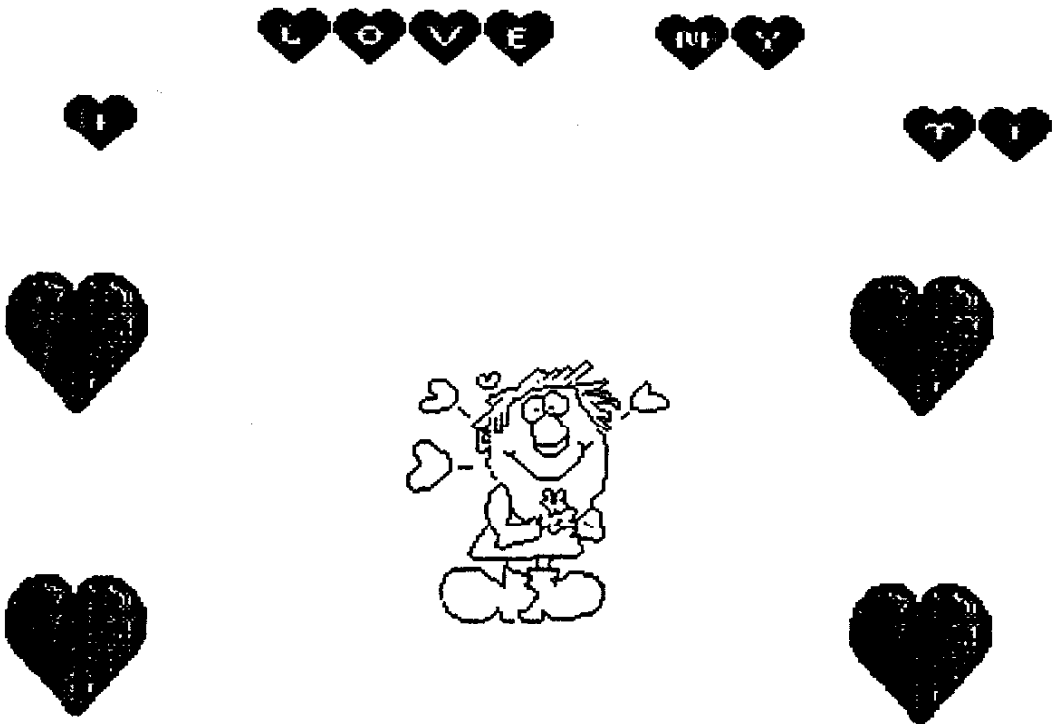
# TI-D-BITS

PHILADELPHIA AREA USERS GROUP NEWSLETTER
COVERING THE TI99/4A
AND MYARC 9640 COMPUTERS

FEBRUARY 1990

Volume 10 Number 2




BE MY

COMPUTERTIME

# THE PHILADELPHIA AREA TI-99/4A USERS' GROUP (FEB '90)

The Philadelphia Area TI-99/4A Users' Group meets twice a month. On the first Saturday of any given month, we meet at the Bucks County Youth Development Center, (YDC, which is next to Neshaminy Mall), Administration Building, beginning at 10:00 am. On the third Saturday of each month, we meet at LaSalle University, 20th Olney, in room H-329 located in the Science Building. Membership to The Philadelphia Area TI-99/4A Users' Group is available to all. We invite anyone that is interested in the TI-99/4A to visit us. Stop in and see what is available to you for your TI and how membership can benefit you!

Current executive board consists of:

| | | |
|---|---|---|
| PRESIDENT..... | Allan Silversteen. | 215-885-7910 |
| VICE PRESIDENT..... | Eric Bray..... | 215-947-7353 |
| SECRETARY..... | Mark Mannop..... | 609-365-1776 |
| TREASURER..... | Don Arsenault..... | 215-368-0446 |

Committees consists of:

| | | |
|---|---|---|
| TI-d-BITS.... | Ralph Field..... | 215-362-2534 |
| | Don Arsenault..... | 215-368-0446 |
| | OPEN | |
| | Rice Hall | |
| LIBRARY ..... | Ted Chemey..... | 215-752-1458 |
| | Rich Mascara | 215-441-4060 |
| MEMBERSHIP ... | OPEN | |
| ASSISTANT TREASURER. | Frank Passini | |
| EDUCATION .... | Barry Traver | |
| | Frank Passini | |
| | Ted Chemey | |
| | Tim Coyne | |
| | Carlo Angelico | |
| EQUIPMENT .... | Rice Hall | |
| PROGRAM ..... | Dr. Eric Bray | |

REMEMBER to be considerate when calling any of the above people. Limit your calls to the early evening hours. (6pm to 9pm)

The opinions expressed herein are those of the individual authors are not necessarily those of the Philadelphia Area TI-99/4A Users' Group or its officers. Nor is the Philadelphia Area TI-99/4A Users' Group or any of its officers responsible for any damage, inconvenience, or loss which may result as a consequence of the use of any written material herein.

TI-d-Bits is published monthly by the Philadelphia Area TI-99/4A Users' Group, c/o Don Arsenault, 1290 Buttonwood Dr., Lansdale, PA 19446. All material herein may be reprinted freely by other non-profit User Groups, (unless otherwise stated), as long as proper credit is given to both source and author. Contributions are encouraged, but no payment is made. Editorial, advertising, and classified, copy MUST be in by the LAST day of the previous month. You can either mail your copy to: TI-d-Bits, c/o The Philadelphia Area TI-99/4A Users' Group, c/o Don Arsenault, 1290 Buttonwood Dr., Lansdale, PA 19446 or send it via modem by contacting Don Arsenault at (215)-368-0446. If your piece contains any diagrams, charts, or code, send a paper copy AT FINAL PUBLICATION SIZE.

Classified ads are printed in blocks. A block consists of 3 lines, 55 characters wide, or any increment of 3 lines. Classified advertising is accepted from members at NO CHARGE for a one block ad, per issue. Additional ads from members may be placed at cost of $1.00 per block. Non members may place classified ads at a cost of $2.00 per block. All advertisements MUST be paid for in advance.

