# The Boston Computer Society TI-99/ 4A User Group Meeting Newsletter September 1988

Edited by J. Peter Hoddie

# The October Meeting

The October meeting of the Boston Computer Society's TI-99/4A User Group will take place on Wednesday October 19 at 7:30 PM at the Massachusetts College of Art on Huntington Avenue in Boston.

The meeting topic has been slanted in favor of a "suggestion" from the BCS front office. As I mentioned last month, it appears that October is something like national computers in education month (along with national hot air campaigning month too....). Since the TI was marketed heavily as an educational machine it seems reasonable that we should be able to get one meetings worth of material out of this. So that's the plan. The Plato educational system will be demonstrated. With some bad luck, I'll end up doing it and then y'all can see what 3+ years at Boston University doesn't do for your mind. There are also lots of other TI cartridge educational programs from assorted companies, some of which will also be shown.

Keeping with the fine tradition of not devoting a single meeting to just one topic, there will be something else at the meeting. I hope to be able to preview at least one new Genial Computerware product, but I can't promise that yet.

The software library will continue to be available at meetings. Disks are \$3 each, or 4 for \$10.

This user group meets once a month. This is probably enough. I could never survive having to deal with this insanity more than once a month. It has been proposed that we have an informal gathering once a month. Mike Wright has offered to make the exchange newsletters available at this meeting. It

plans and answer questions in a slightly more relaxed atmosphere than at a meeting. I believe that I can get the BCS office meeting room for such a purpose once a month. We can hold 10 or so people there before it starts to get interesting. And there is plenty of overflow space. A copy machine is available to zap off copies of interesting newsletter articles. If you are in favor of such a meeting, scream loudly and I'll make the necessary arrangements to give it a try.

The advertisement we placed in MICROpendium to promote the software library has received an overwhelming response. The TI mailbox at the BCS office has been filled to capacity for weeks. It kinda makes you understand where Jerry Price gets all his money. Wonder if he can sue me for this???

As of late I have been using this space to print odd recommendations of businesses in the Boston area. Why? Because I need to fill the space. In July Herrel's Ice Cream got the nod. I still recommend it, but if you get down to Central Square in Cambridge give Toscannini's Ice Cream a try too. I can't decide which is better, but I'm having a lot of fun trying.

The best place for computer books in the area is probably WordsWorth in Harvard Square. Not that they have an overabundance of TI books, but they do have a great selection and good prices. For the overly technical the Tech Coop in Kendall Square has an amazing collection.

The Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

# poT-pourrl by Mike Wright

The Boston Computer Society (BCS) is considered by many to be one of the premier computer societies in the United States, if not in the world. The TI-99/4A user group of the BCS is considered to be one of the "big" TI groups, along with other groups such as Los Angeles and Chicago.

However, some people are unclear as to exactly how the TI group fits into the BCS framework. These people include BCS members and user groups who participate in the TI exchange newsletter program. This short article is an attempt to clarify the situation:

The Boston Computer Society has offices at One Center Plaza, Boston, MA 02108. It supports full-time employees who maintain the membership lists, deal with correspondence, and look after donated hardware and software, among a host of other things.

Anyone can join the BCS by sending \$36 to the above address. This entitles you, among many other things, to belong to two user groups of your choice. (For a complete list of BCS benefits you can request the booklet "The BCS Resource Guide".) User groups include IBM, Apple, Commodore, Tandy, Tlmex-Sinclair and, of course, TI-99/4A (which is separate from the TI Professional). The BCS has about 28,000 members, of whom 240 opted for TI-99/4A membership. It is estimated that there are about 12,000 members in the IBM group.

Being a member of a group entitles you to receive the publications mailed out by the group. This usually corresponds to a newsletter, but in the case of the IBM group it is closer to a fully fledged magazine.

For most groups, the newsletter is mailed to the member. However, in the TI-99/4A group the procedure is slightly different. It was decided that there would be two newsletters—one that would be mailed out and one that would be available at meetings. Since the TI-99/4A group has a meeting each month—even during summer—the meeting newsletter appears 12 times a year. The newsletter you are reading now is the monthly meeting newsletter.

The mailout newsletter appears four to six times a year and can contain material from the meeting newsletter, or from other sources. Different people within the TI-99/4A group are responsible for producing each of the two newsletters.

The original idea was that the meeting newsletter would encourage members to attend meetings. If you didn't come to a meeting you missed out on the newsletter. While the idea has some merit, there are times when you just cannot make a meeting, even with the best will in the world. A New England snowstorm, a sickness, and an out-of-town business meeting are all legitimate excuses.

Then, too, there were out-of-state members who joined that the

NU.C.

TI group expecting to get a fat newsletter, like the ones produced by Los Angeles and Chicageo, and were upset when they didn't. The out-of-staters were further upset when they came across copies of the meeting newsletter and could not understand why they were not receiving it. This situation typically arose when an out-of-stater belonged to a user group that exchanged newsletters with the BCS TI group, because it is the meeting newsletter that is used in the exchange program.

In an effort to make things a tad more equitable, it has been suggested that the two newsletters merge and that the resultant single newsletter be mailed to all BCS members who elect to join the TI-99/4A group. This suggestion was helped by the fact that the BCS main office has offered to do the copying and mailing of the newsletter. However, it appears that the exchange program will still be handled as before.

The only thing that remains to be done is to synchronize the new operation. It is desired to have the new newsletter mailed out to members so that they get it before the monthly meeting, which is on the third Wednesday of each month. This means that all contributors must shorten their submission deadline — sort of like everybody in a marching army troop changing step together.

However, once all the problems are straightened out, we hope that each BCS TI-99/4A member — both local and out-of-state — will benefit from the change.

#### From Fayuh to Classic Fair?

There seems to be a lot of change in the air. Another suggestion has been made that the BCS TI group change the nature of the annual TI Fayuh.

Until now this has been a pure TI affair. But, following on the idea used in Computer Shopper, it is proposed that we enlarge the scope of the Fayuh to include other "classic" computers, such as Commodore, Atari and even IBM. Yep. He who owns a PCjr is also just another orphan.

