The MANNERS Journal of TI Computing

Mar–Apr, 89

BASIC

FORTRAN–99

ASSEMBLY

Extended BASIC

C99

In This Issue:

MANNERS NEWSLETTER Mar - Apr '89

MANNERS President: Jerry Coffey
Membership Chairman: Bill Howard
Newsletter Editor: Dave Ramsey

TI Vendors wishing to place ads in The MANNERS Journal of TI Computing are encouraged to contact the editor at (301) 757-8413, evenings.

The theme of the May-June issue will be HARDWARE! (IF I can get any of you hardware types to write!)

Table of Contents - Mar/Apr Issue:

****

This month's newsletter has fewer but longer articles. Hopefully, everyone will find them of use and learn something from them.

The user group would be very interested in any articles on Forth, Fortran-99, C99, using TI Writer, using Multiplan, and other applications on the TI-99/4A or the Myarc 9640. We are also interested in articles describing hardware add-ons, upgrades, and modifications to improve our TI machines,

If there is a topic you would like to see covered in the MANNERS Newsletter, write to the editor at the new PO Box address on the back cover. Or write the article yourself and submit it on disk. I'll be sure to dress it up and put your name in lights! Hope to hear from you soon! - The Editor.

The Washington DC Area TI Home Computer User Group meets monthly. The regular meeting night is the second Thursday of each month. The May meeting will be held on THURSDAY, the 11th of May at the FAIRFAX HIGH SCHOOL. The June meeting is scheduled for THURSDAY, the 8th of June also at the FAIRFAX HIGH SCHOOL. The topic for May has not been announced. Also, the June topic has not been announced. For directions or other information, call Jim Horn at (301) 340-9617 or Jerry Coffey at (301) 928-1217.

Membership Dues are $16.00 per year and may be sent to the MANNERS Membership Chairman, Bill Howard, at 15204 Louis Mill Drive, Chantilly, VA. 22021. Bill can be reached by phone at (703) 378-1090.

The Mid-Atlantic Ninety-Niners Computer User Group is dedicated to helping users of TI-99/4A computers and all related 4A offspring such as the TI-99/2, the TI-99/8 and the Myarc 9640 computer. We also would be happy to welcome into our group users of any other TMS9900 based microprocessor system.

This publication is available to all MANNERS members as part of their annual membership dues. Any other TI-99/4A Home Computer User may also join The MANNERS User Group at the annual dues rate of $16 per year. MANNERS would also like to make arrangements to exchange newsletter with your user group. Newsletter editors for other TI user groups are encouraged to contact either Mark Georges at the club mailing address or Dave Ramsey at (301) 757-8413 to make appropriate arrangements.

Remember! The MANNERS Newsletter is a USER SUPPORTED PUBLICATION. If you don't write anything, you won't get anything!

A Comparison of Languages
by Ed Hall

Since the theme for this journal is languages, I thought I would show a comparison of how to perform a particular routine by writing the steps involved in three different languages. I chose TI BASIC, TI EXTENDED BASIC (XB) , and ASSEMBLY using the TI EDITOR/ASSEMBLER (E/A). In order to run the programs I have included, it will be necessary to have a joystick which operates as number 1. To see the entire comparison it will also be necessary to have on hand EXTENDED BASIC AND EDITOR/ASSEMBLER packages.

In writing programs it is important to plan out the functions that you wish to perform as completely as possible. Figuring out how to fill in all the holes and squish bugs will be tasking enough while writing, so we should start with a pretty well-defined set of parameters. The parameters I have chosen will remain the same for each program. This is a must in comparisons. First I want to place a 0 somewhere near the middle of the screen. Next I want the 0 to be movable by the joystick in all eight directions. I also want the 0 to leave a trail of asterisks behind as it moves, and flash as it rests in one spot. Last, I want the asterisks to change color by stepping through all sixteen available upon the press of the fire button. Throughout each program, I want all action to be able to occur simultaneously.

Now that we have the basics chosen, let's move on to the writing. Remember that these programs are just what came out of MY head, so later if you would like, go ahead and write some of your own to compare their operation with these. The program listings can be found at the end of the article. The first program is in TI BASIC and as you will notice, it consists of 26 statement lines. Go ahead and enter this program into your system and save it. Then see how it runs and check to see if it meets all the parameters above.

Lines 100 and 110 set variables which will later be used to place the 0/*s on the screen. Line 120 invokes a resident subroutine to check the joystick port for activity. In basic "CALL" is used to run resident subprograms similar to using GOSUB ... to run subprograms within your main program. In this instance the 1 within the parentheses tells the routine to check joystick number 1 and the X and Y will be where the results are placed. Line 130 is used to allow the picking up of fire button activity. This is very poorly documented, but to check for the activation of a fire button, you must perform a key scan using split keyboard and search for an occurrence of 18 in the K variable. Line 140 places an asterisk on the screen. Line 150 modifies the variable X1 which is used in the horizontal placement of the 0/*s. The variables returned by the JOYST routine are 4/0/-4 so X/4 is used to modify X1. Line 160 is a trap in case X1 leaves the boundaries of the screen. Line 170 is the same as 150 except it modifies the Y1 variable. Line 180 is a trap for Y1 as 160 was for X1. Line 190 places the 0 using the modified coordinates. Line 200 checks to see if the fire button was pressed and if not sends the operation back to 120 where the loop continues. If the fire button was pressed, it falls through to 210. Line 210 increments the variable C and Line 220 checks to insure the color will be in bounds. If it is, it skips to line 240, but if not C is reset to 1 in 230 before moving to 240 where the COLOR subroutine is CALLed. Line 250 sends the program back as did 200. Lines 260 - 300 insure X1 stays within the bounds of the screen and allow direct wraparound. Lines 310 - 350 do the same for the Y1 coordinate.

At this time let's start up the EXTENDED BASIC. After starting up XB, load in the program from above and run it. Could you tell a difference in speed? You should have noticed a marked improvement in the movement of the 0 and the change in color, even though it's the same program that was run in BASIC. Still it can be improved. XB adds the ability to create multiple lines which is both a memory saver and time saver in running a program. Our next step is to take advantage of this new ability. The program lines under listing #2 are more compressed, yet the program runs the same way. This XB program takes only 9 lines to equal 26 in BASIC.

Line 100 now performs the operations of two lines from the first program. Notice that C had to be initialized in this file. In the first program C was able to start off as 0 but in this one it has to be greater than 0 to work. Line 110 represents lines 120 - 160 of the first program. Line 120 represents lines 170 and 180. Line 130 represents lines 190 - 230. Line 140 is equal to lines 240 and 250. Line 150 represents 260 290. 160 equals 300 and line 170 replaces 310 - 340 with 180 equalling 350.

Once you've keyed this one in, see how it compares to the other one when run. Did it run faster? It should have, but it still isn't really fast, is it? Well maybe we can speed it up a little more in a while.

Let's regress for a moment and bring back up BASIC with either the MINIMEM or E/A in place and load in the original program from above. If you are using BASIC while the MINIMEM or E/A cartridge is in place additional resident subroutines are available. Among them is an alternate way to place characters on the screen called POKEV. CALLing POKEV (place, character ASCII#+96) writes the character directly into VDP RAM where the screen information is. Place is a number from 0 to 767 which directly represents a screen location from the upper left corner to the lower right corner respectively. The character is whatever you choose to write to the screen. To alter the program to support this, change these two lines:

```
140 CALL POKEV(Y1*32+X1-1,138)
190 CALL POKEV(Y1*32+X1-1,144)
```

The expression is equivalent to using HCHAR even though the position is represented by a single number. Before we move on, make a mental note of the speed with which the 0 can be moved around the screen. Now that we've seen what a little programming in BASIC or XB can do to meet our parameters, let's see how to accomplish them in ASSEMBLY.

First off when using ASSEMBLY, we must take into account many more things than we do in BASIC. We need to access certain memory locations directly and place and manipulate data in precise ways. There are subroutines but they must called only after we have data in the correct places. You'll notice first that the SOURCE code has almost three times as many lines as the BASIC version. However, when this is ASSEMBLED it will run much faster and take up less space in memory. To run the following program you will have to type the whole program into a D/V 80 file as it is written and ASSEMBLE it using the E/A. If you are unfamiliar with the procedures for running the ASSEMBLER, after inputting and saving the text SOURCE code, perform these steps: 1) Plug in the E/A cartridge and press 2 twice to bring up the E/A menu; 2) With the E/A disk in drive #1, press 2 for ASSEMBLE; 3) Press Y at the LOAD ASSEMBLER? prompt and enter the file name you saved the text under as the SOURCE FILE NAME; 4) Enter the name you want the finished file to be called under the OBJECT FILE NAME; 5) If you want a listing of the code, enter a filename or printer under LIST FILE NAME; 6) Enter at a minimum the letter R under options. You might also add the letters CSL but do not add L or S if you've not given a name for the list. The above steps can be found in chapter 2.2 of the E/A manual.