Commercial advertising is accepted for publication at the following rates:

| | |
|---|---|
| Quarter page ..... | $ 5.00 |
| Half page ..... | $ 8.00 |
| Full page ..... | $15.00 |

Commercial advertisements will be placed in the next available issue. All advertisements MUST be paid for in advance.

The editor of TI-d-Bits or the executive board of The Philadelphia area TI-99/4a Users' Group reserve the right to reject any material submitted for publication for any reasons.

The Philadelphia Area TI-99/4A Users' Group's program library is available to all active members at NO CHARGE for copying to your disk. A charge of $2.00 per disk is made for club supplied disks for members. Non members may obtain copies of the library for a fee of $5.00 per disk. A catalog of the library's contents is given to all new members upon request and updates will appear in this publication from time to time. To obtain material from the library, contact the librarian for the best procedure to obtain your requests.

**********
SECRETARY'S NOTES
By P. Mark Wannop
**********

JANUARY MEETING NOTES

The January meeting was held in a room on the FIRST floor of the LaSalle science building, a step in the right direction (unfortunately, most of us didn't find out until we'd climbed up to the third floor). With any luck, the February meeting (last one at LaSalle) will also be on the first floor; at any rate, check the meeting directory posted on the wall (across from the front doors) when you come in...

PATIUG dues are now being collected for 1990; the dues remain at $12.00. Also PACS dues were up in December; while membership in PACS is not required for membership in PATIUG, it should be noted that there will be free parking with a PACS membership card when PACS moves the 3rd Saturday meetings to Drexel in March.

The March PACS meeting at Drexel will be the annual Computer Festival. The TI SIG will participate, probably with a fully equipped Geneve (with hard drive) and a fully equipped TI.

The PACS BBS will be "partly closing" in the near future. PACS members will enjoy full use of all functions of the BBS; non-members can still call and use the BBS, but they will have limited access to the functions.

Speaking of BBS's (notice how I work this in...) Dr. Eric Bray displayed a couple of nice programs that Mike Riccio has been working on, in his spare time out at Carnegie-Mellon University. One is an excellent terminal program for the Geneve 9640.

Called "MYTERM", the program runs out of MDOS on the 9640; the user can set aside a buffer of 8K, 16K, 32K, or all available memory (up to about 900K!). The program takes advantage of the HFDC, if available, by storing directly to the hard drive if desired. The program has advanced windowing capability; one can window while the terminal emulator is running for directories, etc. The progam will emulate VT52 and HP19 terminals, and will run up to 19200 baud. The version Eric showed was the beta test version; the final version should be released in the spring. Price was not available at this time.

Eric also showed Mike Riccio's version of "TETRIS", written for the Geneve 9640; this is a nice implementation of the popular Russian program, and will also be released in the spring. Price to be announced.

We are currently making a mailing of newsletters (along with a cover letter) to members of the defunct 99+ Express group of Deptford, N.J., which disbanded in the fall. While some members of that group have switched to other (read "MS-DOS") computers, there are still some TI stalwarts there; hopefully PATIUG can gain some new members from them. The move of PACS to Drexel might help this, as I think more people from South Jersey know how to find Drexel (as I'm from South Jersey myself...). Drexel is reasonably easy to reach for Jerseyites by way of the PATCO line and the SEPTA Market Street Subway.

Don Arsenault pointed out that many TI groups are either shrinking in membership or being diluted (read "taken over") by MS DOS users. He mentioned that several former TI newsletters have shown up in our newsletter exchange with almost every article being MS-DOS oriented. (I noted this trend myself with the newsletters received by the 99+ Express). We hope to be able to attract members from a wider geographical base as a club dedicated strictly to TI and Geneve. (Frankly, it'd be senseless for us to cover MS-DOS machines even if we wanted to, as this is done admirably in this locality by other PACS SIGs and the PC Club of South Jersey...)

There was some discussion of using the 512K card with the Geneve; also, there is good news for PATIUG members in that Myarc no longer requires us to hav a five-unit minimum for purchases!

Eric Bray began a Geneve workshop; there are several members who are new 9640 owners who need help getting started with the machine. MDOS is relatively similar in operation to MS-DOS, but quite a bit different from the operating system on the TI. If you aren't familiar with MS-DOS (or it's predecessor, CP/M) and have a Geneve, you should look in on Eric's workshop! (For that matter, you should look in on it anyway!!!)

Eric's first point in learning to operate the 9640 is READ THE MANUAL!!! (This goes for ANY hardware or software!) The operating system of the 9640 is more complicated than that of the 99/4A, but it is MUCH more flexible and has much greater capabilities; it is well worth the time invested in learning its operation.

Eric went on to discuss the Disk Operating System (DOS); the advantage of a DOS (rather than having the operating system in ROM as the 99/4A has) is that the operating system can be upgraded or altered as the need arises. (MS-DOS has had a number of upgrades to enable those machines to use newer hardware, and to increase programming flexibility; there have been similar upgrades in Apple DOS - including one complete revision.)

On the 99/4A, some DOS commands were included as part of BASIC, while other functions of disk access and maintenance require a Disk Manager program; a DOS has all

the disk management functions within it. Machines using a DOS require that the DOS be loaded into it before the machine can do anything else; once DOS is loaded, then one can load in a language (BASIC, C, etc.) or execute an "executable" program or a batch file from DOS. (The GPL interpreter that is loaded into the 9640 to run 99/4A programs must be loaded after DOS; it is, in some respects, a "language".)

