Winnipeg

May 86


# Newsletter

## May's Newsletter

The Winnipeg 99/4 User Group is a non-profit organization formed by computer hobbyists for users of the Texas Instruments 99/4A Home Computer and compatables. The content of this publication does not necessarily represent the view of the Winnipeg 99/4 User Group.

Next General Meeting - Date : June 5th, 1986
Time : 7:00 P.M.
Place: Winnipeg Centennial Library
2nd Floor, Assembly Room

Executive 1986:

| | | |
|---|---|---|
| President: | Jim Barnard | 334-5987 |
| Treasurer: | Bill Quinn | 837-7758 |
| Newsletter Editor, Book and User Programs Librarian: | Mike Swiridenko | 772-8565 |
| Contributing Editor: | Paul Degner | 586-6889 |
| Inter-Group Representative and Newsletter Publisher: | Dave Wood | 895-7067 |
| Asst. Newsletter Publisher: | Hank Derkson | |
| Systems Co-Ordinator: | Rick Lumsden | 253-0794 |
| Educational Co-Ordinator: | Sheldon Itscovich | 633-0835 |
| Public Domain Librarian: 822 Henderson Hwy. | Gordon Richards | 668-4804 |
| Module Librarian: | Peter Gould | 889-5505 |

Mailing Address:

NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA, R3C 2Z6

## EDITORIAL COMMENTS:

Hello again! This month features a review of a program I obtained several weeks back, during a visit with Mark Gibson of Grand Forks. A hearty hello to all of the members of MAD HUG! I know how much you like to hear from us. Also in this newsletter is a continuation of my article on software design called Testing and Debugging Software. Programmers Helpfile continues the debugging theme with a brief explanation of how to use the assembly language DEBUG program, while in the XBasic section there is a discussion about useing BREAK, TRACE, and ON ERROR statements. Forth enthusiasts can enjoy my attempt at a hi-resolution graphics dump program. Hints and Tips has Basic to C conversions, and a X/Basic routine to shuffle cards. Thats all for now. Until next month, Happy Programming!

If you have a review, user hints, or helpful programming tips, get them to me for the next newsletter. The deadline that I have set for submissions is one week before the date of the group's meeting. Thanks go to all who have submitted items for this issue of our newsletter.

## MISCELLANIA:

Miscellaneous news and reminders.

One more month till we break for summer.

Paul continues to bring with him a regular monthly disk of programs. The Clubline 99s I expected several weeks ago have not arrived as of the typing of this. I don't know why they are being delayed.

The IBM Qume half-height drives ordered through Paul Degner, after a bit of a scrounge for a working power supply able to supply enough current for the thirsty things, work just fine.

Ron Bryson, an enterprising gentleman, comes around to our monthly meetings with his cartload of computer supplies. He offers competitive prices on a variety of floppy diskettes, diskette holders and other items. We hope to get a price list from Ron for the next newsletter. For more information on prices Ron will be available during our meeting, or you can call him at 269-0675.

## READER RESPONSE:

From the May 1986 Compute! comes the following notice:

TI TIPS BOOK

In an effort to provide easily accessed documentation to TI users, I have put together a TI tips booklet that consists of 99 tips for the TI-99/4A. These are a compilation of suggestions given in our user group newsletter. They include PEEKs, POKEs, listings, hints, and so on. Also included is a complete disk drive memory map, summary of Extended BASIC commands, and a sorting program written in BASIC and machine language. One such tip that may interest your readers allows them to disable FCTN = (QUIT) in Extended BASIC. To do this enter this statement
CALL INIT :: CALL LOAD(-31806,16)

To enable it again, type CALL LOAD(-31806,0). Another POKE allows you to prevent Extended BASIC programs from being listed. Type CALL LOAD(-31931,128) to do this. To unprotect Extended BASIC programs, enter CALL LOAD(-31931,0). The TI tips booklet is available through the Central Iowa 99/4A Users Group for $4 (the cost of materials, printing, and postage) at the following address:

Central Iowa 99/4A Users Group
Box 3043
Des Moines, IA 50316

John Hamilton

## REVIEWS:

This column presents reviews of materials that may be of interest to the user. The views expressed are the opinions of the reviewers, exclusively.