The idea is that such an event is more likely to attract generic vendors — such as disk drive manufacturers and diskette suppliers — and is also more likely to be a commercial success.

A step in this direction has already been taken for the last two years at the TICOFF show in New Jersey. Although I tend to breeze by the Atari and Commodore tables, it can be instructive to see what the competition is capable of. Your comments on such a change are invited.

#### Another expansion box — 2

Here is more complete information on the expansion box mentioned last month (page 4) that is planned for release through Genial Computerware.

The box's approximate size is 13.5"W x 6.5"H x 11.5"D. It has a steel base and an aluminum top. It has 6 horizontal slots for cards (TI uses vertical) and the power supplies: a 150W supplies.

for floppies or hard drives which has four cables carrying +5V and +12V, and a supply for the cards capable of delivering 4A at +7V, 2A at +15V, and 3A at -15V. Remember that TI expects each card to be able to stabilize its own voltages from those on the bus.

The box has provision for mounting one full-height drive. It has a fan which is used to cool the drive and power supplies. In addition, there is a front connector for a keyboard, which would be very convenient for Geneve users.

Unlike the TI box, cards do not protrude from the rear. This has the advantage of protecting cards like TI's RS232 and Myarc's Geneve when in transit. However, it may make some cards a little difficult to use, such as TI's p-code card which has a switch to enable it

According to Corson Wyman of Genial, it is hoped to have the box ready for sale in October. The price will be approximately \$150. For up-to-date information you can contact: Genial Computerware, PO Box 183, Grafton, MA 01519.

#### Keep the dust out

It is said that a 24-can soda carton from the grocery store makes a perfect cover for the TI-99/4A console. All you need to do is cut openings for the cables and the foot of the PEB cable.

#### **EZ-Drawer by Graphic Xpress**

A program called EZ-Drawer by Graphic Xpress is available from Mark Sisco, Graphic Xpress, 4564 Ridgebury Drive, Dayton, OH 45440. It is a disk-based assembly language bitmap graphics generator.

A flyer for the program claims that you can use joystick, keyboard or trackball to change background and foreground colors with ease. You have available 31 graphic commands to draw circles 4 ways, undo mistakes, draw lines, rectangles, boxes, triangles, and freeform objects.

The program also has a multi-color text mode, 4 auto-fill modes. You can save pictures on disk for later viewing or editing and dump pictures to an Epson or TI Impact printer.

The program requires 32K memory, 1 disk drive, E/A, Xbasic or Mini-Memory (must specify when ordering). The price of \$15 + \$1 shipping and handling includes a 32-page manual.

I recently had the luxury of a factory upgrade to my Gramulator that allows me to save and load Milton Bradley MBX cartridges.

The upgrade consists of the addition of four chips — one 13-Input NAND gate, two dual-input NOR gates and a dual D flipflop — a switch and some wiring changes. Externally, it is easily spotted by the maverick switch that is mounted at the right front corner on the Gramulator's top surface. The switch is labeled \*\*\*EX/NORM.

For most operations you leave the switch set at NORM. This even applies to Saving a Cartridge. I lowever, when loading a cartridge you will see that the FILES LOADED INTO display now includes MBX RAM 3, MBX RAM 2, MBX RAM 1 and MBX RAM 0. Appendix E of the Gramulator manual shows the physical addresses where these files are loaded. On an MBX Gramulator the two extra 8K MBX RAM banks (2 and 3) are used.

When running a loaded cartridge from power-up, be sure to follow the manual directions. If you are running an MBX cartridge set the MBX/NORM switch to MBX, set Write Protect to OFF, and set Bank Switching to ON.

Incidentally, most of the MBX cartridges were auto-start. You will have to follow the procedures in Appendix A of the Gramulator manual to either eliminate the auto-start, or purge the cartridge from Gramulator memory. The manual recommends using a business card between the battery and clip as a way to remove power from memory. Not having a card at hand, I found that a TI overlay strip works just as well.

For me, the prime purpose of having the Gramulator was to protect my cartridge collection. Now this useful and invaluable device has been extended so that I can include my MBX cartridges as well. The ability to save MBX cartridges is unique to the Gramulator. Neither the MG Gram Kracker nor the Mechatronics Gram Karte can do this.

The MBX upgrade is available as a kit costing \$15. This includes the four chips, switch, wire and hookup directions. Alternatively, you can return your Gramulator to the factory, but the charge will be \$50.

The Gramulator and the optional MBX upgrade kit is available from CaDD Electronics, 52 Audubon Road, Haverhill, MA 01830. Note the new telephone number: 603-895-0119.

CaDD is now in a position to deal with overseas orders, having worked its way through the requirements of the US Customs service and postal regulations.

Also, CaDD is preparing to release version 1.1 of the Memory Editor in November. In general, this will be a tightening up and improvement of the code, but it will include the ability to do a block memory move.

If there are any other features you would like in the Memory Editor, now's the time to write in and ask. No promises, but if the idea is do-able and there is enough room, CaDD will try and incorporate it. Now there's an opportunity that no Gramulator owner should let slip by.

#### Nostalgia corner

This one comes from a 1983 Texas Instruments advertisement. The headline says: "TI-99/2 Basic Computer". There is a picture of Bill Costa laiding a 99/2 so that the entire keyboard can be seen — sort of the those drink commercials where the fingers

have to be carefully molded to the can so they don't obscure the name. The slogan in the TI ad reads: "The one to start with and get smart with".

Unfortunately for TI, Commodore was the one that got smart — with its marketing. They pressured TI to drop the price on the 99/4A so low that the 99/2 was no longer a contender in the computer wars of 1983.

#### Triton Fall 1988 Catalog

It's good to see that Triton is still supporting the TI-99/4A user. Their Fall 88 catalog has been released and contains the usual mix of interesting and ho-hum items at interesting and ho-hum prices.

Both the Editor/Assembler and TI Writer are now priced at \$14.95. There can't be many people who don't have these, but if you don't they're a steal. Logo II has dropped to \$11.95, an incredible price for a fully featured programming language that includes features such as sprites.

On the other hand, cartridges such as The Great Word Race for \$22.95, Stargazer 1/2/3 for \$39.95 and Scholastic's spelling for ages 11-12 at \$17.95 seem a bit high — mostly because we've become conditioned to cheap software in the TI world.