Eric also discussed batch files; briefly, a batch file is a text file that DOS can read and execute. One can have the machine go though a number of steps by itself by creating a batch file to do so. Once DOS is booted, the 9640 will look for a specific batch file that will autoexecute; one can write this batch file to set up the computer as YOU want it. (This can be impressive; I don't have a 9640, but I have access to a friend's Tandy clone; I set up her autoexec batch file to set aside a 64K printer buffer within memory, a 64K ramdisk within memory, to copy certain non-resident DOS files into that ramdisk, read the no-slot clock, set the screen color to blue and text to white, and other functions. All this is done automatically right after DOS is booted. The 9640 has the same or similar capabilities!)

There was a lively question and answer session; Eric mentioned that the best place to get new programs for the 9640 or 99/4A is local BBS's. While the PATIUG library is always a good source of new programs, one can find programs posted on BBS's within days - and sometimes hours - of release. The three in the Philly area of interest to TI/Geneve users are Bob Rowe's TI-DI-BITS ((215) 672-4051), PACS ((215) 842-9603,9604,9605), and Bullwinkle's Corner ((215) 729/0401). Eric pointed out that when he gets new software from Myarc, he spends some time making sure it is working correctly, and then uploads it to TI-DI-BITS and the PACS BBS. Because of this, the LATEST software from Myarc for the 9640 will always be found on these BBS's immediately upon release!

Myarc Advanced BASIC (ABASIC) was also discussed in comparison to older BASICS used on the TI; while it is downward compatible (i.e., it will run TI console BASIC or XBASIC programs), it is much more advanced in its capabilities.

GOODBYE COMPUTER SHOPPER..

Computer Shopper had the distinction of being the last "newstand" magazine to cover the 99/4A and Geneve. It was also the only "newstand" source of information for owners of Sinclair and Adam computers. It seems that Computer Shopper has decided to lose this distinction...

Taking a glance at the latest issue of Computer Shopper, it seems that they have indeed eliminated the entire "Classic Computers" section. This issue covered the MS-DOS machines, the MacIntosh, the Amiga, and the Atari ST. The Apple II was covered by only one article (which discussed the IIgs, a 16-bit machine largely incompatible with the rest of the Apple II line, and - as always - the Laserwriter printer). Gone were all articles on TI, Commodore 64/128, 8-bit Atari's, Tandy Color Computer, Sinclair, and Adam computers; I guess the management of Computer Shopper no longer cares about selling their magazine to owners of these machines.

I used to buy Computer Shopper every month; after all, I have a TI-99/4A, a Franklin 1000 (an Apple II+ clone), a Tandy Color Computer 3, and, yes, even a Sinclair 1000. With one fell swoop my entire reason for buying this magazine has disappeared. Even if I had one of the machines still covered by Computer Shopper, there are better magazines on the stands that cover them. Also, I prefer buying things "in person" rather than through the mail (and there are plenty of computer shows in this area). So, I guess I - along with a lot of other folks - will be spending that monthly three bucks elsewhere...

(!)(!)(!)(!)(!)(!)(!)(!)(!)(!)(!)(!)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
BACK TO BASICS
TI Extended BASIC
Microsoft GWBasic
A Discussion by Art Byers
From QB-MONITOR
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Last year I wrote a four part series extolling the virtues of the Basic language and of TI Extended BASIC in particular for use on the 99/4A. Those pieces have been reprinted in many TI user group newsletters, so most 99'ers are well aware that Basic is one of my favorite computer languages.

It is only natural, therefore, that I would investigate Microsoft GW Basic which is most often supplied as a "freebie" when an IBM Compatible PC is purchased. In the small space of a computer club newsletter, I cannot go into much depth of detail. After all, the GW Basic manual I have is well over 400 pages, and the TI-XBasic manual with addendums is over 240 pages. My intent is to give you my own views and let you go further from there, if you so desire.

I have heard many conflicting stories about the origination of Basic for the TI. Here is what I believe is most likely true: Regular TI-Basic, or Console Basic as it is often called, was written by Microsoft for Texas Instruments. TI Extended Basic, which is obviously based

on TI-Basic, was written by a team of TI in-house programmers. It is no surprise, therefore, that GWBasic an TI-XBasic are very much alike. They are so similar that many of the more simple programs can run on both TI and PC systems with only minor syntax modifications.

Here are two examples: (1) XB uses a double colon "::" as a separator for multiple statements on the same line. GWB uses a single colon ":". (2) GWB uses a semi colon after text in an INPUT where XB uses a colon. ie: in GWB 100 INPUT "Enter your name";N$
in XB 100 INPUT "Enter your name":N$.

The overwhelming majority of statements, commands, and functions are either identical or there is a simple substitution of names such as SEG$ in TI-XB and MID$ in GWB. Both versions can link to assembly routines. Both can avail the programmer of useful peeks and pokes into varied memory locations. Both have the ease of program debugging generic to interpreted languages. All this is welcome news to a Basic devotee interested in transporting programs from one make of computer to another.

In fact, when reading the rest of this article, keep firmly in mind that MOST of XB and GWB statements, commands, and functions are the same right down to the last parenthesis and comma!!!

## WHICH IS MORE POWERFUL?

Although there are areas where I consider TI-XB to be definitely much superior to GWB, there can be no argument or question that overall GWB has more features, statements, commands, and available memory than TI-XB. (As an aside I'll say that the new Advanced Basic for the MYARC 9640 is better than and more powerful than GWB. However, most of us are using the 99/4A and must use XBasic.) Nevertheless, if you supplement TI XB with some assembly graphics routines Extended Basic not only can hold its own very well, but also in two very important respects outstrips GWB. Let's take a quick look.