Before we get into the program itself let me give a little extra info for the beginners to ASSEMBLY. The syntax of BASIC lines must be met in writing programs in those languages. In ASSEMBLY this same rule applies, however the syntax is much more straightforward. For an explanation of the syntax refer to chapter 3.3 of the E/A manual. Whether you understand how the program works or not, go ahead and try to run it so you can compare its operation with those written in the BASIC languages. The source code is listed as PROGRAM LISTING #3 at the end of this article. When you type it in please observe spaces where they exist and do not place spaces where they don't exist in the code.

It would be much too time consuming and take up too much newsletter to describe in detail the lines of code for this ASSEMBLY language program. Instead I will describe the functions of areas in a more general sense. If further details are needed perhaps a discussion of the code can be arranged at one of our meetings.

The first few lines define the name the program will run under when loaded and set up the parameters for use within the program. Two resident routines will be used. These are VDP Single Byte Write (VSBW) and Keyboard SCAN (KSCAN). The first will place all characters on the screen and the second accepts input from the joystick and keyboard. Since ASSEMBLY does not automatically allow for a FCTN-4 to break program execution, I have allowed for the pressing of letter A to cause the program to stop. As used in this program, the letter A will actually send the computer to the title screen.

On the extreme left of the file you will see words like JOY and KEY. These are known as labels and must start in the first column of the text. The two lines at JOY set up values that will place the initial 0 near the center of the screen. The KEYSRC lines calculate the value to use in the placement of the 0. The first time through the values loaded at JOY are used but from then on the values created by the program are used. The lines at KEY actually place the 0 on the screen and read the input from the keyboard or joystick. The only inputs accepted from the keyboard are Q which equals the fire button and A which causes program termination. The only other inputs accepted are from the joystick. One of the inputs accepted is the fire button. Remember, the fire button is being used to change the color of the asterisks. The lines at KEY2 do the color change based on values from KEY. The lines at MOVCUR place the asterisks on the screen in place of the 0 and determine how to move the cursor if a direction has been given from the joystick. The lines at ADDY, XCHK and ADDX set up the values to be used by the next pass of KEYSRC in placing the next 0 on the screen. These routines also take care of wraparound. The last line of program execution is the EXIT routine which tells the computer to go back to the title screen if the letter A was pressed in KEY above. The statement END under the EXIT label has no function within the program, but is necessary to tell the ASSEMBLER to stop processing the file.

In order to run this program once it is ASSEMBLED, you will need to use option 3, LOAD AND RUN from the E/A menu. At the FILE NAME? prompt enter the drive and filename you have given the ASSEMBLED code and then press ENTER again. At the PROGRAM NAME prompt enter the name JOY. The program will immediately start. After running it for a while to check operation, press A to stop and return to the title screen. If you would like to run this from basic, you will need to change 3 lines in the source code and reASSEMBLE the file. The 3 changes are:

```
    KEY    change >3000   to >9000
    KEY2   change >0385   to >0311
    MOVCUR change >2A00   to >8A00
```

With E/A or MINI-MEMORY in place you can use the following program in BASIC to run the ASSEMBLY program:

```
    10 CALL INIT
    20 CALL LOAD("DSK?.???????")
    30 CALL LINK("JOY")
```

* ?.??????? is the drive and filename you gave to the ASSEMBLED file.

As you can see, there are more ways to write a program than there are languages. You can even combine a BASIC program with some ASSEMBLY routines held in memory and perform CALL LINKs to use them. Let me close with a thought for those who own MINI MEMORIES. How could the ASSEMBLY source code be changed to be able to type it into the line-by-line ASSEMBLER?

****

PROGRAM LISTING #1 (TI BASIC):

```
    100 Y1=12
    110 X1=16
    120 CALL JOYST(1,X,Y)
    130 CALL KEY(1,K,S)
    140 CALL HCHAR(Y1,X1,42)
    150 X1=X1+(X/4)
    160 IF (X1>32)+(X1<1)THEN 26
    0
    170 Y1=Y1-(Y/4)
    180 IF (Y1>24)+(Y1<1)THEN 31
    0
    190 CALL HCHAR(Y1,X1,48)
    200 IF K<>18 THEN 120
    210 C=C+1
    220 IF C<17 THEN 240
    230 C=1
    240 CALL COLOR(2,C,1)
    250 GOTO 120
    260 IF X1>32 THEN 290
    270 X1=32
    280 GOTO 120
    290 X1=1
    300 GOTO 120
    310 IF Y1>24 THEN 340
    320 Y1=24
    330 GOTO 120
    340 Y1=1
    350 GOTO 120
```

PROGRAM LISTING #2 (EXTENDED BASIC):

```
    100 C=2 :: Y1=12 :: X1=16
    110 CALL JOYST(1,X,Y):: CALL
     KEY(1,K,S):: CALL HCHAR(Y1,
    X1,42):: X1=X1+(X/4):: IF (X
    1>32)+(X1<1)THEN 150
    120 Y1=Y1-(Y/4):: IF (Y1>24)
    +(Y1<1)THEN 170
    130 CALL HCHAR(Y1,X1,48):: I
    F K=18 THEN C=C+1 :: IF C=17
     THEN C=1
    140 CALL COLOR(2,C,1):: GOTO
     110
    150 IF X1>32 THEN X1=1 ELSE
    X1=32
    160 GOTO 120
    170 IF Y1>24 THEN Y1=1 ELSE
    Y1=24
    180 GOTO 110
```

PROGRAM LISTING #3 (E/A SOURCE CODE):

```
        DEF  JOY
        REF  VSBW,KSCAN
KEYBRD  EQU  >8374
JOYSTK  EQU  >8376
YOFFST  DATA 32
KEYCK1  DATA >0100
KEYCK2  DATA >1200
DIRCHK  DATA >0000
JOY     LI   R8,12
        LI   R9,16
KEYSRC  MOV  R9,R0
        MOV  R8,R6
        MPY  @YOFFST,R6
        A    R7,R0
KEY     LI   R1,>3000
        BLWP @VSBW
        LI   R3,>0100
        MOV  R3,@KEYBRD
        BLWP @KSCAN
        MOV  @KEYBRD,R3
        SWPB R3
        CB   R3,@KEYCK1
        JEQ  EXIT
        CB   R3,@KEYCK2
        JNE  MOVCUR
        AI   R5,16
        CI   R5,240
        JLT  KEY2
        CLR  R5
KEY2    MOV  R0,R4
        LI   R0,>0385
        MOV  R5,R1
        SWPB R1
        BLWP @VSBW
        MOV  R4,R0
MOVCUR  LI   R1,>2A00
        BLWP @VSBW
        MOV  @JOYSTK,R3
        CB   R3,@DIRCHK
        JEQ  XCHK
        JLT  ADDY
        DEC  R8
        CI   R8,-1
        JGT  XCHK
        AI   R8,24
        JMP  XCHK
ADDY    INC  R8
        CI   R8,24
        JLT  XCHK
        AI   R8,-24
XCHK    SWPB R3
        CB   R3,@DIRCHK
        JEQ  KEYSRC
        JGT  ADDX
        DEC  R9
        CI   R9,-1
        JGT  KEYSRC
        AI   R9,32
        JMP  KEYSRC
ADDX    INC  R9
        CI   R9,32
        JLT  KEYSRC
        AI   R9,-32
        JMP  KEYSRC
EXIT    BLWP @0
        END
```

Long Integer Utilities for c99

Tom Wible

When I was writing high gravity, I found I needed more precision in my calculations. Although the screen is only 256 pixels wide by 192 high, to calculate the distance between opposite corners requires 2 words of precision. The GPL floating point routines are too slow, so I wrote the following long integer utilities.

The first two, getrem and hiword, merely access registers that the c99 compiler uses. Getrem is the equivalent of the remainder operation, but without the need to redivide. As assembly programmers know, the 9900 divide puts the quotient in the register specified, and the remainder in the next register.