**SOFTWARE:**

XB-Detective
Reviewed by: M. Swiridenko

XB-Detective is a utility for XBasic prrogram development, written by Art Gibson and Bill Crowell. Since this program consists of a machine code object file the 32K Mem-exp and a disk drive is required. A printer is recommended, also.

The XB-Detective package comes in a brown envelope with a brief description of the features of the enclosed software. Also on the cover of the envelope is an intriguing dot matrix printer picture of Sherlock Holmes examining a floppy disk through a magnifying glass. Inside the envelope one finds a single page of instructions and a single floppy diskette.

Loading and using the software is extremely easy and explain the lack of volumes of instructions. The program will auto-load if you start from the main screen and then select XBasic, or can be loaded from XBasic with a single CALL LOAD. When the program loads an impressive title screen (in my opinion) is displayed and a cartoon floppy disk, on wheels, scoots across the screen. This is an entertaining distraction.

XB-Detective is different from most XBasic utility programs since it loads into the low-memory area of the memory expansion, and will not interfere with the functioning of an Extended Basic program. Other program utilities I have seen must work with merge, or text files and cannot be loaded along with your XBasic program.

The XB-Detective utility can be called up at any time with a simple 'FCTN 7' key press. Once you do this you are presented with a menu of options to choose from. The Menu is easy enough to follow and almost makes using the XB-Detective self-explanatory. To really understand what each of the five options can do you have to try them a few times. Returning to your XBasic program is just as easy and simply a matter of pressing Option 6 of the main menu.

The five utility options that you may select are:

1) LIST VARIABLES
2) FIND VARIABLES
3) FIND RESERVED WORDS

4) DELETE LINES
5) STRING SEARCH

Option 1 will list for you all of the variables that are in your XBasic program. Option 2 will find and list all line numbers of lines containing a specific variable. Option 3 will find and list all lines in which a specified reserved word occurs. Reserved words are keywords such a FOR, GOSUB, REM, and so on, that are used in the Extended Basic language. Option 4 will delete lines within your program. (A lockup will occur if you enter XB-Detective while editting a line then delete the line you were on.) Option 5 will find any specified occurance of a string in your program. The string may be within quotes, in a data statement, or part of a CALL statement. With each of the options you have the choice of printing the information found or continuing.

Options 1, 2, and 3 are often combined, in other utility programs, and are seen as a single Cross-reference listing. I suppose the reason this wasn't done with this utility is so that the user may list selected information to the screen, as needed. Seperating these features in this way helps to make XB-Detective very versatile. A complete Cross-reference feature would; however, be nice since it is sometimes easier to mull over a single listing with all the required information than it is to scan several pages of selected printout.

Option 4, delete lines, is an interesting feature but seems a bit out of place. What I would have liked to have seen was an option to MOVE, COPY and DELETE lines. This feature would be very welcome since these editing functions are rather awkward to do with the XBasic editor, and editing keys.

Option 5, STRING SEARCH, is usefull, and provides a function that is a combination of an editor and a Cross-reference function. Like an editor you may search for an occurance of some character string, like a cross-reference utility it will tell you the lines in which the string was found.

One feature I found lacking was an option to list all unique CALL strings. This option would show at a glance the 'NAMES' of all subroutines referenced in CALL statements. As it is you must search for a specific CALL name, or list the numbers of all lines containing SUB statements.

All in all XB-Detective performs well, and is easy to use. XB-Detective will be quite usefull to XBasic programmers needing a 'clean-and-fast' interactive Cross-reference utility.

## HELPFUL HINTS AND TIPS!
## (FOR THE USERS, BY THE USERS!)

This column features tips brought to my attention from members of this group, other user group's newsletters, and various other sources. **WARNING:** These hints and tips are to be used at your own risk!

**C99:**

The following BASIC to C conversions were gleaned from the May 1986 issue of Compute!