## AMOUNT OF MEMORY AVAILABLE

First of all, sheer memory space available for Basic programming and required array storage is 60k in GWB as compared to about 37k in XB. Of the 48k memory in the 99/4A, Extended Basic uses either 768 bytes or 960 bytes of VDP RAM just for screen display (32 or 40 column mode). VDP RAM is also used for string variable storage, DSR buffers, line number tables, and color and screen tables, etc. Effectively, if you have opened a line to a printer and opened a couple of disk files (at 512 bytes each) you can safely figure at least 13k available for string storagee in VDP RAM. XB uses 24k high memory for programming as well as storage of the numerical arrays and constants. The lower 8k is there for assembly support linked to Basic programs but not for Basic programming. Even if we make the usual assumption that XB has the full 48k of RAM available one way or another for use by XB programming, GWB has 25% more. You may laugh that a 640k RAM PC has so much overhead that only 10 percent is available for Basic programming. That is not really the point. GWB simply has more memory available, period! (Actually the TI has more than 90k total memory with various ROM chips, but let's not get into that right now as no way does it have 640k of cpu addressable RAM.)

The above is much mitigated by the fact that the 99 was designed as a HOME computer. Very few programs written for home use should require 60k of memory. In six years of programming, I have NEVER run out of memory on the 99/4A for an XB program - though at times I had to do memory squeezing tricks and be very efficient with my code. If the amount of digging you have to do will be handled by a garden spade, is not a diesel powered back hoe overkill?

## WHERE GWB SHINES?

Many, many more features make it an easy and convenient programming language. Let me give you just a few examples:
(1) GWB simply has more goodies for the programmer such as statements and functions to use the PC's built-in clock including retrieval of time and date, elapsed time, as well as the ability to set the clock. Another useful statement is SWAP, which facilitates sorting data. There are many more examples I could list here.
(2) A GWB program can use SHELL commands to temporarily leave the Basic program and use a batch file, or use the many MSDOS commands such as reading directories, rename files or disks, etc., and return to the execution of the Basic program when the outside task has been completed.
(3) The 99/4A has excellent file handling. So does GWB but XB lacks GWB's ability to LOCK and UNLOCK files. Even woefully deficient Apple II Basic has that feature. Why Microsoft left it out of TI Basic, I do not know. Another omission from XB is the RESET statement to close all open files and update the disk directory. That is an important adjunct to any error trap routine or any program that requires swapping of disks. (4) A whole host of graphic commands such as CIRCLE, DRAW, PAINT, etc., are part of GWB. To use these in TI XB we must link to assembly routines such as Quality Soft's Draw and Plot or Mechatronics Expanded Graphics which contains many links to Basic. The latter uses so much overhead, however, that the memory remaining for XB programming is severely restricted. A fine alternative is the Super Extended Basic Module which automatically dumps graphic routines into lo-memory and has other enhancements.
(5) Another real plus for GWB is the ability tio use a few simple Basic statements and commands to save a whole screen to disk, to recall the screen, text or graphics

make no difference.

(6) There are many more features made possible because GWB is a larger version of Basic. These include additional looping commands such as WHILE/WEND. All I can advise is that you take a look at a manual in a bookstore if you want to investigate further.

(7) What may well be GWB's strongest point is that it is tied to a very powerful MS-DOS. It is this DOS connection that may be, for a majority of Basic programmers, the deciding factor in favor of GWB.

(8) GWB is substantially faster in execution than TI XB. I have written articles before on "Apparent Speed". I won't repeat them here more than to say I consider TI XB sufficiently fast for almost every home use except Telecommunications. A Telecom program can be written in GWB that will handle up to 2400 baud, (though probably best only up to 1200 baud as you must program in stop/start [Ctrl S and Ctrl Q] transmissions to the remote to allow GWB to catch up to itself at 2400 baud). Above that a compiled language program is required. XB could handle 110 baud or 300 baud.

## A LIMITATION OF GWBASIC

Much to my surprise, GWB limits you to a maximum of 255 dimensions in an array, [DIM A$(255)]. This applies to both string and numeric variables. To those of us in XBasic used to making DIMension statements much larger than this, either single [DIM A$(400)] or multi dimensional [DIM A$(100,4)] this is a potentially disabling choke collar on a very fast greyhound.

I am inclined to believe this is deliberate. If the TI with its limited RAM can dimension huge arrays, certainly Microsoft could have done the same with anywhere from 256k to 640k in the standard XT clone. Why?? Perhaps because they contemptuously don't think of the home user as writing sophisticated programs that would require large arrays. Perhaps they want to be sure that for sophisticated programs, users must go out and BUY software written in other languages, for it is a truism that the average user will program in Basic, IF they program at all.

There are ways of programming around this restriction but they sure make things unnecessarily complicated and difficult.

TI-XB has several areas where even after all these years, it still outstrips GWD.:

(1) The 99/4A, although a design of the 70's, was far ahead of its time in graphics and graphic definition. Even without the high number of pixels that the latest PC's have on their monitor screens, the TI can hold its own. What is more, the 99/4A's use of and ability to control Sprites is the equal or superior, even today, of many new computers. The normal IBM clone does not yet have sprites!!! What is more, all this graphic and sprite capability (such as the MAGNIFY command) is part of XB. GWB does not have sprites.

(2) XB statements and commands for screen display and input are superior on the TI. I consider TI's DISPLAY AT(row,column)SIZE() very much better than GWB's LOCATE(row,column,cursor,raster,raster). I think the ACCEPT AT(row,column)SIZE()VALIDATE() to be more logical and easier to u than GWB's methods. I even like the TI's CALL KEY much better than GWB's INKEY$. It gives more information and is more powerful.

(3) Here is the number one unequalled way that TI XB has it all over GWB: The ability to write truly structured programming by writing your own subprograms that you can CALL by the name you give them. For example: CALL INTEERESTCALC(PRINAMOUNT,RATE,PERIOD) is tons more meaningful than GOSUB 1300. This alone weighs so very heavily in favor of XB, that for many 99/4A (and 9640) users it may well be the deciding factor in making their judgement of which version of Basic is better.

So, I suggest, again, that some time when you are near one of the very large bookstores that have an extensive computer section, you peruse any GWBasic manual. I suspect you may just come away feeling much better about TI's Extended Basic.

## TIPS FROM THE TIGERCUB

#55

Tigercub Software
156 Collingwood Ave.
Columbus OH 43213

I am still offering over 120 original and unique entertainment, educational and utility programs at just $1.00 each, or on collection disks at $5.00 per disk.