The function hiword merely returns the high-order word of the result of a multiply. C99 ignores this value.

The mpydiv function completely bypasses c99 arithmetic so that the 2-word product of a multiply can be divided by the 3rd argument in the call. Again, I wrote this to improve the precision of my gravitational calculations.

And of course, any calculation of distance between points requires a square root. I found this algorithm in Dr. Dobb's(sorry, can't remember the issue, but it was in an article on graphics primitives for the comode-door 64;-). The algorithm involves testing each bit in the answer against each bit in the square, along with carries between words. The original was written for 2 words on the c64: 16 bits!

[Ed. note: It was issue #103, May 1985 and the article was "Solid Shape Drawing on the Commodore 64" by Richard Rylander. I highly recommend Dr. Dobb's Journal, fondly known as DDJ, to any programmer looking for a computer magazine of real substance!]

The hypotn function merely computes the sum of the squares of the numbers before falling into the square root routine.

```
*
* long integer utilities for c99
*      tom wible
*
    DEF  GETREM,HIWORD,HYPOTN,MPYDIV
*
* get remainder from last c99 division
*
*  i = 5 / 3;     /* i == 1 */
*  j = getrem(); /* j == 2 == 5 % 3 */
*
GETREM MOV  1,8
       B    *13
*
* get high order word from last c99
* multiply
*
*     i - 32000 * 5;  /* i == -   */
*     j = hiword();   /* j ==     */
*
HIWORD MOV  7,8
       B    *13
*
*
* multiply 1st arg X 2nd arg, divide
* by 3rd carrying 2 word intermediate
* value
*
*  i = mpydiv(a,b,c);
*    /* i == (a*b)/c */
*
MPYDIV MOV  @>0006(14),0
       MPY  @>0004(14),0
       DIV  @>0002(14),0
* check for overflow
       JNO  NOVERF
* set both quotient and remainder
* to max positive
       LI   0,>7FFF
       MOV  0,1           * to max positive
NOVERF MOV  0,8
       B    *13
*
* compute integer hypotenuse
*  from 2 1-word arguments,
*    using 2-word sqrt
*
*    h = hypotn(opposite, adjacent);
*   /* == sqrt(opp*opp + adj*adj) */
*
HYPOTN MOV  @>0002(14),1
       MOV  @>0004(14),3
       ABS  1        * assume signed
       ABS  3
       MPY  1,1      * product in r1,2
       MPY  3,3      * product in r3,4
       A    4,2      * loword
       JNC  HYPTN    * no carry?
       INC  1        * yes
HYPTN  A    3,1      * hiword
       JMP  SQRT
*
*  compute integer square root from  2
*  word argument using algorithm  from
*  dr. dobb's journal
*   translated from 6508 assembly
*
*     i = lsqrt(high_word, low_word);
*
*
LSQRT  MOV  @>0004(14),1   * hiword
       MOV  @>0002(14),2   * loword
RDCNHI EQU  1
RDCNLO EQU  2
TESTHI EQU  3
TESTLO EQU  4
TEMPHI EQU  5
TEMPLO EQU  6
ROOTHI EQU  7
ROOTLO EQU  8
* counter, start at 16 shifts
SQRT   LI   0,16
       CLR  TESTHI
       CLR  TESTLO
       CLR  TEMPHI
       CLR  TEMPLO
       CLR  ROOTHI
       CLR  ROOTLO     * the answer

LOOP   SLA  ROOTHI,1
       SLA  ROOTLO,1   * guess LSB = 1
       JNC  SQRT02
       INC  ROOTHI
SQRT02 INC  ROOTLO    * try next bit on
       JNE  SQRT03
       INC  ROOTHI
* shift radicand into temp
SQRT03 SLA  TEMPHI,1
       SLA  TEMPLO,1
       JNC  SQRT04
       INC  TEMPHI
SQRT04 SLA  RDCNHI,1
       JNC  SQRT05
       INC   TEMPLO
SQRT05 SLA   RDCNLO,1
       JNC   SQRT06
       INC   RDCNHI
* shift radicand into temp 2nd bit
SQRT06 SLA   TEMPHI,1
       SLA   TEMPLO,1
       JNC   SQRT07
       INC   TEMPHI
SQRT07 SLA   RDCNHI,1
       JNC   SQRT08
       INC   TEMPLO
SQRT08 SLA   RDCNLO,1
       JNC   SQRT09
       INC   RDCNHI
SQRT09 MOV   TEMPHI,TESTHI
       MOV   TEMPLO,TESTLO
       S     ROOTLO,TESTLO
       JOC   SQRT10
       DEC   TESTHI
SQRT10 S     ROOTHI,TESTHI
       JLT   SQRT11
       MOV   TESTHI,TEMPHI
       MOV   TESTLO,TEMPLO
       INC   ROOTLO
       JNE   SQRT12
       INC   ROOTHI
SQRT12 DEC   0
       JNE   LOOP
       JMP   SQRT14
SQRT11 DEC   ROOTLO
       JOC   SQRT15
       DEC   ROOTHI
SQRT15 DEC   0
       JNE   LOOP
SQRT14 SRL   ROOTLO,1
       SRL   ROOTHI,1
       JNC   SQRT16
       AI    ROOTLO,>8000
SQRT16 B     *13
```

Getting The Most From Your Cassette System

by Mickey Schmitt

[This is the second in a series of articles covering use of the TI cassette system. Part 3 will appear in the next issue of The MANNERS Journal of TI Computing. - The Ed.]

IV. Cassette - Tips - Tricks - and Tidbits - Part I

This topic may sound a little strange to you, but I hope that it proves to be well worth reading, as I pass along what I've learned about computers the hard way, and what I've learned from my fellow TI friends.

Looking back on my very "first computer days", it's hard to believe that I was once such a "rookie". I knew absolutely nothing about computers back then (as you will soon find out!)

I will always remember the very first thing that I ever learned about the computer, and to this day I am still impressed with the fact! "The computer uses the same type of cassette recorder and cassette tape to store a "program" on, as you would use to record your favorite music on." With this thought in mind, I soon learned that it wasn't necessary to purchase "special data cassette tapes" for the computer. The "standard" c-60 cassettes will work just fine with your computer and they are so much more "economical", than those "special computer cassettes!"

Over the years I have decided on using maxell c-60 cassette tapes for my own personal computer use, but I will be the first to admit that there are alot of other brands of cassette tapes that would work with your computer just as well. I would however, caution you against using any type of radio shack cassette tape (computer or standard) and any type of Certron tape, as these particular brands of cassette tapes have been known to give people trouble in the past.

Believe me, there is nothing more frustrating than finding out that a program which you just saved onto one of these types of cassettes, will not load back properly from the same cassette at a later time! The reason for this particular problem occuring is that the program is being played back at a slightly different tape speed than which it was recorded at, thus creating a slight distortion in the sound of the tape. As a word of warning, using either of these two brands of cassettes may be hazardous to your present state of mind!

In keeping with my promise that I made last month that you would enjoy a good laugh at my own expense, just wait till you hear what I used to do! Would you believe that I used to load programs into my computer, run them, and then save them back onto their original cassettes in the very same location as they were recorded on the tape in the first place. (Without ever even editing any of the programs!) Don't ask me where I ever got the idea that once you loaded a program off of a cassette tape that it was physically removed from the cassette tape but that is the impression that I was under at that time.

Of course, I plead that at that time in my "computer learning" I was not a member of any computer club nor did I know anyone who even owned a computer so I was left to struggle on my own and make alot of mistakes along the way in the process. I did learn this the hard way but I bet that I'll never forget it either: "If you are only running a program and you are not making any changes in the program whatsoever it is not necessary to save the program back onto its original cassette in its original tape location because it never really left the tape in the first place!" It is always there (unless you record over it!) you may laugh if you wish but it's all a part of learning and we all had to start somewhere!

IV. Cassette - Tips - Tricks - and Tidbits - Part II

In this part, I am continuing with the topic of cassette tips, tricks and tidbits, as I try to pass along more of what I've learned the hard way, and what I've learned from my fellow TI friends.

In keeping with the spirit of learning from one's own mistakes, I would highly recommend using the following guidelines when you are working with your cassette system.