Triton is still plugging the Triton Turbo XT at \$499, but I haven't met many people who have one, or that intend to buy it. For about the same price these days you can pick up a complete clone, so why bother with the bridge box and the TI keyboard? If you're not on their mailing list and would like a catalog, write to: Triton Products Company, PO Box 8123, San Francisco, CA 94128. To order, call toll-free 800-227-6900 Mon-Fri 6am to 6pm, Sat 9am to 4pm Pacific time.

#### c.Column

# By Donald L. Mahler

First of all, some odds and ends:

- 1) Clint's QDE,QDA, etc. will not run in MDOS1.06 (in fact. he wrote a new version of QDA!) BUT the original programs run without problems in MDOS 1.08! (the modified QDA will NOT)
- 2) If you had problems with Kirkwood's SEARCH program in the Feb MICROPENDIUM, do not dismay; the main program was correct, but there was a typo in the supplied strien(); it should read:

Another form of this function, using an array instead of a pointer:

```
/* (? Warren Agee)*/
strlen(s)
char s[];
{int n; n=0;
while(s[n])
    n++;
return(n);}
```

The program this month is greatly modified from one I downloaded from CIS; the original used fixed values for the number and base, so I made it variable, but the assembly portion which is the essential function is not my work. As it stands now, you can change any number to any base:

```
/* CHANGING BASES */
/* extracted from C.Pulley's RUNOFF by W.Agee
/* modified by dlm
                      */
extern printf();
extern scanf();
main()
char s[20];
 int n,b;
 puts("Enter number: ");
 scanf ("%d", &n);
 puts("\n Enter base to use: ");
 scanf ("%d", &b);
 printf("%d in base %d is: ",n,b);
 itoneb(n,s,b);
 puts(s);
#asm
ITONER MOV
            02(14),7
                          BACE
       MOV
            @6(14),2
                          N
       LI
             6,BUF#
                          WORK BUFFER
       CLR
                          DIGIT COUNTER
ITONB1 CLR
            1
       DIV
            7,1
                          N/BASE, REM IN 2
       AΙ
             2,48
                          MAKE CHAR
       CI
            2,58
       JL
            ITONB2
                          IF REM<2
       AΙ
            2,7
ITONB2 SWPB 2
       MOVB 2, *6+
                          TO BUF
       INC
                          COUNT
       MOV
             1,2
                          QUOTIENT
       JNE
            ITONB1
                          IF MORE DIGITS
            @4(14),7
       MOV
                          S
ITONB3 DEC
            6
                          STORE IN REVERSE
       MOVB *6, *7+
       DEC
            5
                          IF MORE
       .TNF:
            ITONB3
       SB
             *7,*7
                          ZERO BYTE
       В
             *13
```

BUF#

#endasm

22A

# Review of TI-Base By Joe Rawlins

This is my first attempt at a product review, I hope not my last. I am tardy in getting it to Peter, he requested it for the August meeting. [mind you, I'm not complaing. I'll take anyhing anytime I can get it. -jph] I just could not get into the program. I had no motivation. For me to use a program I need a reason that my mind thinks is important.

I wanted to use TI-BASE on a Church mailing list that I maintain in PRBASE. However, that reason soon vanished when I found that there was not any simple way to convert my PRBASE file to TI-BASE. Idid not want to enter almost 400 records manually again (Dennis Faherty has told me that he will be working on utilities to handle this as soon as he has finished Version 2). I still needed a reason, which happened on Friday September 2nd. A data-base of 286 AT clones.

Some basics of TI-BASE Version 1.1. It is a relational data-base with a command language similar in structure to dBASE III PLUS. The minimum system requirments are 32K and one disk drive, two drives and/or a RAM disk are highly recommend. For your money (\$24.95 + \$2.50 shp) you receive a 40 page user's guide, a TI-BASE systems disk, a supplemental TUTOR disk, and a keyboard overlay. The disks are not copy protected so that a working copy may be made and the original put away for safe keeping.

TI-BASE is easily loaded from the Editor/Assembler, Mini Memory, or Extended Basic modules, the procedures are explained well in the documentation. However it does not tell you that with a TI disk controller it takes 1 minutes and 50 seconds to load (an eternity when you are use to loading programs from a RAM disk). You can load from a RAM disk by placing all the files on the system disk on your RAM disk, however your RAM disk must be named TIBASE and can not be numbered greater than DSK3 (hopefully the latter stipulation will be changed with Version 2). Even on the RAM disk load time is 40 seconds.

After the system is loaded you are prompted to enter the date in the format MM/DD/YY, there is not any error checking done on this entry and anything or nothing may be entered, however the DATE directive will not respond with the correct information if it is not entered correctly. The system will then attempt to execute a file named SETUP from the drive which TI-BASE was loaded. SETUP contain the system default status and anything you want to include.

You are now presented with a screen showing your system status, as described in the SETUP file, and the imposing "." prompt at the lower left of the screen above the activity bar. On page 2-1 of the documentation, a command file named TUTOR on the supplemental disk is mentioned. A new user is advised to run this file from the dot prompt by typing DO TUTOR. It may answer a lot of questions for the novice user. Help is also available from the dot prompt by pressing Function-7. With the screen help you get basically the same dialogue and

examples as in the manual for the covered areas. If you didn't understand the manual the help screens are not much help (dBASE III+ suffers from this same problem), however if you have just forgotten a directive structure and you know it is on the help screens it is handy.

To use a data-base you must have a data-base. There are several on the TUTOR disk, used as examples and in the command file TUTOR. You must CREATE your own for your specific application. A new data-base is defined with the directive CREATE DSKn. Filename at the dot prompt. You now have an interactive screen where you enter Fieldnames, field Type, field Width, and Decimals(if Type is N). You may define up to 17 fields for your data-base. If you need more fields in your data-base a second, or up to 5 data-bases, each with up to 17 fields, could be related by a common field and open at the same time in TI-BASE. Once all the fields have been defined Function-8(EXECUTE) creates the data-base and asks if you want to input data at this time. A yes will let you enter data VIA an interactive display. No returns you to the dot prompt and the data-base structure is ready for you to USE on the specified disk.