The contents of the first 52 issues of this newsletter are available as ready-to-run programs on 5 Tips Disks at $10 each.

And my three Nuts & Bolts Disk, $15 each, each contain over 100 subprograms for you to merge into your own programs to do all kinds of wonderful things.

My catalog is available for $1, deductable from your first order (specify TIGERCUB catalog).

**********

## TI-PD LIBRARY

I have selected public domain programs, by category, to fill over 200 disks, as full as possible if I had enough programs of the category, with all the Basic-only programs converted to XBasic, with an E/A loader provided for assembly programs if possible, instructions added and any obvious bugs corrected, and with an autoloader by full program name on each disk. These are available as a copying service for just $1.50 postpaid in U.S. and Canada. No fairware will be offered without the author's permission. Send SASE for list or $1, refundable for 9-page catalog listing all titles and authors. Be sure to specify TI-PD catalog.

**********

The Tigercub has dipped a cautious paw into the cold dark mysterious waters of assembly, while still keeping a firm grip on trusty old Extended Basic. The result is an XBasic program that writes an assembly program!

The following subprogram, when merged into any program which has reidentified characters, and called after the characters have been reidentified, will write a source code which can be assembled into object code, loaded from XBasic and linked to instantly access the character set.

The source code is based on 2FONTS/S by Barry Traver, who gives credit to Mac McCormick, David Migicovsky and Karl Schuneman.

```
19000 SUB CHARSUB(HX$())
19001 DISPLAY AT(12,1)ERASE
ALL:"Source code filename?":
"DSK" :: ACCEPT AT(13,4)SIZE
(12)BEEP:F$ :: OPEN #1:"DSK"
&F$,OUTPUT
19002 DISPLAY AT(15,1):"LINK
ABLE program name?" :: ACCEP
T AT(16,1)SIZE(6):P$
19003 DISPLAY AT(18,1):"Rede
fine characters from  ASCI
I     to ASCII"
19004 ACCEPT AT(19,7)VALIDAT
E(DIGIT)SIZE(3):F
19005 ACCEPT AT(19,21)VALIDA
TE(DIGIT)SIZE(3):T
19006 PRINT #1:TAB(8);"DEF";
TAB(13);P$ :: PRINT #1:"VMBW
   EQU  >2024" :: PRINT #1:"
STATUS EQU  >837C"
19007 NB=(T-F+1)*8 :: CALL D
EC_HEX(NB,H$):: A=768+F*8 ::
 CALL DEC_HEX(A,A$)
19008 FOR CH=F TO T :: IF CH
<144 THEN CALL CHARPAT(CH,CH
$)ELSE CH$=HX$(CH)
19009 IF FLAG=0 THEN PRINT #
1:"FONT";:: FLAG=1
19010 FOR J=1 TO 13 STEP 4 :
: M$=M$&")"&SEG$(CH$,J,4)&",
" :: NEXT J :: M$=SEG$(M$,1,
23)&"  "&CHR$(CH)
19011 PRINT #1:TAB(8);"DATA
"&M$ :: M$="" :: NEXT CH
19012 PRINT #1:P$;TAB(8);"LI
   R1,FONT" :: PRINT #1:TAB(
8);"LI   R0,>"&A$ :: PRINT #
1:TAB(8);"LI   R2,>"&H$
19013 PRINT #1:TAB(8);"BLWP
@VMBW":TAB(8);"CLR  @STATUS"
:TAB(8);"RT":TAB(8);"END" ::
 CLOSE #1
19014 SUBEND
19015 SUB DEC_HEX(D,H$)
19016 X$="0123456789ABCDEF"
:: A=D+65536*(D>32767)
19017 H$=SEG$(X$,(INT(A/4096
)AND 15)+1,1)&SEG$(X$,(INT(A
/256)AND 15)+1,1)&SEG$(X$,(I
NT(A/16)AND 15)+1,1)&SEG$(X$
,(A AND 15)+1,1):: SUBEND
```

Now to try it out. You probably know that CALL CHARSET will restore reidentified characters below ASCII 96 to normal form, but not those above, so let's write a routine to restore those. Clear the memory with NEW, merge in the above, which you should have SAVED with -

SAVE DSK1.CHARSUB,MERGE

by MERGE DSK1.CHARSUB. Add a line -

100 CALL CHARSUB(HX$()) and RUN. Answer the filename prompt with DSK1.OLDLOW/S, the next prompt with OLDLOW and select ASCII 97 to 127.

When done, insert the Editor/Assembler module and its disk Part A. Select Assembler, Y to load assembler, give the source code DSK1.OLDLOW/S, object code DSK1.OLDLOW/O, just press Enter at next prompt, and R for options. You should get 0000 ERRORS.

Now key in this routine to test your program.

```
100 CALL INIT :: CALL LOAD("
DSK1.OLDLOW/O"):: FOR CH=33
TO 126 :: CALL CHAR(CH,"FF81
8181818181FF"):: PRINT CHR$(
CH);:: NEXT CH
101 CALL KEY(0,K,S):: IF S=0
 THEN 101 ELSE CALL CHARSET
102 CALL KEY(0,K,S):: IF S=0
 THEN 102 ELSE CALL LINK("OL
DLOW")
110 GOTO 110
```

Press any key to restore the upper case characters by CALL CHARSET, any key again to use the CALL LINK.

You are now ready to use the routine to copy all kinds of character sets from the programs in your library. You don't have any such programs? Not to worry. You don't have to reidentify characters one by one with one of those graphics editor programs. You can just manipulate the existing hex codes of the normal characters. I have created nearly 50 different character sets by that method!

The space occupied by a character on the screen is really an 8x8 square of 64 tiny dots. Various dots are turned on (colored) and off (transparent) to create a pattern - just the opposite of light bulbs on a scoreboard.