When saving your programs onto a cassette, you should get into the habit of recording them onto a counter reading which ends in a zero. This may sound like an unnecessary procedure to follow at first but let me assure you that it is a very good habit to get into as it actually serves two useful purposes. First, it will make it much easier to locate a program on your cassette tape as you are watching the counter reading speed by, and second, (and far more important) it will allow you some additional blank space between your programs. That way you can make changes on a program and then save it back onto your cassette at the same tape location as the original was located without accidently writing over the first part of the following program or the ending of the preceeding program! Believe me, I learned this the hard way. If you don't give yourself a little extra room to work with, you run a very high risk of overwriting your programs when you try to save them back over your originals!

If you have a cassette tape that you wish to keep permanently and are afraid that you may accidently record over it you can break out the left rear tab of the side of the cassette that you want to save, or you can break out both tabs if you wish to save both sides of the cassette. Following this procedure will prevent you from accidently recording over your programs. If, however, you decide at a later time that you would like to record over a cassette that has had its tabs broken out, all is not lost. A piece of cellophane tape placed over the tab opening will allow you to once again record programs onto the cassette.

One of the most important things that I have learned about cassette tapes is that if you don't keep up with them you start to collect alot of junk. This junk that I am referring to is the many bits and pieces and parts of programs that were saved when you were creating or using a program. Once your final program is completed get rid of all your junk saves! If you don't do it right away you'll forget about it and the next thing you know, you start saving new programs onto a cassette that is full of junk and then you end up having to waste alot of valuable time, checking an entire cassette just to find out what's what! Once a junk tape has served its purpose record over it with a volume setting of zero. That way the junk will be erased and you won't have to wonder if that particular program or tape was important anymore!

V. Understanding Cassette Error Codes and Messages - Part I

Understanding cassette error codes and messages is not as difficult as it may seem. Unfortunately, trying to find a list of the error codes and messages that deal specifically with the cassette recorder has been a difficult task! In doing my research for this particular article, I have had to combine many different sources of information in order to be as informative and complete as possible.

Basically, cassette error codes and messages can occur during one of two different types of commands. More specifically, I am referring to the loading (old cs1) procedure and the saving (save cs1) procedure. In this part, I will be examining the error codes and messages that can occur during the loading procedure.

When the computer finishes loading the data, it tells you whether or not it read the data properly. If the data were read correctly, you would see the following message appear on your monitor or tv screen:

```
* data ok *
press cassette stop     cs1
then press enter
```

If, however, the computer did not successfully read your program into memory, an error occurs and the computer prints one of the following error messages:

```
* ERROR - NO DATA FOUND
  PRESS R TO READ          CS1
  PRESS C TO CHECK
  PRESS E TO EXIT
```
or
```
* ERROR DETECTED IN DATA
  PRESS R TO READ          CS1
  PRESS C TO CHECK
  PRESS E TO EXIT
```

When this occurs, you have a choice of using one of the following three options. Note, however, that the single-letter responses (R, C, or E) that you type in at this time must be in uppercase characters!

1. Press R to repeat the reading procedure. However, before repeating this procedure, check to make sure that you have put the cassette tape in correctly, that it is the correct cassette tape and that it has been placed in the cassette recorder with the correct side facing up. Then follow the directions as they appear on your monitor or tv screen.
2. Press C to check the data you have read into memory. At this point you may wish to adjust your cassette recorder's volume control and tone setting. Then follow the directions as they appear on your monitor or tv screen.
3. Press E to exit from the loading procedure. At this time another error message is displayed, indicating that the computer did not properly read your program into memory:

```
* WARNING: CHECK PROGRAM IN MEMORY *
I/O ERROR 56
```

If I/O error 56 appears, something definitely went wrong. But don't panic! Generally speaking, when the error message "error - no data found" occurs, the computer did not recognize the cassette recorder at all during the "old cs1" routine. On the other hand, when the error message "error detected in data" occurs, the computer recognized only part of the data that the cassette recorder was sending to the computer.

When this happens, recheck your cassette recorder's volume control and tone setting. Then recheck your cassette cable. make sure that both ends of the cable are attached to the computer and to the cassette recorder. While you are at it, make sure that the color-coded wires leading to the cassette recorder are connected correctly. The cassette recorder will not operate properly if the color-coded wires are reversed!

V. Understanding Cassette Error Codes and Messages - Part II

In this part, I will be examining the error codes and messages that can occur during the "saving" (save CS1) procedure.

When the cassette recorder finishes saving your program, the computer will tell you whether or not the program was recorded successfully. If the program was recorded successfully, you would see the following message appear on your monitor or tv screen, after you completed the necessary steps in the checking procedure:

```
* DATA OK
* PRESS CASSETTE STOP     CS1
  THEN PRESS ENTER
```

If, however, the cassette recorder did not successfully record your program onto the cassette tape, an error occurs and the computer prints one of the following messages:

```
* ERROR - NO DATA FOUND
  PRESS R TO RECORD       CS1
  PRESS C TO CHECK
  PRESS E TO EXIT

* ERROR IN DATA DETECTED
  PRESS R TO RECORD       CS1
  PRESS C TO CHECK
  PRESS E TO EXIT
```

When this occurs, you have a choice of using one of the following three options. Note, however, that the single-letter responses (R, C, or E) that you type in at this time must be in uppercase characters!

1. Press R to repeat the recording procedure. However, before repeating this procedure, check to make sure that you have put the cassette tape in correctly, and that there is enough blank tape left on the cassette tape in which to record the program on. Then follow the directions as they appear on your monitor or tv screen.

2. Press C to check the data you have read into memory. At this point you may wish to adjust your cassette recorder's volume control and tone setting. Then follow the directions as they appear on your monitor or tv screen.

3. Press E to exit from the saving procedure. At this time another error message is displayed, indicating that the cassette recorder did not properly save your program onto the cassette tape..

```
* WARNING:
  CHECK PROGRAM IN MEMORY
* I/O ERROR 66
```

If I/O error 66 appears, something definitely went wrong. But don't panic! generally speaking, when the error message "error - no data found" occurs, the computer did not recognize the cassette recorder at all during the "save cs1" routine. On the other hand, when the error message "error in data detected" occurs, the computer recognized only part of the data that the cassette recorder was sending to the computer. When this happens, recheck your cassette recorder's volume control and tone setting. Then recheck your cassette cable. Make sure that both ends of the cable are attached to the computer and to the cassette recorder. While you are at it, make sure that the color-coded wires leading to the cassette recorder are connected correctly. The cassette recorder will not operate properly if the color-coded wires are reversed!

V. Understanding Cassette Error Codes and Messages - Part III

```
*************************************
* General areas to check when       *
* cassette error codes and messages *
* occur                             *
*************************************
```

1. Make sure that your cassette recorder is connected to your computer console correctly. The cassette recorder interface cable must be connected to the 9-pin plug at the rear of the computer console. Don't confuse this plug with the 9-pin joystick port on the side of the console; they are not interchangable! While you are at it, make sure that the color-coded wires which plug into the cassette recorder are attached correctly as well. The cassette recorder will not operate properly if the color-coded wires are reversed! They must be "black" to the recorder's remote jack, "white" to the recorder's earhone jack, and "red" to the recorder's microphone jack.
2. If you are using d/c current instead of a/c current, make sure that your batteries are fresh! Weak batteries will cause your data to be distorted!
3. Make sure that your cassette recorder's volume control and tone settings are adjusted properly. Generally speaking, a volume control of "8" and a tone setting of "9" are recommended.
4. Make sure that your cassette tape head is clean. If you can't remember the last time that you cleaned it - then it's been too long!
5. Make sure that you are using a high-quality cassette tape. A tape of poor-quality yields "poor-performance" headaches and total frustration!
6. Make sure that your cassette tape is not any longer than a C-60 cassette. (which is 30 minutes per side). Longer tapes are thinner and provide less fidelity.
7. Make sure that your cassette tape is in good condition - that it has not been damaged or accidently erased. If in doubt, try another tape!
8. Make sure that you have put the cassette tape in correctly - that it is the correct cassette tape and that it has been placed in the cassette recorder with the correct side facing up. Also, make sure that the cassette tape has been positioned at the beginning of the desired program.
9. Make sure that your cassette tape was recorded with your cassette recorder or an identical model. If the cassette tape was originally recorded using a different type of cassette recorder, it is possible that the program will not load properly. When this occurs, you have no choice but to either obtain another copy of the program, using your cassette recorder to "save" the program or "load" the program again this time using the cassette recorder that had originally "saved" the program.

****

[Editor's Note: Anyone wishing to help rebuild the MANNERs cassette library should contact Jerry Coffey who can put you in touch with others interested in working on this project. The MANNERS cassette library ceased when many users migrated to disk systems and our cassette librarian left the club. With the influx of new members, some using cassette systems, restoring the cassette library has become important.]