If you have only a single unrelated data-base and wish to enter data simply type, at the dot prompt, USE DSKn.Filename and then APPEND when the dot prompt returns. This gets you the interactive data entry screen or EDIT (record #) to edit existing data, much like PRBASE. If you have created several data-bases that are related and you want to enter all the data at once for each related record a command file is convenient if not necessary.

Command files may be created with the EA Editor or TIWriter(PF option C), a more convenient tool is included with TI-BASE. MODIFY COMMAND editor is entered from the dot. prompt by typing MODIFY COMMAND (filename). You have a full screen 40 column text editor that will accept about 40 lines of text using the EA or XB modules, or 102 lines using the MM module. I have found that 40 lines are adequate since you may next command files up to 5 deep (40\*5=200). The size of a command file that may be loaded into the editor can become smaller if you enter the editor with data-bases open or local space not cleared. If you do this and your file is too large to fit the available memory you receive an error message stating "can not get dynamic memory" and you are returned to the dot prompt. You may now CLOSE ALL data-bases, CLEAR LOCAL, and reenter the editor. The portion of your command file that would not fit is GONE! GONE! GONE! Hopefully this will be corrected in Version 2.

I created two data-bases and a short command file to enter data in sequence in one and then the other using the REPLACE directive on the second to relate the two with the same Model name.

\* Command file CLONE

SET TALK OFF SET RECNUM ON

```
SET HEADING ON
CLEAR LOCAL
LOCAL AGAIN N 2
REPLACE AGAIN WITH 1
CLEAR
SELECT 2
USE SPEC
SELECT 1
USE N286
WHILE AGAIN=1
  APPEND
  REPLACE TOTAL WITH PRICE + EGA_MON_;
  PR + HD PRICE
  SELECT 2
  APPEND BLANK
  REPLACE 2.MODEL WITH 1.MODEL
  EDIT
  CLEAR
  SELECT 1
  WRITE 10,5 "1. For another entry"
  WRITE 12,5 "2. To end entries"
  READ 14,20,ACAIN
ENDWHILE
CLEAR
WRITE 10,11 "CLOSING DATA BASES..."
CLEAR LOCAL
CLOSE ALL
CLEAR
RETURN
```

By using APPEND, APPEND BLANK, and EDIT directives I was able to write a relatively simple command file to enter my data (I also used REPLACE to calculate a TOTAL so that field did not have to be input). I have included this command file to show some of the power that is available with TI-BASE and a little work on your part.

The structure of your data-base may be changed with the directive MODIFY STRUCTURE. Fieldnames may be changed with out disturbing the data, however if the record size is change existing data will be deleted. A way around this is to duplicate the data-base structure and data files with different names, make your changes to the duplicate, then wright a command file to replace each field of each record with data from the old file.

UH-OH!! This review almost turned into a tutorial.

Back to the review. The command language contains 45 directives for data manipulation and disk management. Disk management directives include: CATALOG, COPY, DELETE, and FORMAT, all of which seem to operate faultlessly.

There are a few BUGS with the directives READ and REPLACE. READ will not accept character input into a character variable. READ and REPLACE used with a numeric variable or field need the variable or field to be one space greater than what is read (ie. a numeric field, width defined as 1, and you try to REPLACE fieldname with the number 2, will result in an \* being place in the field to indicate that the field size was too small). Hopefully these will be corrected in Version 2

also.

The 40 page manual is barely adequate for a novice data-base programer. Most examples deal with numeric rather than character handling.

The manual should never have made it out the door of a good printing shop. On most pages the letters are broken and the punctuation marks are hard to read. Other pages have pluggy letters, while yet others have dark and light print on the same page. Not very professional.

I would recommend TI-BASE to anyone who needs more than PRBASE offers and can do a little programing. Actually a quite impressive performer for the memory constraints imposed by the TI.

# Random Ramblings

By J. Peter Hoddie

Over the summer I presented a number of assembly language programming examples. The response was not exactly overwhelming, so this month I'll switch gears and try writing something about the language that we all love to hate - BASIC. Mostly I'll stick with Extended BASIC since if you don't got that, well 'nough said. I'm also going to try to keep each program as short as possible. If you're like me you probably hate keying in long programs. Besides they are also tougher to read. ALL the programs in this article take up only 3 or 4 sectors on disk. Yet they all do something almost useful!

First I'd like to present the one program I've probably typed in more times than any other. This is program just totals a whole pile of numbers as you type them in. It acts like a primitive calculator. And my 9640 is much less likely to be buried under papers on my desk than a little calculator.

```
10 T = 0
20 INPUT Z
30 T=T+Z
40 PRINT T
50 GOTO 20
```

This program has some pretty exciting features. It displays the last 12 values entered on the screen, it displays the total after each number, and it is rather short. To clear the total back to zero either restart the program, or enter the negative of the last displayed total number. OK. A bit dull? So I'll try something else.

What follows is a set of programs which will allow you to maintain a catalog of all the files you have on all your disks. Or at least a whole bunch of them. To perform sorting on the list, I will explain how to use my sort program on BCS disk XXXXXX to do that, since assembly language still has a few advantages. This set of programs is user hostile, mostly. It is not intended to be a commercial application. It does illustrate that a little BASIC code goes a long way. You can add disks to the list, create a sub-list based on several criteria, print the list or a part of it, search it, and even edit it in TI—Writer if so inspired. Some of you will see this code, and have ideas for improvements. Try



it out. Please send me a copy of your version so it can be printed here for others to use.

The first program is what I call "Main." It when you run it, it adds then disk located in DSK2 to the database file you specify. If the database file does not exist, a new one is created. You are not warned of this occurrence. To add another disk, change the disk in drive two and run it again. If you only have one drive, sorry. The program could be modified for that situation, but I didn't do it that way in this version. If you prefer not to use DSK2 as the input drive, change the code in line 150.