And those on-and-off dots are really the binary numbers which the computer uses. But fortunately the computer lets us use hexadecimal numbers rather than binary. The following will print out a reference chart of decimal to binary to hexadecimal. You can easily convert it to dump to a printer.

```
10 DISPLAY AT(6,1)ERASE ALL:
"DEC BIN HEX"
100 FOR J=0 TO 15 :: CALL DE
```

```
C_BIN(J,B$):: CALL DEC_HEX(J
,H$):: DISPLAY AT(J+8,1):J;T
AB(5);B$;TAB(10);SEG$(H$,4,1
):: NEXT J
21020 SUB DEC_BIN(D@,B$):: D
=D@ :: IF D=0 THEN B$="0000"
 :: SUBEXIT
21021 IF D=1 THEN 21022 :: X
=D/2 :: B@$=STR$(ABS(X<>INT(
X)))&B@$ :: D=INT(X):: IF D>
1 THEN 21021
21022 B@$="1"&B@$ :: B$=RPT$
("0",4-LEN(B@$))&B@$ :: B@$=
"" :: SUBEND
21039 SUB DEC_HEX(D,H$)
21040 X$="0123456789ABCDEF"
:: A=D+65536*(D>32767)
21041 H$=SEG$(X$,(INT(A/4096
)AND 15)+1,1)&SEG$(X$,(INT(A
/256)AND 15)+1,1)&SEG$(X$,(I
NT(A/16)AND 15)+1,1)&SEG$(X$
,(A AND 15)+1,1):: SUBEND
```

And this routine will show you how each letter is formed, by binary 0's (off) and 1's (on), for each key you press. I put it in merge format so you can MERGE it into any program and CALL it to examine the characters.

```
17000 SUB CHARVIEW
17001 !programmed by Jim Pet
erson Feb 1989
17002 DISPLAY AT(1,1)ERASE A
LL:"CHARACTERS IN BINARY & H
EX";;"Press any key to see
the  binary representation
 of thescreen character and
its hexcode."
17003 DISPLAY AT(8,1):"Press
 Enter to see the char-acter
."
17004 CALL KEY(0,K,S):: IF K
=13 THEN 17005 ELSE IF S=0 O
R K<32 OR K>143 THEN 17004 E
LSE 17007
17005 CALL CHAR(48,"FF"&RPT$
("81",6)&RPT$("FF",9))
17006 CALL KEY(0,K,S):: IF S
<1 THEN 17006 ELSE CALL CHAR
(48,"003844444444443800103010
01010103B"):: GOTO 17004
17007 CALL CHARPAT(K,CH$)
17008 R=12 :: FOR J=1 TO 15
STEP 2
17009 H$=SEG$(CH$,J,1):: CAL
L HEX_BIN(H$,B$)
17010 DISPLAY AT(R,8):B$
17011 H$=SEG$(CH$,J+1,1):: C
ALL HEX_BIN(H$,B$)
17012 DISPLAY AT(R,12):B$ ::
 DISPLAY AT(R,18):SEG$(CH$,J
,2):: R=R+1 :: NEXT J :: DIS
PLAY AT(22,6):CH$ :: GOTO 17
009
17013 SUBEND
17014 SUB HEX_BIN(H$,B$):: H
X$="0123456789ABCDEF" :: BN$
="0000X0001X0010X0011X0100X0
101X0110X0111X1000X1001X1010
X1011X1100X1101X1110X1111"
17015 FOR J=LEN(H$)TO 1 STEP
-1 :: X$=SEG$(H$,J,1)
17016 X=POS(HX$,X$,1)-1 :: T
$=SEG$(BN$,X*5+1,4)&T$ :: NE
XT J :: B$=T$ :: T$="" :: SU
BEND
```

And to reidentify a character, you just change the numbers and letters in the 16-digit hex code which represents the binary pattern. By writing little routines to switch those digits around, all kinds of things can be done.

For instance, the normal characters always have the top row of dots turned off, to provide spacing between lines of text on the screen. If you want taller characters you will have to double-space the lines, but you can create them by making the numerals and upper case characters consist of the 2nd-7th rows, the 7th row again, and the 8th row - it just happens to work out.

```
18000 SUB HIGHCHAR :: FOR CH
=48 TO 90 :: CALL CHARPAT(CH
,CH$):: CALL CHAR(CH,SEG$(CH
$,3,10)&RPT$(SEG$(CH$,13,2),
2)&SEG$(CH$,15,2)):: NEXT CH
 :: SUBEND
```

I made that a subprogram so you can MERGE it in and use it to modify other character sets.

If we take the hex code apart, 2 digits at a time, and reassemble it backward,

```
100 CALL CLEAR :: FOR CH=33
TO 90 :: CALL CHARPAT(CH,CH$
):: FOR J=1 TO 15 STEP 2 ::
CH2$=SEG$(CH$,J,2)&CH2$ :: N
EXT J :: CALL CHAR(CH,CH2$):
: CH2$="" :: NEXT CH
110 DISPLAY AT(12,1):"?NWOD
EDISPU";"YT EHT DENRUT OHW !
YEH" :: GOTO 110
```

That one was in my first Tips newsletter, years ago, but it is much more effective at assembly speed.