NetNotes - Around the The World of TI Computing
Dave Ramsey

There are a few updates to last month's NetNotes column. First and perhaps foremost, Bob Fowler and Bill Cavanaugh have done it again! The BBBB now boasts an online library of almost 200 TI-99/4A programs and data files and almost 100 Myarc 9640 programs and data files. Bob is now trying to get the users to help pitch in to add a 40 megabyte hard drive to the BBS and when that happens, the BBBB will have the capacity for literally thousands of TI and 9640 programs! Good luck Bob and all BBBB users are encouraged to send Bob $10 to help defray the cost of the 40 meg hard drive. (Bob doesn't use these hard drives himself. They are dedicated to the BBS and hold the files for running the BBS and the programs that the users can download.)

On another front, PC Pursuit has made drastic changes to their rate structure. PC Pursuit was once the finest telecommunications service that a user could subscribe to. It didn't offer online databases or forums or anything like Delphi or CIS. But it did give unlimited dialing privileges into the 30 PC Pursuit cities around the country for a flat rate of only $25 per month. By using this service you could avoid all long distance charges for calling any of these cities from your hometown - if you had a Telenet "indial" node that was local to you. And since Telenet has over 10,000 indial nodes around the US, it was very easy for most telecommuters to access the various BBSs around the country through PC Pursuit.

Alas, those days are now over. PC Pursuit has raised their price to $30 per month. But that is not the bad part. The difficult part is that there is now a 30 hour cap per month on use of the network. Now 30 hours may seem like alot to many computerists who don't use telecommunications. And while I can go for several months with usage under 30 hours, there were also many months when I used PC Pursuit for perhaps as many as 60 hours. But now is you violate the 30 hour cap, you will be slapped with a $4.50 per hour charge for network usage.

Now the reason that I say that this isn't competitive anymore is because Delphi, a leading telecommunications service firm, has recently begun to offer access to Delphi for $4.80 per hour under their "Advantage Plan". Delphi offers forums, online shopping, online games, databases, program libraries, and much, much more. PC Pursuit is only offering access to the Telenet packet switching network with no further bells or whistles.

It is because of this excellent price offered by Delphi, and the perhaps fatal rate hike made by Telenet for PC Pursuit, that I can no longer recommend PC Pursuit to TI computerists entering the world of telecommunications. Instead, I now can recommend Delphi as the finest and most economical telecommunications bargain in the United States.

And since I have changed my recommendation, it is only fitting that I spend a little time bringing MANNERS members up to date about Delphi. Delphi has many services available for non-computer use. But for the TI-99/4A hobbyist, the biggest allure of Delphi must certainly be the revamped TI Net area on Delphi.

Our own Jeff Guide is the system manager of the TI-Net area on Delphi. Likewise, I am currently the MANNERS representative online there. Our own Jerry Coffey is also present there to help new users and coordinate activities.

In addition to many of our own users online at Delphi, many of the notable authors of the TI world are regular participants there. Al Beard, author of Fortran-99 for the TI is a regular vistor as is Mike Dodd, who authored the latest versions of Myarc's Disk Manager software for use with the Myarc floppy disk controller card. Another TI notable who is frequently present is Barry Traver, the TI Computer Shopper columnist and BASIC programming authority. Paul Charlton, author of the 9640 MDOS operating system and J. Peter Hoddie, author of MyWord and co-owner of Genial Computerware also regularly frequent TI-Net. There are many others as well but I will sum this up by saying that if you cannot get an answer to your technical or programming questions from the crowd on TI-Net, then you probably can't get an answer!

But what else does Delphi offer? In addition to the TI-Net Forum, where everyone can and does leave messages and questions about programming, using software, and comments about software, thre is also the TI-Net database area. In this area, the Delphi user has access to thousands of TI-99/4A programs. These programs are separated out into categories for easier browsing. After the user selects a category, he can perform a DIR command, which gives a summary of the programs available in that area. The summary allots one line to each program and shows when it was uploaded, who uploaded it to Delphi, and a short description.

After you check out the category with the DIR command, you can issue the READ command. For example, say that the latest version of some adventure game was listed as entry number 36. You could tell Delphi to "READ 36" and the Delphi computer would display on your screen the full description of that item. After showing the description for it, you would be given the option to download this program to your TI-99/4A Home Computer, to look at the next entry in that category, or to quit. Needless to say, this gives the TI user access to a veritable treasure trove of software and information about his TI computer.

There is much more to say about Delphi but it will have to wait until next month! I'm out of space for now. See you on the bit stream!

c99 and Internal files
by Dan Gazsy

When Clint Pulley first released the c99 compiler, an individual by the name of Tom Bentley had the foresight to provide library functions for the floating point accumulator (FAC) and a general purpose I/O library called TCIO. Initially I didn't find much of a need for the FLOAT library other then to implement SIEVE type programs.

Two months ago, I decided to write some c programs to perform file maintenance on a bbs message base. My first course of action was to check around to see what c code had been written along these lines. I found the FLOAT and TCIO libraries were all I needed to implement my application. As an added bonus, I ran across a disk catalog function written by Tom Bentley which used both libraries.

Before you write any applications to process internal type files, you should have an understanding of how internal records are structured. When the file is created, you allocate the number of bytes for each record. The fields associated with each record can be either numeric fields or string fields. Numeric fields use 9 bytes; the first byte contains the length of the field (always 8 bytes) used to represent the value in float format. String fields are stored in the same manner as dv80 or df80 strings. The first byte contains the string length; immediately followed by the string. The fields are stored in the record in the sequence they appeared in the print statement (or however the file was created). All unused characters are filled with nulls (0). For example if I created a file of 70 characters and wrote 3 fields (name, address and age) to it, they would look like the following:

```
byte 01    - length of name string
byte 02-10 - Name string
```

byte 11	- length of address string
byte 12-24	- Address string
byte 25	- length of age field (always 8)
byte 26-33	- Age field
byte 34-70	- Null characters

Great you might say! Now how do I read these records from a file into my c program and make decisions based on the values of the fields? Well to start, you open the file with "topen" and read the records with a "tread". In the event you are not sure of the size of the input record, set up an array of 256 bytes to be on the safe side. Once you complete the "tread", the entire record is in the array you specified as the buffer argument. Now all that's left is to parse the record into fields. To do this, you must know how many fields are in each record, what type field they are (string/numeric) and the order they appear in the record. Below appears six functions which we will use to parse the records. The first two functions (getstr and getnum) will be used to parse the input record and the next two functions (putnum and putstr) are used to parse the output record. The last two functions (strlen and getfn) are used to size strings and to open TESTFILE for input or update mode. We lumped all of these into this section because it's what I'd consider a utility. In the interests of good programming practice, we'll compile the following 79 lines of code as you would create a support library. In addition to compiling this program, you'll also have to assemble it. For test purposes, let's call these 79 lines UTILITY;C and the output of the c compiler will be called UTILITY;S. As expected the input file for the Assembler will be called UTILITY;S and the output will be called just plain old UTILITY. Congratulations, you've just created a support library! [ed. note: see listing #1]

To demonstrate that this isn't magic and is compatiable with files created by other programming languages, I've created the input file from a Basic program! [see listing #2.]

What follows now is simple c program to read the file we created in the above Basic program and display the fields. We will be referencing the library functions we created earlier and code will be included in our main program which will demonstrate how to reference these functions. [see listing #3.]

The goal of the function called "main()" has a rather simple purpose. It is to open the file called "TESTFILE", read record 0 into a buffer, parse the buffer into known fields and finally display them. As you can see, I named the program TESTREAD;C, the output code of the compiler was called TESTREAD;S and the assembler output code was named TESTREAD. To execute the program you will have to specify the following files TESTREAD, UTILITY, TCIO, FLOAT and CSUP.

The last part of this column will display a method to combine fields entered on the screen into a record; then write the record back to the file TESTFILE. Like the earlier test, I called this one TESTWRIT;C, the output of the c99 compiler was called TESTWRIT;S and the assembler output was called TESTWRIT. To execute this program you'll need to load the following files: TESTWRIT, UTILITY, TCIO, FLOAT and CSUP. Again remember these are load and run (option 3) files. [see listing #4.]

To insure that the files written in c99 are compatiable with other languages, we'll read the file and display the fields with a basic program. [see listing #5.]