For those of you into serious XB programming, the PRINT USING clause in line 220 is worth checking out. Its a really easy way to right justify numeric fields. I suspect there is a more elegant approach but my XB manual is at home, and I'm at school so my memory will have to do. Here's the program (3 sectors on disk):

```
100 IMAGE #####
110 DIM FN$ (127), FS (127), FT (127), FL (127)
120 INPUT "Database file: ": 2$ :: OPEN
#1:Z$, APPEND
130 FOR Z=1 TO 7 :: READ T$(Z):: NEXT Z
140 DATA DIS/FIX, DIS/VAR, INT/FIX, INT/
VAR, PROGRAM, DIR, EMULATE
150 OPEN #2:"DSK2.", INPUT , INTERNAL, FIXED
160 INPUT #2:DN$ :: Z=0
170 INPUT #2:FN$(2),FT(2),FL(2),FS(2)
180 IF FN$(Z)="" THEN 210
190 Z=Z+1 :: GOTO 170
200 CLOSE #2
210 FOR Z=0 TO Z-1 :: PRINT
#1:FN$(Z);TAB(12);T$(ABS(FT(Z)));TAB(20);
220 PRINT #1, USING 100:FS(Z);:: PRINT
#1:TAB(28);:: PRINT #1, USING 100:FL(Z);::
PRINT #1:TAB(35);DN$ :: NEXT Z
230 CLOSE #1
```

The format of the output file is Display/Variable 80, so it can be edited in TI-Writer. Each file on the disk is listed on a separate line. The filename is listed, followed by its type (DIS/VAR, PROGRAM, etc.). Then the file length in sectors is given. In the final field, the name of the disk that the file was on is listed. The fields are listed in columns which makes sorting a piece of cake.

Before going on to the next "big" piece of the program, I want to present another of my favorite short XB programs. This one just lists a file to the screen. It is very useful for looking at a file without having to load up TI—Writer or whatever. I include it here because it can be used to look at a database file quickly.

```
10 ON ERROR 50
20 LINPUT "Filename: ":A$
30 OPEN #1:A$, INPUT
40 LINPUT #1:A$ :: PRINT A$ :: GOTO 40
50 END
```

The next program has many functions. With this program you can search a database, print it out, print selected records, copy

a database, combine two databases, create a new database using all or some records from an existing database, and probably something else. You can search for a match, non-match, for a file that contains or does not contain a string, and for a field that starts with a string or does not start with a string. In addition you can request all records.

This program is really a general purpose engine for processing an input file. It could be used on a database containing other types of data with little or no effort. You could use TI-Writer to enter the data, or create another program to build the database, like as the one that scans disks above.

Here's the program. Instructions follow the listing. (4 sectors on disk)

```
5 ON ERROR 1000 :: ON WARNING NEXT
10 INPUT "Input file: ":I$
15 OPEN #1:I$, INPUT
20 INPUT "Output file: ":O$ :: IF O$="" THEN
30 ELSE 0=2
25 IF ASC(O$) = ASC("+") THEN OPEN
#O:SEG$(O$,2,LEN(O$)-1),APPEND ELSE OPEN
#0:0$, OUTPUT
30 INPUT "Start column: ":P1 :: INPUT
"Length: ": P2 :: IF P2<1 THEN 30
40 PRINT "1 for EQUALS": "2 for CONTAINS": "3
for STARTS WITH":"4 for ALL" :: INPUT "Your
choice: ":T :: IF ABS(T)>4 THEN 40
45 IF T>0 THEN TF=-1 ELSE TF=0
50 T=ABS(T):: IF T<>4 THEN LINPUT "Search
String: ":S$ :: IF LEN(S$)>P2 THEN 50
100 LINPUT #1:Z$
105 ON T GOTO 110,120,130,140
110 IF (SEG$(Z$,P1,P2)=S$)=TF THEN 140 ELSE
150
120 IF (POS(SEG$(Z$,P1,P2),S$,1)<>0)=TF THEN
140 ELSE 150
130 IF (SEG$(Z$,P1,MIN(P2,LEN(S$))-S$)-TF
THEN 140 ELSE 150
140 PRINT #0:ZS
150 GOTO 100
1000 END
```

The program starts by asking for an "Input file." Give it the name of the database file you want to process. Next it asks for an "Output file." To send output to the screen (for your edification or otherwise), just press ENTER for the output file. Otherwise, give it the name of the file to create. To append to an existing file, enter a plus sign ("+") before the filename, and all output will be added to that file. For example, an output file of "+DSK1.FILE" would cause all output to be appended to the file "DSK1.FILE" while an output file of "DSK1.FILE" would cause a new file to be created.

Next you are prompted for a start column, and field length to search on. These refer to the numbers described below. For example, to search the filenames themselves, enter a start of 1, and a length of 10. The length value must be larger than zero.

After defining the field to work with, you are queried for the type of search to perform. Each option is now described.

1 for EQUALS. This allows you to search for all records that match exactly. For example, if you only wanted to find all files named "CONTROL" you would use this option.

2 for CONTAINS: This allows you to search for all records that contain a particular string. For example, if you wanted to find all the files that contained the word "PLATO" somewhere in the filename, you would use this option.

3 for STARTS WITH: This allows you to search for all records that begin with a particular string. For example, to find all files that being with the letters "ED."

4 for ALL: This option automatically selects all records.

To select the search method, enter the corresponding number. You can "NOT" any of these options by entering the negative of the number. For example, entering -2 tells the program to select all records that DO NOT CONTAIN the search string. This feature can be very useful when trying to eliminate certain files. For example to eliminate all files named LOAD, use search method -1 (NOT EQUALS) with a search string of LOAD.

After you have specified the search method, you must specify the search string at the next prompt. The search string prompt is skipped if you selected 4 for ALL. Once the search string is specified, the program goes to work reading the input file, and writing the match records to the output file.

[WARNING: If you search using the EQUAL TO option, the search string you enter must be exactly the same as the field you are searching. This includes length. If you are searching the 10 character filename field for the file "EDITOR" you must enter it with 4 trailing spaces, or there will be no match. EQUAL TO means what it says. In most cases you will probably want to use the STARTS WITH option (why do you think I put it there????)]

That's all there is to using the program. I suppose I should now explain how to do the operations described in the introduction, so here goes.

To combine two files, use the name of the file you want to append as the input file, and the name of the file to append to as the output file, preceded with a plus sign.