This one shades characters on their left edge by turning on the pixel to the left of the leftmost "on" pixel, if any. Also try it in combination with HIGHCHAR.

```
18001 SUB NEWCHAR3 :: FOR CH
=48 TO 122 :: CALL CHARPAT(C
H,CH$):: FOR J=1 TO 15 STEP
2
18002 CH2$=CH2$&SEG$("0367CD
EF",POS("01234567",SEG$(CH$,
J,1),1),1)&SEG$(CH$,J+1,1)::
 NEXT J :: CALL CHAR(CH,CH2$
):: CH2$="" :: NEXT CH :: SU
BEND
```

This one uses HIGHCHAR to heighten the character and then blanks out three rows. Try following it with NEWCHAR3.

```
18030 SUB NEWCHAR10 :: A$="0
0" :: FOR CH=48 TO 90 :: CAL
L CHARPAT(CH,CH$):: CH$=SEG$
(CH$,3,10)&RPT$(SEG$(CH$,13,
2),2)&SEG$(CH$,15,2)
18031 CH$=SEG$(CH$,1,4)&A$&S
EG$(CH$,7,2)&A$&SEG$(CH$,11,
2)&A$&SEG$(CH$,15,2):: CALL
CHAR(CH,CH$):: NEXT CH :: SU
BEND
```

The next one, which works only on ASCII 97-122, makes tall characters ridiculously elongated above.

```
18050 SUB NEWCHAR20 :: FOR C
H=97 TO 122 :: CALL CHARPAT(
CH,CH$):: CALL CHAR(CH,SEG$(
CH$,7,2)&RPT$(SEG$(CH$,9,2),
4)&SEG$(CH$,11,6)):: NEXT CH
 :: SUBEND
```

This one has the chaacters raised by one line, widened one column at left and two columns at right to make a full 8x8 character which must be double-spaced horizontally and vertically.

```
18090 SUB NEWCHAR27 :: FOR C
H=48 TO 122 :: CALL CHARPAT(
CH,CH$):: CH$=SEG$(CH$,3,10)
&RPT$(SEG$(CH$,13,2),2)&SEG$
(CH$,15,2):: FOR J=1 TO 15 S
TEP 2
18091 CH2$=CH2$&SEG$("014589
CD",POS("01234567",SEG$(CH$,
J,1),1),1)&SEG$("0129",POS("
048C",SEG$(CH$,J+1,1),1),1)
18092 NEXT J :: CALL CHAR(CH
,CH2$):: CH2$="" :: NEXT CH
 :: SUBEND
```

Those who have my Nuts & Bolts disks will see how valuable this assembly can be to make instantly available the routines for double height and double width characters, etc., etc. And if you have Todd Kaplan's amazing ALSAVE routine from the Genial Traveler Vol. 1 No. 3, you can imbed them in your XBasic program for fast loading.

And you can merge CHARSUB into any character editor or sprite defining program and, with a bit of modification, use it to convert your creations into fast-loading assembly.

These assembly loads are compatible with my BXB, so you can also load character

sets into sets 15 and 16, ASCII 144-159. However, the CHARPAT statement cannot access ASCII above 143, so in this case you must dimension an array in the program you are copying from, as DIM HX$(159), and place the hex codes in the array using the ASCII as the subscript number, such as CALL CHAR(CH+64,CH$) :: HX$(CH+64)=CH$, so that they will be passed to the subprogram. And don't CALL INIT after you have called BXB!

So, now you try creating your own screen fonts!

Memory full,

Jim Peterson

!!!!!!!!!!!!!!!!

PROGRAMMING MUSIC ON THE TI

by Jim Peterson

You don't need to know very much about programming in order to program music on the TI-99/4A - but it is almost essential that you know how to read sheet music. You don't really have to be able to play any instrument, or to know anything more about music, but it does help - the sheet music will probably have several notes for each chord, and a knowledge of music will help you to decide which three of those to use.

Music is programmed using the CALL SOUND subprogram. Each CALL must have a duration and at least one frequency with a volume.

The duration is measured in milli- seconds (thousandths of a second) and can be from 1 to 4250. The User's Reference Guide admits that the actual duration obtained may be off by as much as 1/60 of a second. Actually, any duration between 1 and 50 will be of the same length, and thereafter the actual duration increases in steps of about 8. This really causes no problems, because it is best to assign the desired duration of the shortest note to a variable, and then multiply that variable for longer durations. This also makes it very easy to change the tempo of the music.

When a CALL SOUND is executed, the frequencies will continue to play for the specified duration while the computer continues to execute the next statement. Thus, animated graphics can be combined with music, providing that the graphics commands can be executed before the end of the duration.

However, if the duration is given a negative value, between -1 and -4250, the CALL SOUND will be stopped and replaced by the next CALL SOUND. Thus, a CALL SOUND with a negative duration, placed within a loop, will sound continuously until the loop ends.

The frequency is given in Hertz, which is one cycle per second, and can be from 110, which is A below middle C, to a high of 44733 which is far above the range of hearing of most humans. However, the Reference Guide admits that the actual frequency may vary by as much as ten per cent. In fact, the deviation can be much greater than that, because any frequency from 31953 to 43733 ends up as exactly 37287! However, the frequencies within the audible range of music are close enough.

Each frequency must be given a volume, which can be from 0 (loudest) to 30 (inaudible). Actually, there are only 15 volume levels, because the odd numbers have the same volume as the preceding even number.

The CALL SOUND may optionally contain one to three frequencies, each with its own volume, and may also optionally contain a "noise", also with a volume. The "noise" may be a negative number from -1 to -8; of these, -1 to -4 are called "periodic noise" and -5 to -8 are called "white noise". The User's Reference Guide mentions that the -4 and -8 noises vary with the frequency of the third tone specified - that is the only clue we were given that bass notes are possible.

To create a bass note, the CALL SOUND must contain three frequencies and a -4 noise. The first two frequencies may be musical notes, or frequencies above audible range if only the bass is wanted, and the volume may be audible or inaudible. The third frequency will be sounded in the -4 noise, and will be two octaves and a 7th below its actual frequency - therefore the 3rd frequency is given an inaudible volume of 30 unless the actual frequency can be used in a 7th chord.

The easiest way to program bass notes is to multiply the frequency by 3.75 in the 3rd voice. This will lower it by two octaves.

The formula to create a musical scale of 4 octaves, starting with frequency F, is DIM N(48):: F=110 :: FOR J=1 TO 48 :: N(J)=INT(F*1.05946309^(J-1)) :: NEXT J :: N(1)=F

Once this has been executed, music can be programmed using the array subscripts, such as CALL SOUND(100,N(1),0,N(5),0)

However, most music programmers prefer to assign the frequency values to mnemonic variables, such as A1=110 for A of 1st octave, C2S=277 for C sharp of 2nd octave, etc. Music is then programmed by CALL SOUND (100,A2S,0) etc.