Listing #1 - "C99 and Internal FIles"

```
entry getstr,getnum,putnum,putstr,strlen,getfn;
#include "dsk1.floati"
#include "dsk1.tcioi"

getstr(buff,t)
   int *buff;
   char *t;
   { char *b;
     int j,siz;
     b=*buff;           /* set up b with byte address of input rec */
j=siz=*b++;        /* set up j & siz with # bytes in string */
*buff=*buff + j + 1; /* push to next input field */
    while(j--)
      *t++=*b++;    /* transfer string from input rec to string */
    *t = '\000';    /* terminate with null byte */
return(siz);        /* return size of string */
  }

getnum(buff,t,f)
  int *buff;
  char *t;
  float *f;
  { char *b;
b=*buff;         /* set up b with byte address of input rec */
 ++b;            /* point data past the float size indicator */
*buff = *buff + 9;
fcpy(b,f);      /* copy the input field into the float */
ftos(b,t,0,0,0); /* convert field to string variable */
  }

putstr(buff,t)
  int *buff;
  char *t;
  { char *b;
    int j,siz;
    b=*buff;    /* set up b with byte address of output rec */
    j=siz=strlen(t);  /* set up j & siz with # bytes in string */
    *buff=*buff + j + 1; /* push to next input field */
    *b++=j;      /* put string length in 1st byte of field */
    while(j--)
      *b++=*t++; /* transfer string to output rec */
return(siz);     /* return size of string */
  }

putnum(buff,t,f)
  int *buff;
  char *t;
  float *f;
{ char *b;
  int i;
  b=*buff;
/* set up b with byte address
```

```
   of output rec */
  *b++=8;
/* put float id in output rec */
  *buff = *buff + 9;
  while(*t==' ')
/* get rid of leading spaces */
    t++;
  stof(t,b);
/* copy the string field
   into output record */
}

strlen(s)
  char *s;
{ int n;
    n=0;
  while(*s++)
      ++n;
  return(n);
  }

getfn(text,m)
  char *text;
  int m;
{ int unit;
  unit=0;
  if(m==0)
    unit=topen(&text[0],INPUT+RELATIVE+INTERNAL+FIXED,0);
  else
    unit=topen(&text[0],UPDATE+RELATIVE+INTERNAL+FIXED,0);
  return(unit);
}
```

Listing #2 from "C99 and Internal Files"

```
100 OPEN #1:"DSK1.TESTFILE",
 RELATIVE,INTERNAL,OUTPUT,
 FIXED 70
110 A$="Dan Gazsy"
120 B$="22 6th Street"
130 C=37
140 PRINT #1,REC 0:A$,B$,C
150 CLOSE #1
```

Listing #3 - "C99 and Internal Files"

TESTREAD;C listed below:

```
extern getfn(),getstr(),getnum();
#include "dsk1.tcioi"
#include "dsk1.floati"
#include "dsk1.stdio"

main()
```

```
{
  int fp,b_ptr,size,i;
  char buff[256],sdum[81];
  float fdum[FLOATLEN];

  putchar(FF);
  for(i=0;i<256;i++)
  buff[i]=0;
  fp=getfn("DSK1.TESTFILE",0);
  if(fp>0) {
    tread(buff,0,fp,&size);
    b_ptr=buff;
    getstr(&b_ptr,sdum);
    puts("Name:");
    puts(sdum);
    getstr(&b_ptr,sdum);
    puts("\nAddress:");
    puts(sdum);
    getnum(&b_ptr,sdum,fdum);
    puts("\nAge:");
    puts(sdum);
  }
  else
    puts("Error, file could not be opened!");

  tclose(fp);
  puts("\n\nStrike any key to continue");
  getchar();

}
```

Listing #4 - "C99 and Internal Files"

TESTWRIT;C listed below:

```
extern getfn(),putstr();
extern putnum(),strlen();
#include "dsk1.tcioi"
#include "dsk1.floati"
#include "dsk1.stdio"

main()
{
 int fp,b_ptr,size,i;
 char buff[256],sdum[81],ndum[81];
 float fdum[FLOATLEN];
 putchar(FF);
 b_ptr=buff;
 for(i=0;i<256;i++)
 buff[i]=0;
 while(1) {
  puts("Enter name (max 20 chars):");
  gets(ndum);
  size=strlen(ndum);
  if(size<21) break;
 }
 putstr(&b_ptr,ndum);
 while(1) {
  puts("Enter addr. (max 20 chars):");
  gets(ndum);
  size=strlen(ndum);
  if(size<21) break;
 }
 putstr(&b_ptr,ndum);
 while(1) {
  puts("Enter age (max 3 digits):");
  gets(ndum);
  size=strlen(ndum);
  if(size<4) break;
 }
 putnum(&b_ptr,ndum,fdum);
 fp=getfn("DSK1.TESTFILE",1);
 if(fp>0) {
  puts("Writing record");
  twrite(buff,0,fp,70);
  tclose(fp);
 }
}
```

Listing #5 from "C99 and Internal Files"

```
100 OPEN #1:"DSK1.TESTFILE",
 RELATIVE,INTERNAL,INPUT,
 FIXED 70
110 INPUT #1,REC 0:A$,B$,C
120 PRINT "NAME:"&A$
130 PRINT "ADDRESS:"&B$
140 PRINT "AGE:"&STR$(C)
150 CLOSE #1
```

TI BBS Support
Dave Ramsey

Bob Fowler recently gave me a list of changes that have occurred in the BBBB bulletin board since I last wrote about. The first change is that while access to the board is still free, Bob has decided to grant file access to only those folks who help him keep the board running. To gain access to the file system, Bob suggests a donation of $10 per year. Given the volume of software and the availability of TI expertise on this board, I can say that $10 is a bargain for anyone who uses it.

Another change has been the addition of another message base. There are now separate message bases for the TI-99/4A and the 9640. Each message base holds up to 200 messages. Further, the BBBB has been upgraded to now support 2400 baud access.

Planned enhancements for the future include wordwrap in the message editor and the addition of a 40 meg hard disk for the BBBB to increase the number of files online for the user community. As of the 5th of February (when I am writing this), Bob has 260 TI-99/4A downloads and 134 9640 downloads online. In addition, there are 50 text files with articles from other user groups and general information for TI users on a wide range of subjects.

[Ed. Note: Since this issue went out so late, in early April, the number of files online has grown greatly. Check in on Bob's board to see the current status as it changes daily with new uploads by prominent TI shareware and public domain authors.]

Finally, of the 160 users currently using the system, about 75% are long distance callers and 30 are known to be 9640 users.

So, if you haven't called Bob and Bill's Bulletin Board with your TI and your modem, give them a ring at (301) 292-1492. They are a local call to the DC metropolitan area.

99 Fortran Benchmark Tests
Peter McPhie, MCTIUG

In March of 1987, Stephen Shaw published a series .of mathematical programs to evaluate all of the languages of the 99/4A. Since then, a new language has become available - 99 Fortran from LGMA Products. As a scientist, this is the first language I think of when number-crunching somes up, so rewriting some of these programs seemed like a good way to evaluate this new package.

I bought 99 Fortran, version 2.1, for $50 at Christmas 1987 and since then have upgraded to version 3.1.3 for another $15. For your money, you get a 160 page manual which describes the language well, but will not teach you Fortran, plus three SSSD disks. One disk contains the boot program, menu, editor, debug, compiler, linker, and a stand-alone run-time module so you can distribute your programs. The others contain a collection of linkable, precompiled subprograms, routines, and functions. Among these are 73 mathematical functions and 52 Extended BASIC and assembly language routines.

With version 3, you can now put many of the library functions on the disk with the editor to save alot of disk juggling. Unfortunately, not all will fit since a TI disk directory is limited to 127 entries.

I sent the original versions of these programs to Al Beard, the author of 99 Fortran. Al suggested some improvements, plus the self-timing subroutine which I included. I ran all of these benchmarks on a black 99/4A with 32K, TI disk controller, and 2 Tandon drives. Mr. Beard also ran the programs on a Geneve in TI mode and got slightly faster times for all of them. The fastest and slowest times for other languages are also reported.