To print a file, specify an output file of "PIO" and a search method of ALL (4).

To print records that match a search specification, enter an output file of "PIO" and enter the search criterion normally.

To copy a database, enter the original name for the input file, the new name for output file, and specify search method of ALL (4).

This program could be modified to be much more user friendly. For example, instead of entering the Start Column and Length it could be changed to present a list of field names to search. Other search options could be added. A major improvement would be to do all screen access with DISPLAY and ACCEPT AT, which would keep the screen from scrolling so much and generally look a lot more professional.

For the disk files database, the field layout is as follows:

<u>Contents</u>	Start	Column	Length
Filename	1	10	
Filetype	12	7	
Record Size	20	5	
File Size	28	5	
Diskname	35	10	

Using the two main programs presented here, it is possible to do most basic database operations. You can add records. search, print, select, merge, create sub-databases, and even edit if you use TI-Writer. The missing feature is sorting. Sorting in BASIC is slow. A good solution is to use an existing assembly language sort program. It just so happens. I wrote one of those, and it is on BCS disk #83, complete with source code. Others are available. If you got serious about modifying this program I would recommend Andy Dessoff's sort package with works within Extended BASIC, so you can call it from your XB program. It is probably the fastest, most full featured sort available from XB. If you can deal with loading up a 100% assembly language program, I believe my sort is a bit faster, with a larger capacity. The other nice thing about using mine, is that is works with the Start Column and Length numbers above. You don't have to count columns to determine your fields. Note that my sort program is also fairware and if you start using it you owe me a few bucks.

Another program that would be nice to add to this system is one which allowed you to print out reports in a format that wasn't just a record dump. Alas this would be challenging to fit into 5 sectors, and I just don't have the time this month.

Someone with some time and inspiration could turn these programs Into something truly useful. This son of BASIC programming is what I do to proto-type some assembly routines. Its a good way to prove that an algorithm or concept works before going to the pain of coding it in assembly. I also hope that this column proves that you can do useful work in a short BASIC program. Barry Traver has spent years preaching the virtues of Extended BASIC. He does have a point. The one thing that is lost in using BASIC is speed of execution. On the other side, development time is cut by using a high(er) level language.

For those of you into really serious BASIC programming, there are some pretty neat tricks in these short programs. There includes a pretty painless way to output to the screen or disk, how to handle both negative and positive logic, and something else. Perhaps in a future column I'll get around to explaining how the stuff really works.

# Bitmap mode on the TI-99/4A

by Mike Wright. Version 08-Aug-88

Supplement to the BCS TI-99/4A User Group September 1988 meeting newsletter

There are four graphics modes available on the TI-99/4A. They are called:

#### Graphics

The display is 24 rows by 32 columns. This is the mode used by Basic and Extended Basic.

#### Text

The display is 24 rows by 40 columns. This is the mode used by TI-Writer and the editor supplied with the Editor/Assembler.

#### Multi-color

The display is 48 rows by 64 columns. There is not much practical application for this mode.

#### Bitmap

The display is 192 rows by 256 columns making a total of 49152 pixels. This mode cannot be accessed from Basic or Extended Basic because it requires too much VDP memory. It can be accessed from a TMS-9900 assembly language program, or from Forth.

The TMS-9918A Video Display Processor (VDP) is capable of displaying 16 colors. It would seem desirable if each of the 49152 pixels could have its own color. This is not possible in the 16K of VDP RAM on the TI-99/4A.

A display of 192x256 requires 49152 bits. At 8 bits to a byte, this requires 6144 bytes just to keep a record of which pixels are on or off.

Now add the requirement that each pixel be 1 of 16 colors. 16 colors can be represented by 4 bits. Thus each pixel would need 5 bits (1 for on/off, and 4 for color) to represent this. This calculates to 192x256x5 or 245760 bits. At 8 bits to a byte, this requires 30720 bytes — nearly double the 16384 available in VDP RAM.

Instead, TI compromised and allowed 8 consecutive pixels to share the same foreground and background color. One byte (8 bits) is used to record 16 foreground and 16 background colors for 8 pixels. Thus the size of this color table calculates to:

 $192 \times 256 / 8 = 6144$  "pixels"

Even with this reduction, the color table takes up more than a third of the available VDP memory.

As an exercise, you should calculate the amount of VDP RAM required by a 640x200 display with each pixel having 16 colors.

### **VDP write-only registers**

When an assembly language program is run using

#### 3. LOAD AND RUN

from the Editor/Assembler module, the VDP processor is in graphics mode. To change the mode you must place values in the 8 write-only registers of the TMS-9918A VDP processor.

There is no direct TMS-9900 assembly language instruction that allows you to do this. Instead you have to write to the port connecting the VDP processor and the 9900 central processor. However, it is easier to use the TI-provided utility called VWTR.

The 8 VDP registers are described in Sec. 21.1, p326 of the Editor/Assembler manual. VWTR is described in Sec. 16.1, p249 of the Editor/Assembler Manual.

To write to a VDP register using VWTR you place the VDP register number and the value to be written in R0. You then do a BLWP QVWTR. The VDP register number is placed in the left byte of R0. The value to be written is placed in the right byte of R0.

For example, to write the value >2C to VDP register 7:

LI RO,>072C BLWP @VWTR

#### Enabling bitmap mode

To enable bitmap mode, you must set up the VDP registers. The values you place in these registers tell the chip where to place certain tables in VDP memory. By reading the TI Editor/Assembler manual (Sec. 21.1 VDP write-only registers, p326) you may get the impression that there is a lot of flexibility in placing these tables when enabling bitmap mode.

For example, the manual says that the base address of the color table is calculated by multiplying the value placed in VDP Register 3 by >40. This implies that the color table can be located at various locations in VDP memory. In practice, the various tables nearly always appear in the same place in VDP memory.

There are also errors in the Editor/Assembler manual which are confusing when trying to work out where the tables should go. For example, on p335 the manual originally said that a value of > 04 was required in VDP register 3 to place the color table at > 2000. A subsequent correction published by TI changed the value to > FF. If you then read p327 and multiply this



value by >40, you will find the base address is theoretically >3FC0, which is not the correct answer.