From a beginner's viewpoint, one of the reassuring aspects of the C language is that it has many things in common with BASIC. You can write a large part of your C program using statements that are just like or very similar to Basic statements. Of course, they have no line numbers, are written with lowercase letters, and end with a semi-colon. But they use the same keywords as BASIC statements, and perform the same or nearly the same operations. These are the C language statements that are equivalent to Basic statements:

1) Assignment statement
2) IF statement
3) FOR loop
4) GOTO statement

An assignment statement in Basic looks like this: 100 ITEM=4875.
The same statement in C would look like this: item=4875;.
Basic does not care wether the constant assigned to a numeric variable is an integer value or a decimal value, C does. The way C is told what kind of values a particular variable may accept is by a declarator statement.
For example the declarator statement for 'item' would look like this: float item;.
Float means that 'item' may be assigned decimal values. A variable may also be specified as being of only 'integer' type. When an integer value is assigned to a 'float' or floating point variable it is converted and saved as a floating point value. A float value assigned to an 'integer' variable will be converted to an integer value. Constants are also recognized as being of type float or of type integer. Using an integer constant were a float constant is required may give incorrect results since integer operations may be performed. The declarator statement also tells the C compiler to reserve a specific amount of memory for the designated varaible. Other examples of assignment statements are:

```
      BASIC VERSION                          C VERSION
200  AVE = (A + B + C)/3                   ave = (a + b + c)/3.0;  (float ave,a,b,c;)
250  SIDE = SIN(X) * HYP                   side = sin(x) * hyp;
```

The C 'if' statement never needs a "then". It uses parenthises around the relational or logical expresion. You may also leave off the else clause if you need to.
Here are some examples:

```
100  IF YEAR<1984 THEN A=28 ELSE A=30     200 IF MONTH=2 THEN N=29
```

In C this would be:

```
      if (year<1986)                         if (month=2)
        a=28;                                  n=29;
      else
        a=30;
```

Unlike Basic you may not branch to a specific like number from within an 'if' statement. You may; however, have multiple statements or 'blocks' of statements. These blocks are enclosed within braces.
Example: 400 IF YEAR>1983 THEN RATE=RATE+.01::BASE=BASE-500::SURCHARGE=SURCHARGE+.02
In C this is:

```
      if (year>1983) {
        rate=rate+.01;
        base=base-500;
```

```
        surcharge=surcharge+.02;
      }
```

C's 'for' loop is much more versatile than BASIC's FOR-NEXT loop. The C version has no NEXT statement since the C loop ends with a ';' or a ')'. Statement blocks are also allowed in the 'for' statement. The ';' will end a single statement 'for' loop. The expressions within the parenthises immediately following the 'for' specify the starting value, terminating condition, and the stepping value of the loop.

An example:

```
600  SQ,ODD = 1                          sq = odd = 1
610  FOR R=1 TO 15                       for (r=1; r<=15; r=r+1 ) {
620  PRINT SQ,R                             printf("%d   %d\n",sq,r);
630  ODD=ODD+2                              odd=odd+2;
640  SQ=SQ+ODD                              sq=sq+odd;
650  NEXT R                                }
```

A statement that is different from XBasic, but allows you to perform loops is the 'while' statement. It looks like this:

```
while (n<=21) {                      100 IF N>21 THEN 200
    scanf("%d",n);                   110 INPUT N
    n=n+5;                           120 N=N+5
   }                                 130 GOTO 100
                                         200 REM loop jumps to here.
```

BASIC's GOTO statement allows you to write unmanagable programs, yet it is unavoidable in many BASIC programs. C's 'goto' statement is seldom required. You should avoid using the 'goto' statement to keep your programs understandable.

Here is an example:

```
400  GOTO 450
```

in C the equivalent is:

```
goto there;.
```

Line numbers are not possible in C but labels are. 'there' is an example of a label. You must label the statement you wish to go to in a 'goto' statement.