The first benchmark evaluates a simple mathematical function with integer variables, with 1000 repeats. It ran with no problems. Memory address >202A contains the system clock and can be set to 0 then peeked to time programs when the disk drives are not running.

```
        PROGRAM INTMATH
        INTEGER I,X,Y,PEEK,IENDTIME
        CALL LOADM(Z'202A', 0)
        X = 0
        Y = 9
        WRITE(6,10)
10      FORMAT('1START')
        DO 20 I = 1,1000
20        X = X + Y - Y*Y/Y
        WRITE(6,30)
30      FORMAT(' FINISH')
        WRITE(6,40) X
40      FORMAT(I8)
        IENDTIME = PEEK(Z'202A')
        IENDTIME = IENDTIME/10
        WRITE(6,50) IENDTIME
50      FORMAT(' TIME = ',I8)
        STOP
        END
```

Run Times

v2.1.5	2.5 seconds
v3.1.3	0.5 seconds
C99	0.5 seconds
Myarc XB	18.0 seconds

The next program performs a similar loop but uses real arguments, which serves to slow things down.

```
        PROGRAM REALMATH
        INTEGER I,IENDTIME,PEEK
        REAL X,Y
        CALL LOADM(Z'202A', 0)
        X = 0
        Y = 9.9
        WRITE(6,10)
10      FORMAT('1START')
        DO 20 I = 1,1000
20        X = X + Y - Y*Y/Y
        WRITE(6,30)
30      FORMAT(' FINISH')
        WRITE(6,40) X
40      FORMAT(F8.1)
        IENDTIME = PEEK(Z'202A')
        IENDTIME = IENDTIME/10
        WRITE(6,50) IENDTIME
50      FORMAT(' TIME = ',I8)
        STOP
        END
```

Run Times

v2.1.5	8 seconds
v3.1.3	14 seconds
TI-XB	22 seconds
Pilot	576 seconds

Benchmark 3 loops through a complicated trigonometric formula to check out the accuracy and speed of the functions called. This program gives the wrong answer in version 2.1.5 due to an error in ATAN(X) when X>1. This was corrected in version 3. Additionally, benchmark results are given in single precision (S.P.) and double precision (D.P.)

```
        PROGRAM TRIGLOG
        INTEGER I,IENDTIME,PEEK
        REAL X,Y,COS,SIN,ATAN,ALOG
        CALL LOADM(Z'202A', 0)
        X = 0
        Y = 9.9
        WRITE(6,10)
10      FORMAT('1START')
        DO 20 I = 1,1000
20        X = COS( SIN( ATAN( ALOG(Y))))
        WRITE(6,30)
30      FORMAT(' FINISH')
        WRITE(6,40) X
40      FORMAT(F8.1)
        IENDTIME = PEEK(Z'202A')
        IENDTIME = IENDTIME/10
        WRITE(6,50) IENDTIME
50      FORMAT(' TIME = ',I8)
        STOP
        END
```

Run Times

v2.1.5	
S.P.	670 seconds
D.P.	680 seconds
v3.1.3	
S.P.	390 seconds
D.P.	400 seconds

TI-XB	362 seconds
Pilot	1710 seconds

The most surprising thing about this program is the great similarity between the less accurate, but usually fast single precision mode, where numbers are represented by 4 bytes, and the accurate but slow double precision mode, where each number is 8 bytes. If you read the manual carefully, you'll discover that FORTRAN 99 only emulates single precision for input and output, and that all calculations are carried out internally as double precision. This is where I learned that the undocumented error message "BAD REFERENCE" means that a subprogram that you used is not on the library disk.

The following program was really meant to use arguments in degrees and Brigssian (base 10) logs. Most TI languages only support radians and natural logs so the routine has to be rewritten to get the required answer, with the factors M and L. Note that FORTRAN 99 does have the functions LOG10 and DLOG10 but I wrote it this way to compare with Shaw's routine. Incidentally, both versions give the right answer because the argument of DATAN was less than 1. This example is written in double precision mode.

```
      PROGRAM DPTLOG
      INTEGER I,IENDTIME,PEEK
      REAL X,Y,DCOS,DSIN,DATAN,DLOG
      REAL M,L
      CALL LOADM(Z'202A', 0)
      X = 0D0
      Y = 9.9D0
      M = 0.01745D0
      L = 2.303D0
      WRITE(6,10)
10    FORMAT('1START')
      DO 20 I = 1,1000
20    X = DCOS(M*DSIN(M*DATAN(
     +    M*DLOG(Y/L))))
      WRITE(6,30)
30    FORMAT(' FINISH')
      WRITE(6,40) X
40    FORMAT(F8.3)
      IENDTIME = PEEK(Z'202A')
      IENDTIME = IENDTIME/10
      WRITE(6,50) IENDTIME
50    FORMAT(' TIME = ',I8)
      STOP
      END
```

Run Times

v2.1.5	
S.P.	600 seconds
D.P.	630 seconds
v3.1.3	
S.P.	390 seconds
D.P.	410 seconds
TI-XB	386 seconds
TI-BASIC	640 seconds

[Editor's Note - Peter has results and commentary on another three 99 Fortran benchmarks. Unfortuntaely, I didn't have the space to include them in this issue. Look for the second part of the "99 Fortran Benchmark Results" in the May-June issue of the newsletter.]

****

For Sale

1. TI PBox, Disk drive, controller, RS232 card, 32K card, and TI-Writer - $275.00
2. TI Writer - $14.00
3. Program recorder with cassette cable - $15.00
4. Serial Printer cable for Smith Corona printer - $5.00
5. TI Console with manuals, video modulator, and power supply in original box - $35.00
6. TI console (bare) - $20.00
7. TI Joysticks - $8.00
8. Home Computer Magazines - $1.50 ea.
9. Enthusiast '99 - $1.00 ea.
10. 99'er Home Computer Magazine - $1.00 ea.
11. MICROpendium Magazine - $.75 ea.

For more information, call Tom Romig at (703) 455-9276.

****

HOW TO BUY NEW FLOPPY DISK DRIVES

Richard Roseen

[Editor's Note: Additional questions can be directed to Richard through The MANNERs post office box number listed on the back of the newsletter.]

1. Check for quality the main mechanical parts of the drive. They should be located on a solid die cast peice of metal. In other words solid metal structure throughout as the base of the drive that holds the motors, soleniods and other movable parts. Avoid any drive put together with metal plates.

2. New drives should be sold to you in antistatic plastic wrap (usually tinted looking) and may have a fitted styrofoam container, will always be half height, never full height, at least two sided, at least capable of 360k double sided double density. The 720k 80 track drives are now getting rare due to the newer 1.2meg. drives. The 1.2 meg. drives can be used at 720k (more on that later). New 3.5" drives are 720k or 1.44 meg. They should follow the rule of die cast body as above also. Newer 3.5" drives will have a thickness much less than a half height 360k drive. Only the new Myarc HFDC has promise of possible drivers to support 1.44meg 3.5" or 1.2meg. 5.25" use. Certain CorComp controllers have floppy disk controller chips that can handle the 1.44 meg data rate, but as for the device drivers, who knows? No older Myarc disk controller will be fully capable of the 1.44 meg. data rate because of the FDC chips they use. The above also pretty much applies to the use of 1.2 meg 5.25" drives. The 5.25" 1.2meg and 3.5" 1.44 meg. drives can be used for 720k storage with the eprom driver support of the two Myarc controllers; however, if disk rotation speed cannot be jumpered through lack of information on the drive options, you would be forced to live with odd ball 720k format disks only readable by someone else with 720k capability and 3.5" 1.44 meg. or 5.25" 1.2meg. drives.

4. The newest drives always have a directly driven disk rotation motor. This means you will not see any belt driven disk rotation.

5. Warrantee's: ask what the manufacturer's warrentee is. The warrentee should be at least one year from date of purchase. Also, check to see what the seller's guarantee is on the drive. Typically the seller's guarantee is full replacement for 30 to 90 days, in addition to the one year manufacturer's warrentee. The warrantee will give you plenty of time to verify that you do not have a lemon drive.

6. Get the seller's business card with address and phone. Get a receipt in which you and the seller have a copy which must contain the serial number of drives bought and date as well as the cost. If the seller's address is on the receipt clearly that will substitute for the business card. These requirements are necessary for the manufacture's warrentee and so you can later find the seller or manufacture for information. It is not always possible that the seller has info on the drive, but it will not hurt to ask for data manuals or schematics.

7. For quality look for heads mounted on assemblies that are mounted to move solidly not jerkily such as on two rails instead of one. For low mechanical noise or low clattering (increased reliability and longer life) look for solid movement of the head assembly by a stepper motor through two following exampees: stepper motor that drives a screw shaft or two straps that wind on or off the stepper motor shaft and on or off of the head assembly as the heads move in either direction. Heads take the biggest beating in floppies and

more often involved in alignment of a drive. An example of the stepper motor that drives a screw shaft is the 3.5" 720k Chinnon and Fujitsu. An example of the strap that winds on or off the stepper motor shaft and on or off the head assembly is the Mitsumi 360k 5.25" drives.