To avoid this kind of frustration, use the values in this article. They do work.

The tables that have to be set up are:

#### Pattern Descriptor Table

This starts at VDP > 0000 and is 6144 bytes long. You can think of it as being a "CALL CHAR". To describe a "character" you need 8 bytes. Since there are 768  $(32 \times 24)$  characters on the screen, you need  $8 \times 768 =$ 6144 bytes for the table.

To place the table:

LI RO,>0403 **@VWTR** BLWP

#### Screen Image Table

This starts at VDP > 1800 and is 768 bytes long. Each byte points to an 8-byte pattern in the Pattern Descriptor Table. Since a byte can only have 256 possible values, the first 256 entries in the Screen Image Table point to the first 2048 entries in the Pattern Descriptor Table, the next 256 entries in the SIT point to the next 2048 entries in the PDT, and the last 256 entries in the SIT point to the last 2048 entries in the PDT.

This makes a total of 768 entries in the SIT pointing to  $768 \times 8 = 6144$  entries in the PDT.

To place the table:

RO,>0206 LI BLWP **@VWTR** 

#### Sprite attribute list

This starts at VDP > 1B00 and is 256 bytes long. Each sprite attribute takes 4 bytes. They represent the bit row, bit column, character code, and early clock (left nybble) and color (right nybble) of the sprite.

To place the table:

RO,>0536 LI BLWP **@VWTR** 

The sprite descriptor list will then start at VDP >1C00 and the sprite motion table will be at VDP >1F00. There is no auto-motion of sprites in bitmap mode.

In addition, all unused sprites should be disabled by placing a byte of > D0 after the last sprite entry. Since we are not using sprites, it should be placed at the beginning of the Sprite Attribute List at >1B00.

To disable sprites:

LI RO,>1B00 LI R1,>D000 BLWP @vsbw

#### Color table

This starts at VDP > 2000 and is 6144 bytes long. Each byte describes the color of 8 consecutive pixels. The left nybble describes the color of the pixels that are on. This can be one of the 16 available colors. The right nybble describes the color of the pixels that are off. This can be one of the 16 available colors.

There are 256 x 192 - 49152 pixels on the screen. One byte is required to describe the color of 8 pixels. Therefore the length of the table is:

49152 / 8 = 6144 bytes

To place the table:

RO,>03FF LI ■ BLWP **@VWTR** 

#### Initialize the Screen Image Table

The SIT is 768 bytes long. It needs to contain > 00through >FF repeated three times. Each byte will then point to a corresponding 8 bytes in the Pattern Descriptor Table.

To initialize the SIT:

	LI	RO,>1800
	CLR	R2
BIT1	CLR	R1
BIT2	BLWP	@VSBW
	INC	RO
	AI	R1,>0100
	CI	R1,>0000
	JNE	BIT2
	INC	R2
	CI	R2,3
	JNE	BIT1

#### Initialize the Pattern Descriptor Table

The PDT is 6144 bytes long. It starts at > 0000 and ends at >17FF. To initialize it, we place 0's throughout the table. This means that no pixels are on.

To initialize the PDT:

CLR RO CLR R1 BLWP **@VSBW** BIT3 INC RO CI RO,>1800 BIT3 JNE



#### Initialize the Color Table

The Color Table is 6144 bytes long. It starts at >2000 and ends at >37FF. To initialize it, we place 0's throughout the table. This makes the foreground and background color of any pixel to be transparent. Thus even if a pixel is on, it will have the same color as the screen

To initialize the Color Table:

	LI	RO,>2000
	CLR	R1
BIT4	BLWP	@vsbw
	INC	RO
	CI	RO,>3800
	JNE	BIT4

#### Enable bitmap mode

Bit 6 of VDP Register 0 must be set to enable bitmap mode. Bits 0-5 must be off. Bit 7 is always 0, unless you are using an external video input signal to the VDP chip.

To enable bitmap mode:

LI	RO,>0002
BLWP	<b>@</b> VWTR

#### Set the screen color

You should select the background screen color. If not it will default to the standard Editor/Assembler green.

To set the screen color to transparent:

LT	RO,>0700		
RT.WP	<b>ØVWTR</b>		

#### Manipulating the tables

Once VDP tables are in place and initialized, and bitmap mode is enabled, you make changes to the screen display by changing values in the tables.

The most fundamental problem to solve is the addressing of a single pixel.

The bitmap screen has 192 rows, numbered 0—191, and 256 columns, numbered 0—255. It would have been convenient if the bitmap was linear, but on most computers the design of the VDP chip precludes this.

The TI implementation has pixels 0—7 at the top left corner of the screen. Pixels 8—15 are immediately below 0—7. This arrangement continues for a total of 8 rows through pixel 63. Pixel 64 is then found on the top row, next to pixel 7. In effect, you have 8x8 "characters", each containing 64 pixels.

	Pixel numbers on bitmap screen															
0	1	2	3	4	5	6	7	3	8	66	67	68	69	70	7)	·
8	9	10	11	12	13	14	15	72	73	•			,			-
16	17	18	19	20	21	22	23	80	81	•	•	•	•	,	Ė	•
$\Box$	•	-	•	-			-			٠	1	Å	٠	Ė	,	•
56	57	58	59	60	-61	62	63	120	121	122	123	124	125	126	127	
Ŀ	•	-		-		-	-	•	-	•	•			,	-	-

These 8x8 units continue horizontally for 32 "characters". At this stage the last pixel in the first "character row" (8th pixel row) is 2047. At the start of the next pixel row, pixel 2048 is below pixel 56.

Although we wish to deal with single pixels, in practice the smallest unit we can read or write from VDP memory is a byte. The description of the bitmap memory in bytes is similar to that for pixels.

Byte 0 is at the top left corner of the screen. Byte 1 is below it. This continues for 8 bytes. Byte 8 is then next to byte 0. This continues to byte 255 (the end of the first 8x8 row of the screen). Byte 256 is immediately below byte 7, and so on.

Thus, to change a pixel on the screen, we need to calculate the byte in the Pattern Descriptor Table. Once we have the byte, we need to set or clear the bit representing the pixel and write the byte back to the PDT.

