Manager 1000 that load into the GRAM 1 and 2 space to replace TI BASIC on main menu. Also a version that puts both of them in the cartridge space together. Several files on GRAM Kracker modifications from TI Forum on CompuServe.

```
dskdir. v2.0. 12-dec-96
Disk name               = BCS-JPH-GK
Sectors total           = 360
Sectors used            = 274
Sectors available       = 84
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README       7 DIS/VAR 80   >022 006
002 >003 CART         12 DIS/FIX 80   >028 011
003 >004 CART/A       41 DIS/VAR 80   >033 040
004 >005 DM1000/G1    34 PROGRAM      >05b 033
005 >006 DM1000/G2    34 PROGRAM      >07c 033
006 >007 FAST/G1      34 PROGRAM      >09d 033
007 >008 FAST/G2      34 PROGRAM      >0be 033
008 >009 GPLMOV/GKT   29 DIS/VAR 80   >0df 028
009 >00a GRAM/0       34 PROGRAM      >0fb 033
010 >00b INIT/GKT      7 DIS/VAR 80   >11c 006
011 >00c NOAUT2/GKT    8 DIS/VAR 80   >122 007
```

Specialty Disk 3. Horizon RAM Disk Menu Operating System (1)

This is an operating system for the Horizon's RAM disk created by John Johnson. It allows you to create a menu of programs to be presented at power up along with whatever cartridge is in place. In effect, you can create you own main menu of programs that you regularly use.

```
dskdir. v2.0. 12-dec-96
Disk name               = S3
Sectors total           = 360
Sectors used            = 333
Sectors available       = 25
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 -README      14 DIS/VAR 80   >022 013
002 >005 MENT;SRC     24 DIS/VAR 80   >04c 023
003 >006 MENU         21 PROGRAM      >063 020
004 >007 MENU;DOC     75 DIS/VAR 80   >077 074
005 >008 MENU;SRC1    38 DIS/VAR 80   >0c1 037
006 >009 MENU;SRC2    98 DIS/VAR 80   >0e6 097
007 >00a MENUA_05     33 PROGRAM      >147 032
008 >00b MENUB_05     17 PROGRAM      >167 001 >00c 015
009 >003 VER_MENU     13 DIS/FIX 80   >02f 012
```

Specialty Disk 4. Horizon RAM Disk Menu Operating System (2)

```
dskdir. v2.0. 12-dec-96
Disk name               = S4
Sectors total           = 360
Sectors used            = 184
Sectors available       = 174
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 EDIT1        33 PROGRAM      >022 032
002 >003 EDIT2         6 PROGRAM      >042 005
003 >004 EDIT;DOC      6 DIS/VAR 80   >047 005
004 >005 MGR1         33 PROGRAM      >04c 032
005 >006 MGR2         31 PROGRAM      >06c 030
006 >007 TERM         33 PROGRAM    Y >08a 032
007 >008 TERN         33 PROGRAM    Y >0aa 032
008 >009 TERO          9 PROGRAM    Y >0ca 008
```

Specialty Disk 5. Kroll GRAM Disk

A collection of GRAM files and GPL source code for those interested in getting the most out of their GRAM device.

```
dskdir. v2.0. 12-dec-96
Disk name               = KROLL
Sectors total           = 360
Sectors used            = 348
Sectors available       = 10
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 GEM10*G0      7 DIS/FIX 80   >022 006
002 >003 GEM10*G0/S   15 DIS/VAR 80   >028 014
003 >004 GEM10*G6      7 DIS/FIX 80   >036 006
004 >005 GEM10*G6/S   15 DIS/VAR 80   >03c 014
005 >006 GEM10*G7      7 DIS/FIX 80   >04a 006
006 >007 GEM10*G7/S   15 DIS/VAR 80   >050 014
007 >008 GPL*LOADER   19 PROGRAM      >05e 018
008 >009 GRAMMY/CAT   19 DIS/VAR 80   >070 018
009 >00a OKI92*G0      8 DIS/FIX 80   >082 007
010 >00b OKI92*G0/S   17 DIS/VAR 80   >089 016
011 >00c OKI92*G6      7 DIS/FIX 80   >099 006
012 >00d OKI92*G6/S   17 DIS/VAR 80   >09f 016
013 >00e PRGS*BY*MK   20 DIS/VAR 80   >0af 019
014 >00f RAG*ALTER    27 DIS/FIX 80   >0c2 026
015 >010 RAG*ALTER1   43 DIS/VAR 80   >0dc 042
016 >011 XB*TITLE      7 DIS/FIX 80   >106 006
017 >012 XB*TITLE/L   68 DIS/VAR 80   >10c 067
018 >013 XB*TITLE/S   30 DIS/VAR 80   >14f 025 >014 004
```