A labeled statement looks like this:

```
there:year=year+1;/* you may label any statement.*/
```

The remark between the '/*' and the '*/' is a comment and the '/*' and '*/' are comment delimiters. The comment delimiters correspond to a REM statement in BASIC.

For more information about the C language you may consult one of the many books available on the subject.

**EXTENDED BASIC**

This routine was discovered while writing a Cribbage game. Its function is to 'shuffle' a 'deck of 52 cards'. The deck of cards is represented by an array of the numbers 1-52.

There are many ways of mixing a collection of numbers. The principle of this program is to swap elements within the array until a desired mix is achieved. The swap is done by taking a randomly chosen element from the upper part of the card array and exchanging it with a selected element of the lower part of the array. Since the exchange of elements within the array has the effect of mixing two elements at the same time only half of the array need be mixed in order to mix the entire array. This results in rather quick mix of the cards. Here is the program:

```
100 DIM DECK(52)
110 REM  INITIALIZE THE CARD DECK.
120 FOR I=1 TO 52
130 DECK(I)=I
140 NEXT I
150 GOSUB 1000 ! SHUFFLE THE DECK.
160 REM REST OF PROGRAM.
170 END
1000 REM  SHUFFLE THE DECK.
1010 FOR I=1 TO 26
1020 RANDOMIZE
1030 CARD=I+INT(RND*(52-I)+1)
1040 REM  EXCHANGE SELECTED CARDS.
1050 TEMP=DECK(I)
1060 DECK(I)=DECK(CARD)
1070 DECK(CARD)=TEMP
1080 NEXT I
1090 RETURN
```

## CURIOSITIES AND PASTIMES

This column features a monthly BRAIN TWISTER for your intellectual entertainment.

The following is called "Monkey and the Pulley". The source of this puzzle states that "it is quite easy if you have a pretty clear head.". I hope it won't keep you up late into the nite, as this one is tough!

A rope is passed over a pulley. It has a weight at one end and a monkey at the other. There is the same length of rope on either side and equillibrium is maintained. The rope weighs four ounces per foot. The age of the monkey and the age of the monkey's mother together total four years. The weight of the monkey is as many pounds as the monkey's mother is years old. The monkey's mother is twice as old as the monkey was when the monkey's mother was half as old as the monkey will be when the monkey's mother was three times as old as the monkey. The weight of the rope and the weight at the end was half as much again as the difference in weight between the weight of the weight and the weight and the weight of the monkey.

Now, what was the length of the rope?

Here is the solution to last month's puzzle. Football Results:

```
Scotland vs England    3 -- 0
Scotland vs Whales     2 -- 1
Scotland vs Ireland    2 -- 0
Whales   vs Ireland    2 -- 1
```

## Article: DEBUGGING and TESTING SOFTWARE

Author: M. Swiridenko

To complete last months article about software design I will continue with a discusion of the procedure of testing and debugging of software. Testing is done to show that a program will operate correctly. Testing to find out why a program performs incorrectly and the subsequent correction of the error is the process commonly know as debugging. Debugging is a term that originated in the days of the vacuum tube computer. Many computer failures, in those days, were caused by insects bogging down the computer's internal relays. Thus came the phrase, 'You have a 'bug' in your computer?!'.

Like designing software debugging of software can be done in a Top-down or Bottom-up manner. In top-down testing high-level controlling programs are tested using dummy subprograms to isolate the source of errors. As errors in the higher-level subroutines are eliminated lower-level routines will be introduced and further testing will be done. In Bottom-up testing, testing begins by insuring that each of the many low-level subroutines of a program function correctly. The subroutines at the lowest levels will first be tested individually. As errors are eliminated higher-level controlling subroutines are included and testing continues. The approach most often taken is a combination of the two methods. Tested low-level routines are added as high-level routines are corrected.

Debugging of a program procedes from the simple task of eliminating syntax errors to the more complex task of ensuring that the logic of the program is correct. Syntax errors are the result of typing mistakes, or the use of incorrect program statements. Syntax errors are usually found out before a program will run. Logic errors, on th other hand, are the result of incorrect data operations or flow of program control, and occur during the execution your program. Incorrect assignment of values, wrong signs, and misuse of conditional operations will all result in logic errors.