The following Basic program will calculate the byte number given the row and column:

100 INPUT ROW
110 INPUT COL
120 WHOLEROW = INT(ROW/8)\*8
130 WHOLECOL = INT(COL/8)\*8
140 REMROW = ROW - WHOLEROW
140 BYTE = (WHOLEROW\*32)
+WHOLECOL+REMROW
150 PRINT BYTE

It would be easy to do this in assembly language. However, since multiplications and divides are relatively slow processes, TI provided a routine to do this. It is on p336 of the Editor/Assembler Manual.

To use this routine you must enter it in your program, probably as a subroutine. You then place the row value of the pixel to be changed in R1, its column value in R0, and BL to the routine. It returns the byte to be changed in R4, and the pixel within that byte in R5

The color of the pixel can be calculated by adding >2000 to R4. Remember that the foreground color will affect all on pixels in the PDT byte, and that the background color will show where the pixels are off.

#### References

#### The Editor/Assembler Manual

I believe it is next to impossible to read the description starting on p334 and implement bitmap mode. However, you must read this section and absorb as much as possible.

#### Enthusiast 99, Mar 84, p24

Bill Gronos provides an introduction to bitmap mode. Some of the information is good, but I get confused by the painted eggs and Godfrey Grafix. The table on p30 is laid out very badly typographically. The examples provided do work, although Gronos prefers to write direct to the VDP processor instead of using VWTR. The advantage is that it's faster.

#### National Ninety-Niner, Sep 84

A good example from John Phillips, formerly a TI programmer. He places a sprite hand on the screen and uses it to pull down a "window" of color. Well commented and worth spending some time on.

# Compute!'s Guide to TI-99/4A Assembly Language by Peter Lottrup

The only real source for a discussion of bitmap mode. The examples are designed for the Mini Memory, but are easily converted to run under the Editor/Assembler.

The programs Hi-Res Bouncer (p212) and Hi-Res Draw (p218) do work. Note the error on p213: There is a BLWP @>6034 (VWTR) missing after address >7D24.

	16K VDP RAM in bitmap mode						
Description	Address (hex)	Address (dec)	Size (dec)	Comment			
Free	>3FFF	16383	2048	Disk buffers			
VDP	>3800	14336		and PABs			
Color	>37FF	14335		Each byte controls			
table	>2000	08192	6144	foreground/background color of 8 pixels			
Sprite	>1FFF	08191	256	Auto sprite motion			
motion table	>1F00	07936	256	cannot be used in bitmap mode			
Sprite	>1EFF	07935		8 bytes/character.			
descriptor list	>1C00	07168	768	Starts with > 80 (128).			
Sprite	>1BFF	07167		4 bytes/sprite.			
attribute list	> 1B00	06912	256	Bit row, bit column char code, early clock/color			
Screen	>1AFF	06911		Usually contains			
image table	> 1800	06144	768	>00 - > FF repeated 3 times			
Pattern	>17FF	06143		8 bytes for each entry			
descriptor table	>0000	00000	6144	in screen image table. 8 x 768 = 6144			

Bitmap demo program Supplement to BCS newsletter August 1988 Mike Wright DEF START VWTR, VSBW REF MYWS BSS 32 START LWPI MYWS Step 1: set screen color RO,>0700 LI BLWP **QVWTR** Step 2: enable bitmap mode RO,>0002 LI BLWP **QVWT**R Step 3: Move screen image table to >1800 RO,>0206 LI @VWTR BLWP Step 4: Move the color table to >2000 LI RO, > 03FF BLWP **QVWTR** Step 5: Move the pattern descriptor table to >0000 LI RO,>0403 BLWP **@**VWTR Step 6: Move the sprite attribute list to >1B00 RO,>0536 LI @VWTR BLWP Step 7: Disable all unused sprites RO,>1B00 LI R1,>D000

LI

BLWP

@VSBW

```
LI
                       RO,>1800
           CLR
                       R2
BIT1
           CLR
                       Rl
BIT2
           BLWP
                       @vebw
           INC
                       RO
                                   Next byte in table
                                   Next incremental value
           AI
                       R1,>0100
                       R1,>0000
                                   Passed >FF00 yet?
           CI
           JNE
                       BIT2
                                   No. Do next
           INC
                                   Do next third of table
                       R2
                                   Done three thirds yet?
           ÇI
                       R2,3
           JNE
                       BIT1
                                   No. Do next
           Step 9: Clear the pattern descriptor table
           CLR
                       RO
           CLR
                       R1
BIT3
           BLWP
                       @VSBW
           INC
                       RO
           CI
                       RO,>1800
           JNE
                       BIT3
           Step 10: Clear the color table
                       RO,>2000
           LI
           CLR
                       R1
                       @VSBW
BIT4
           BLWP
           INC
                       RO
           CI
                       RO,>3800
           JNE
                       BIT4
           Put "char" on screen
           LI
                       R6,>0200
                                   Color
BIT5
           LI
                       R7,0
                                   Row Y
BITG
           LI
                       Re,o
                                   Row X
BIT7
           MOV
                       R8,R0
                                   X position of pixel
           MOV
                       R7,R1
                                   Y position of pixel
           BL
                       @ROWCOL
                                   Find byte to change
           MOV
                       R4,R0
                                   R4 has the byte
                       R1,>0F00
                                   "char" -- try different values in MSB
           LI
                       @VSBW
                                   Put "char" on screen
           BLWP
```

Step 8: Initialize the screen image table

	AI	RO,>2000	Now do the color
	MOV	R6,R1	
	BLWP	@vsbw	
	AI	R8,8	Next "char" along row
	CI	R8,255	Reached end of row yet?
	JLT	BIT7	No. do next "char"
	AI	R6,>0100	Next sequential color
	INC	R7	Next row
	CI	R7,192	Done whole screen?
	JLT	BIT6	No. Do next
	JMP	BIT5	Do forever
*	X in RO,	Y in Rl	
ROWCOL	MOV	R1,R4	
	SLA	R4,5	
	SOC	R1,R4	
	ANDI	R4,>FF07	
	MOV	RO,RS	
	ANDI	R5,7	
	A	RO,R4	
	s	R5,R4	
-	RT		
`	END		•