8. 3.5" drives can be hooked up bare without the 5.25" bracket with 34 pin socket IDC (insertion displacement connector) connected to the square pins on the 3.5" drive. If this is done then the odd ball but findable 4 pin 3.5" drive power connector must be used. These are odd ball because they are not the same as the 5.25" drive power connectors. These connectors do not have a polarity tabs and can make difficult getting the proper polarity or orientation of the connector to plug in. Go for the works get the 5.25" bracket and the card edge adapter board that includes standard 5.25" power connector. These adapters may have a jumper for use on PC XT or AT clones, be sure to select XT.

9. Unless you have help from a Guru or user who has successfully installed and used the same drives, then get info from the seller or manufacturer on drive selects, other jumper options or features, and resistor packs. On some new drives the resistor pack is permenantly soldered to a high density logic board with a jumper to disable or enable the use of the resistor pack for installation as lesser drive or drives on the chain. If such a drive is the last drive in a chain whose other resistor packs can be removed, there is no problem.

10. Buy or at least shop for any drive or power connectors or power supplies or cases as you may or may not need depending on what you already have.

11. The least expensive power supplies, drive connectors, cables, etc. are sold by vendors selling chips and electronic parts, not by the dealers of floppy drives. The chip parts dealer will have alot of the necessary parts for homebuilt linear supplies at the lowest total cost of parts. A general list for a linear supply is a transformer, AC line cord and plug, switch, filter capacitor rated above 2200uF (micro farads), bridge rectifier or diodes, linear regulators both 5 and 12 volt.

12. Power requirements: some 3.5" drives require less that 1 amp for 5 and 12 volts. Some 3.5" drives are very low power and some require only a 5 volt supply. 3.5" drives require the least power. New 5.25" half height drives never require more than 1 amp on 5 and 12 volt lines and can be as low 1/2 amp. on the 5 and 12 volt line. Add the amperage required for each drive for each 5 and 12 volt line to check your power supply needs for your drives. Drives can be powered separately because the 34 pin cable will carry the common logic signal ground between all drives on the the train and the computer. If building a linear supply be sure the transformer, bridge rectifier diodes and linear regulator exceed your amperage needs. The transformer should be at least 12.6 VAC RMS and 6.3 VAC RMS (transformers are commonly rated with RMS voltage at their secondaries).

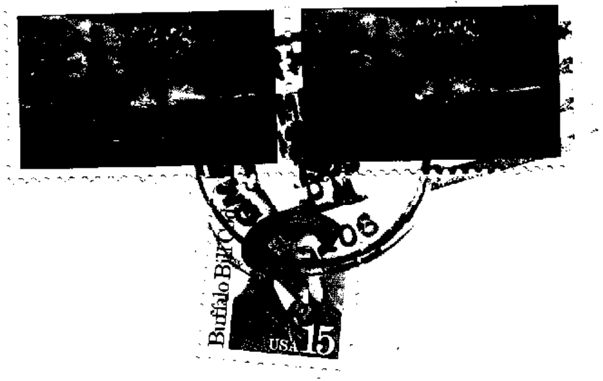
This information was kept as general as possible so as to guide the 4A buyer. My preferences are Mitsumi drives 3.5" and 5.25" any density. These drives are the most quiet drives you WILL ever hear. They have a jumper block to enable disable the resistor pack though have not verified the identity of the jumper as of yet. Another preference are the NEC 1036 3.5" 720k drives. They are small, quiet and durably solid, and like any other 3.5" drive lightweight and low power. Also, I recommend Chinnon 3.5" 720k drives. These are much the same as the NEC drives except for screw shaft stepper motor and extremely low power and 5 volt only operation make it better. These drives may be the lowest power in the industry.

```
 DOWNLOADED FROM THE SUPER OOPS BBS. SYSOP: RALPH JOHNSON
 ========================================================
* Assembly language Level 2 routine table created by
* Ralph T. Johnson (01/31/89)
*----------------------------------------------------------
*                         DSKx | WDSx (on Myarc HFDCC only) |
*------------------------------|----------------------------|
*>10 SECTOR READ/WRITE         |>20 SECTOR READ/WRITE       |
*>11 FORMAT                    |>21 --- NOT SUPPORTED ---   |
*>12 MODIFY FILE PROTECTION    |>22 MODIFY FILE PROTECTION  |
*>13 FILE/SUBDIR. RENAME       |>23 FILE/SUBDIRECTORY RENAME|
*>14 DIRECT FILE INPUT (EXT)   |>24 DIRECT FILE INPUT  (EXT)|
*>15 DIRECT FILE OUTPUT(EXT)   |>25 DIRECT FILE OUTPUT (EXT)|
*>16 BUFFER ALLOCATION         |>26 --- NOT SUPPORTED ---   |
*>17 --- NOT SUPPORTED ---     |>27 SET CURRENT PATHNAME    |
*>18 --- NOT SUPPORTED ---     |>28 CREATE SUBDIRECTORY     |
*>19 --- NOT SUPPORTED ---     |>29 DELETE SUBDIRECTORY     |
*----------------------------------------------------------
*      __________ OPERATION PERFORMED ___________
*      |10|11|12|13|14|15|16|20|22|23|24|25|27|28|29|
*-----------------------------------------------------|
*>834A |1a|1b|  |  |  |  |  |  |  |  |  |  |  |  |  |
*-----------------------------------------------------|
*>834C |2a|2b|2c|2c|2d|2d|2e|2a|2c|2c|2d|2d|2c|2c|2c|
*-----------------------------------------------------|
*>834D |3a|3b|3c|   |3d|3e|   |3a|3c|   |3d|3e|  |  |  |
*-----------------------------------------------------|
*>834E |4a|4a|4b|4c|4b|4b|   |4a|4b|4c|4b|4b|4d|4d|4d|
*-----------------------------------------------------|
*>8350 |5a|5b|   |5c|5d|5d|   |5a|   |5c|5d|5d|  |  |  |* Error codes returned
*-----------------------------------------------------|* here
*>8352 |  |  |  |  |  |  |  |6 |  |  |  |  |  |  |  |
*----------------------------------------------------------------
*>8354 | Length of device name                                   |
*----------------------------------------------------------------|
*>8356 | Points to the period after the "WDSx" or "DSKx" in PAB|
*----------------------------------------------------------------
* 1a--Sector number returned
* 1b--Total sectors formatted
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* 2a--Device number (if MSB is set, file buffer is in CPU)
* 2b--Device number
* 2c--Device number (if MSB is set, filename or pathname is in CPU)
* 2d--Device number (if MSB is set, filename & buffer are in CPU)
* 2e--Number of files to reserve buffer space (like call files)
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* 3a--Read/Write flag. If 0 then write. If not 0 then read.
* 3b--Number of tracks on a format
* 3c--File protect flag. 0=Unprotect, <>0=Protect
* 3d--0=Open file and Transfer file params, or its the # of
*     sectors to transfer.
* 3e--0=Create file and copy additional info, or # of sectors to transfer.
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* 4a--Buffer start address (can be VDP or CPU)
* 4b--Pointer to 10 character filename padded to the right with spaces.
* 4c--Pointer to NEW filename/directory
* 4d--Pointer to 40 character pathname
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* 5a--Sector number
* 5b--(MSB) 0 & 1=Single dens., 2=Double dens., 3=High dens. (LSB) # of sides.
* 5c--Pointer to OLD filename/directory
* 5d--Pointer to additional file info
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* *6* MSB sector number for sectors above 65534
*******************************************************************************0
* Additional info used on 14,15,24, & 25 (pointed to at >8350)
*
* >83xx      Buffer start address
* >83xx+2    Number of first sector
* >83xx+4    (MSB) file status, (LSB) Number of records per sector.
* >83xx+6    (MSB) EOF offset, (LSB) Logical record size
* >83xx+8    Number of level 3 records
* >83xx+10   (MSB) MSB of first sector, (LSB) MSB of level 3 records allocated.
* >83xx+12   Extended records length.
*******************************************************************************0
```

The MANNERS Journal of TI Computing

P.O. Box 4005

Rockville, MD 20850



Dallas TI Home Computer Grp
P O Box 29863
Dallas TX 75229

12/31/99 C