Testing for logic errors involved the use of break points, memory and/or variable dumps, inserting dummy routines, and line number or execution tracing. All of these techniques are used to pinpoint the faulty logic within the program. Once the logic is pinpointed it can be corrected. Once the logic is corrected the methods used to locate the error must be deleted from the program, otherwise the program may be hindered in its execution.

Break points, print statements, or even audio/visual effects, when inserted into several points of a program, can be used to indicate a program's flow of control. For instance if you set indicators throughout your program and execute your program you may tell by the sequence of the appearance of the indicators exactly how your program is performing. An indicator which does not appear or appears too many times tells you that there is something wrong with your program. By knowing where you placed the particular indicator, and how the program was supossed to perform, you can get some idea of where the error is located.

Tracing can also be used to reveal the flow of control of your program. Tracing usually results in a messy display of line numbers. The indicators that I have already mentioned are often a more subtle way of doing the same thing. The amount of tracing output necessary can be reduced by using subtle indicators to locate the part of the program where the suspected error may be, then using a trace function to show local flow of execution.

Memory and/or variable printouts will further narrow down the possible location of the error. Before and after printouts, of variable and/or memory contents, at selected points will help to reveal what happens within your program. Changes in variables that shouldn't change or variables that don't change when they should will reveal problems with assignments, operations, and conditionals.

In conclusion:
Testing and Debugging both play an important part in the development of computer software. A good approach to these tasks will reduce the amount of frustration and time wasted spent trying to correct errors in a program.

## PROGRAMMING HELP FILE:

The purpose of this column is to present, to the user, techniques and information that will be useful in the writing of programs for the TI-99/4A home computer.

### BASIC/EX-BASIC:

Frequently when writing programs a programmer must make certain his work performs correctly. Extended Basic has several builtin statements that aid in the process of correcting programming errors, and verifying the correct operation of a program. The statemerts I will discuss here are BREAK, TRACE, ON ERROR, and there related statements, CONTINUE, UNBREAK, UNTRACE, CALL EPF, ON BREAK, and ON WARNING. For further information on these or other X/BASIC statements consult the appropriate manual.

To follow the operation of your program you will use the TRACE statement. TRACEing the execution of your program will allow you to see which lines are being interpreted at any particular moment. TRACE, when executed before your program or from within a program will turn on a feature which list the line number of a statement prior to its interpretation. This is a software feature of the Extended Basic interpreter. UNTRACE will simply turn off the TRACE feature. As you may expect you are able to trace the execution of specific segments of your program with carefull placement of the TRACE and UNTRACE statement. TRACEing is usefull when you don't really know when branching or GOSUBs are occuring.

There are times during program execution when you want to make sure the values of variables are correct before allowing your program to execute further. In this case you would use a BREAK statement. A BREAK statement will temporarily halt execution of your program from within. To set a break-point within your program place a BREAK statement at the desired location. When the BREAK statement is encountered your program be interrupted and will stop executing.

Once your program has halted, you are placed in immediate mode and may view variable contents by PRINTing them. All of the variables within your program are available to you with the exceptions of defined characters and sprites. These are cleared and the standard character set is loaded when a BREAK occurs. If you don't change your program, you may let it resume from the point it stopped by entering the CONTINUE statement. CON is the abbreviated form of CONTINUE.

A function key that has the same effect as BREAK statement but operates externally to your program is the FCTN 4 or CLEAR key. As with a BREAK statement you may list variables and resume with a CON statement.

Another method of settting BREAK points within your program is to specify line numbers, seperated by commas, in a BREAK statement. When the line(s) specified are encountered a BREAK will occur. UNBREAK will remove all specified BREAK points when, or specific BREAK points when line numbers are specified. ON BREAK is used to control external

BREAKing of your program. Using ON BREAK NEXT will cause the CLEAR key to be ignored. Using ON BREAK STOP will resume the receognition of the external BREAK key. ON WARNING allows you specify actions to be taken in situations were WARNING messages are given. ON WARNING NEXT will let the program ignore all warning situations. ON WARNING STOP will halt your program when a WARNING situation occurs. ON WARNING PRINT allows the standard WARNING messages to print.