An even better method, perhaps, is to set up the array, as above, and then assign its values to mnemonic variables, such as A1=N(1). This makes it possible to change the key of a composition by simply changing the value of F.

Most musical compositions consist at least partly of repetitive phrases. The first thing to do when programming from sheet music is to go through the score and mark each series, of more than a few notes, which is repeated at least once within the score. Then, program each of these phrases as a GOSUB.

The simplest method of

music programming consists of nothing but a long list of CALL SOUNDs. This permits very fast music to be played, but it is tedious to program and takes up much memory.

Another method sometimes used is to place all the notes in DATA statements and then use a loop to READ them and play them in a single CALL SOUND, such as FOR J=1 TO 800 :: READ A,B,C,D:: CALL SOUND(A,B,0,C,0,D,0):: NEXT J

This method has limitations of speed, because DATA is read rather slowly, and is difficult to debug.

Some programmers get around the speed limitation by reading all the notes into an array, either from DATA statements or from a separate disk file, and then play them in a loop - FOR J=1 TO 1000 STEP 4 :: CALL SOUND(A(J),A(J+1),0,A(J+2),0,A(J+3),0):: NEXT J

This method can take as much as three minutes to initialize before the music begins, and is very difficult to debug.

The best method of all, first used by Sam Moore, Jr., and refined by Bill Knecht and others, uses the mnemonic variables, and GOSUBs to a single CALL SOUND (or to one of 4 CALL SOUNDS, to play 1 to 4 notes), with the variables in the CALL SOUND being redefined from the mnemonic variables before each GOSUB. Since the GOSUB is executed very rapidly, there is no limitation on speed, and the variables need be specified only when they change. For instance, the GOSUB might be:

1000 CALL SOUND(D*T,N1,V1,N2,V2,N2,V3):: RETURN, with D having been pre defined as the basic shortest duration. Then, a program line might read:

100 T=1 :: V1,V2,V3=0 :: N1=C3 :: N2=G2 :: N3=E1 :: GOSUB 1000 :: N2=E2 :: N3=G1 :: GOSUB 1000 :: N2=C2 :: N3=E1 :: GOSUB 1000 ...etc. to play a contin uous melody note with a pattern of accompaniment notes.

As a variation of the above, the GOSUB may be to a loop which plays the chord with a negative duration and a decreasing volume, as

FOR V=0 to 30 STEP 5 :: CALL SOUND(-99, N1,V,N2,V,N3,V):: NEXT V :: RETURN

This gives the "decay", or fading volume, which is typical of piano music or other music where a string is struck or plucked. The slowness of loop execution places some limitations on the use of this technique in Extended Basic programming, but it has great potential in assembly, FORTH, or compiled Basic.

Many other effects are possible. Alternating two CALL SOUNDs in a loop, with the frequency in the second being a multiple of 1.01 to 1.03 times the first, gives a tremolo effect - FOR J =1 TO T :: CALL SOUND(-99,N1,V1):: CALL SOUND(-99,N1*1.03,V1):: NEXT J

Alternating the volume rather than the frequency, or both, can give other tremolo effects.

Two notes in one CALL SOUND, with the second being multiplied by about 1.03 over the first, gives a richer sound CALL SOUND(100,N1,V1,N1*1.03,V1)

Extending the -4 note into the melody range gives a

**********

PROGRAMS THAT WRITE PROGRAMS
Part 5
By Jim Peterson

In addition to writing programs in MERGE format, the same techniques can be used to analyze or modify programs which have been SAVEd in MERGE format. The D/V 163 file editor in Part 2 of this series was an example.

Here is a simple program to remove REM statements -

```
100 DISPLAY AT(3,5)ERASE ALL
:"REM REMOVER": : :"Program
must be SAVEd in":"MERGE for
mat by":"SAVE DSK(filename),
MERGE"
110 DISPLAY AT(12,1):"FILENA
ME? DSK" :: ACCEPT AT(12,14)
:F$ :: DISPLAY AT(14,1):"NEW
 FILENAME? DSK" :: ACCEPT AT
(14,18):NF$
120 OPEN #1:"DSK"&F$,VARIABL
E 163,INPUT :: OPEN #2:"DSK"
NF$,VARIABLE 163,OUTPUT
130 LINPUT #1:M$ :: A=POS(M$
,CHR$(131),1):: B=POS(M$,CHR
$(154),1):: A=MAX(A,B):: IF
A=3 THEN 150 :: IF A=0 THEN
PRINT #2:M$ :: GOTO 150
140 PRINT #2:SEG$(M$,1,A-1)&
CHR$(0)
150 IF EOF(1)<>1 THEN 130 ::
 CLOSE #1 :: PRINT #2:CHR$(2
55)&CHR$(255):: CLOSE #2
```

The REM statement will begin with either a !, which is CHR$(131), or REM which is CHR$(154). So, line 130 reads in the lines one at a time. A finds the position in the line of ! and B finds the position of REM; one or the other, or both, will not be present and will equal 0. Then MAX finds the larger of A and B, which will be whichever one is present, or 0 if neither.

If ! or REM is in the 3rd position, immediately after the 2-byte line number, we want to delete the line entirely, so we do not reprint it. If A=0 then neither ! nor REM is present, so we reprint the entire line in the new file.

Otherwise, the REM statement is obviously a tail remark, so we reprint to the new file the segment of it starting with the first character and consisting of the number of characters one less than the position of the ! or REM. And, since we have lopped off the end of the line, we do not forget to replace the end-of-line marker CHR$(0).

If we have not reached the end of the file, we go back for the next line. Otherwise, we close the old file, but we remember to add the end-of-file marker to the new file before we close that too.

**********