Specialty Disk U1. Infocom Games Rapid Loader

Do you play Infocom adventure games. Doesn't it seem they take forever to load? This disk lets you load your favorite Infocom game in 1/3 the time.

```
dskdir. v2.0. 12-dec-96
Disk name               = INFOCOM/RL
Sectors total           = 360
Sectors used            = 336
Sectors available       = 22
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CUTTHROATS   14 PROGRAM    Y >022 013
002 >003 DEADLINE     14 PROGRAM    Y >02f 013
003 >004 DSKCAT        7 PROGRAM    Y >03c 006
004 >005 ENCHANTER    14 PROGRAM    Y >042 013
005 >006 FILE/L2       9 PROGRAM    Y >04f 008
006 >007 HITCHHIKER   14 PROGRAM    Y >057 013
007 >008 INFIDEL      14 PROGRAM    Y >064 013
008 >009 LOAD         16 PROGRAM    Y >071 015
009 >00a MENU         33 PROGRAM    Y >080 032
010 >00b PART2         8 DIS/VAR163 Y >0a0 007
011 >00c PLANETFALL   14 PROGRAM    Y >0a7 013
012 >00d RLOAD/DOCS   15 DIS/VAR 80 Y >0b4 014
013 >00e SAMPLER      14 PROGRAM    Y >0c2 013
014 >00f SORCERER     14 PROGRAM    Y >0cf 013
015 >010 STARCROSS    14 PROGRAM    Y >0dc 013
016 >011 SUSPENDED    14 PROGRAM    Y >0e9 013
017 >012 UTIL1        33 PROGRAM    Y >0f6 032
018 >013 UTIL2        19 PROGRAM    Y >116 018
019 >014 WITNESS      14 PROGRAM    Y >128 013
020 >015 ZORK-I       14 PROGRAM    Y >135 013
021 >016 ZORK-II      14 PROGRAM    Y >142 013
022 >017 ZORK-III     14 PROGRAM    Y >14f 013
```

Specialty Disk U2. Artist Converter

Artcon+ is basically an Extended Basic to TI-Artist Converter. That and more!

```
dskdir. v2.0. 12-dec-96
Disk name               = ARTCON+
Sectors total           = 360
Sectors used            = 355
Sectors available       = 3
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 ARTCON       14 DIS/FIX 80 Y >022 013
002 >003 ARTCON/DOC   24 DIS/VAR 80 Y >02f 023
003 >004 DSKCAT        7 PROGRAM    Y >046 006
004 >005 FILE/L2       9 PROGRAM    Y >04c 008
005 >006 LOAD          9 PROGRAM    Y >054 008
006 >007 MAX-RLE2     42 PROGRAM    Y >05c 041
007 >008 PICTURE1_C   25 PROGRAM    Y >085 024
008 >009 PICTURE1_P   25 PROGRAM    Y >09d 024
009 >00a PICTURE2_C   25 PROGRAM    Y >0b5 024
010 >00b PICTURE2_P   25 PROGRAM    Y >0cd 024
011 >00c READ/FIRST    5 DIS/VAR 80 Y >0e5 004
012 >00d SCREEN/DOC    7 DIS/VAR 80 Y >0e9 006
013 >00e SCREENS       3 DIS/VAR163 Y >0ef 002
014 >00f SHOW/DOC     10 DIS/VAR 80 Y >0f1 009
015 >010 SHOWSPRITE   15 DIS/VAR163 Y >0fa 014
016 >011 SPRITE/DOC   14 DIS/VAR 80 Y >108 013
017 >012 SPRITES       7 DIS/VAR163 Y >115 006
018 >013 WOOD/DRAW1    7 DIS/VAR163 Y >11b 006
019 >014 WOOD/DRAW2    7 DIS/VAR163 Y >121 006
020 >01e WOODSTOCK2   75 INT/VAR254 Y >127 065 >015 009
```

Specialty Disk U3. 1000 Words

If you want to have documents containing sections of text alternating with sections of graphics, 1000 Words gives you the capability. 1000 Words converts picture files from TI-Artist to TI-Writer format.

```
dskdir. v2.0. 12-dec-96
Disk name               = U3
Sectors total           = 360
Sectors used            = 281
Sectors available       = 77
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 40
Number of sides         = 1
Density                 = single

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 DEMOL_C      25 PROGRAM      >022 024
002 >003 DEMOL_P      25 PROGRAM      >03a 024
003 >004 DEMOPT1       2 DIS/VAR 80   >052 001
004 >005 DEMOPT3       2 DIS/VAR 80   >053 001
005 >006 DEMO_C       25 PROGRAM      >054 024
006 >007 DEMO_P       25 PROGRAM      >06c 024
007 >008 DOCS-PF      97 DIS/VAR 80   >084 096
008 >009 PON           2 PROGRAM      >0e4 001
009 >00a PRON          2 DIS/VAR 80   >0e5 001
010 >00b WORDS        33 PROGRAM      >0e6 032
011 >00c WORDT        33 PROGRAM      >106 032
012 >00d WORDU        10 PROGRAM      >126 009
```

Specialty Disk U4. Dreadnought

Most of us learned this game in school. Sometimes called Battleship, the object is to sink the other guy's ships before he sinks your. The disk also contains a color-code game, a lottery generator, and a calendar program to display or print any month from 1583 to 9999.

```
dskdir. v2.0. 12-dec-96
Disk name               = U4
Sectors total           = 360
Sectors used            = 275
Sectors available       = 83
Sectors/track           = 9
Disk formatted (DSK)    = yes
Disk Manager protection = no
Tracks per side         = 0
Number of sides         = 0
Density                 = unknown

No. FDR  Filename   Size Type       P
--- ---- ---------- ---- ---------- -
001 >002 CALENDAR      4 PROGRAM      >022 003
002 >003 CALENDAR1    57 INT/VAR254   >025 056
003 >004 CALENDAR2    21 PROGRAM      >05d 020
004 >005 CALENDAR3    18 PROGRAM      >071 017
005 >006 CRYPTO       27 PROGRAM      >082 026
006 >008 DREDNOT1     21 PROGRAM      >0c9 020
007 >009 DREDNOT2     48 INT/VAR254   >0dd 047
008 >00a DREDNOT3      5 PROGRAM      >10c 004
009 >00c DREDNOT4      4 DIS/FIX 19   >117 003
010 >00b DREDNOT5      8 DIS/FIX 28   >110 007
011 >00d LOAD         12 PROGRAM      >11a 011
012 >00e README        4 DIS/VAR 80   >125 003
013 >007 RNDLOTTO     46 PROGRAM      >09c 045
```

Cross-Reference

Assembly Utilities

4 Super-Bug Debugger
9&10 TI-Forth Source Code
41 PULSAR Utilities
42 Universal Disassembler
50 RAG Assembler
51 RAG Assembler
56 Memory Manipulator
77 RAG Linker
78 Clint Pulley Disk
91 9640 Technical Material

C99

27 C99
40 C Tutorial
45 c99 Library
68 C Programs
88 c Libraries
98 c99 Windows
106 c99 for MDOS

Disk Utilities

12 Masscopy
13 Disk Manager 1000
24 Disk Utilities
43 Disk Utilities 11
48 Disk Manager 99
55 Disk + Aid
85 Catalog Library
87 Disk Utilities

Forth

14 XB-Forth
30 Forth Programs #1
31 Forth Programs #2
39 Forth Tutorial
75 TI-Forth Demo

Games

8 TI Basic Games
23 Games-n-Graphics
35 Extended BASIC Games
38 Assembly Language Games
49 More Games
52 JPH Games Disk
69 Colossal Caves
90 XB Games
103 GEE

Graphics

36 Speech/Graphics Demos
37 Graphics Demos
54 RLE Graphics
59 J.P. Graphics
62 RLE Pictures #1
63 RLE Pictures #2, Famous Logos
64 RLE Pictures #3, Space and Science Fiction
65 RLE Pictures #4, Famous Folks
89 Picasso
102 TASS2001

Music

32 Music Programs
33 Music Programs #2
44 Sorgan
53 Music Compiler
66 Music Disk 3
67 Music Disk 4
73 TI-Sings
74 Pop Music Demo
84 South Pacific

Pascal (p-Code) System

95 p-Code Units by Anders Persson
96 p-Code Units - Source 1
97 p-Code Units - Source 2
108 p-Code Utilities

Telecommunications

7	Terminal Emulator Programs
18	Fast-Term
70	Mass Transfer
80	Omega and Archiver 11
10	Telco — Program Disk

XB Utilities

1	XB Assembly Language Routines
2	Disk Copy and Disassemble
11	Graphics Design System
16	Neatlist
17	Screen Dump
19&20	Sprite Builder
47	Disk One
71	STAR, Super TI Assembly Routines
72	TI-Keys
76	Screen Enhancement Package
81	BASIC Builder
82	Textload/EA5Load
86	XB Tools

9640 disks

92	9640 Terminal Emulators
93	9640 PR Base
94	9640 Routines by JP Hoddie
107	MDOS Quick & Dirty Development System