ON ERROR is used to trap errors that may happen in a program. This statement is useful when the normal XBasic error messages don't tell you too much about what is going on. By specifying a line number you may branch to your own special error handling routine. The line number you specify must begin a series of statements ending with a RETURN statement. A usefull statement within your error handling routine is the CALL ERR statement. CALL ERR is a special subroutine that will return the error number and line number of the current error state of your program. To resume normal error handling you would use ON ERROR STOP. The next error encountered will then terminate execution of your program.

By selective placement of the statements discussed one can narrow down a search for programming errors, and 'Kill them Bugs DEAD!'. DE-bug! Boss! De-bug!

**ASSEMBLY:**

The following discussion is based upon portions of chapter 23 of the E/A manual, and chapter 10 of Ira McCormick's book 'Learning TI 99/4A Home Computer Assembly Language Programming'. For additional information consult those texts.

Debugging an assembly language program can be a long process if you do it by trial and error. A software debugger utility can speed up the process of finding program errors quite significantly. One ssuch debugger program is provided with the Editor/Assembler package. It is the DEBUG program of the SIDE A disk. This discussion will be a brief explanation of how to load and use this program.

The DEBUG program allows you to control the execution, and to examine/alter memory, and register contents of your application program. Execution is controlled by setting the contents of the Workspace, PC, and status registers and by the setting of program break-points. Break points are the assembly equivalent of X/Basic's BREAK statement. To use the Debugger you will need an assembly source listing of your program. A source listing will give you a reference to addresses were your program instructions and data area are located.

To load the DEBUG program use option 3 of the E/A module and load in your object program, then load in the DEBUG program by entering DSKn.DEBUG, where n is the drive number. Your program must not auto-start or you will never be able to load the Debugger. After both programs have been loaded press return and enter the program name 'DEBUG'. You wil get the Debugger message and a '.' prompt. All commands to the Debugger are single character and will be entered from the period prompt.

The commands that will get you started are the M, R, B, E, W, and Q commands. Other commands are '>', '.', 'X,Y,Z', AND H.

M followed by a hex address will display the contents of the word of memory at the given location. After the contents are displayed you may change the word by entering a hex value. Pressing the space bar will display the next word of memory. Pressing a carriage return will take you back to the period prompt.

R will display the contents of the WS, PC, and STATUS registers one at a time. As each register is displayed you may alter itss contents by entering a hex value. Pressing a space bar will display the next register. A carriage return will return you to the period prompt. By setting the contents of these CPU registers you can set the execution conditions for the exeution of your program, and the subsequent control by the debug program.

B followed by a hex address, allows you to set a breakpoint. B followed by a carriage return will list all break points that have been set. You may set from 11 to 16 breakpoints. Entering a '-' will clear all breakpoints. Entering a '-' after a hex address will clear a specific breakpoint. After you set the CPU registers and execute the application program any breakpoint encountered will return you to the Debug program. A B and the contents of the CPU registers will be displayed, eg.- B 2000 A010 3000, followed by the period prompt. From the prompt you may then inspect/alter registers, and memory locations. As breakpoints are encountered they are removed and thus must be reset if they are needed again i.e.- in a loop. Execution may be resumed from the breakpoint provided that the CPU registers aren't altered.

To execute your program from the Debugger you must set the contents of the CPU registers to valid values prior to entering a E command. Your program will start executing from the location in the PC register and will continue until a breakpoint, lockup, or end of the program occurs.

Entering a W at the period prompt followed by a carriage return will display the contents of all of the working registers. Entering a number before pressing the return key and the contents of the specified register are displayed for inspection. To change the contents of the register enter a hex value. Enter a space to get the contents of the next register, a '-' to get the contents of the previous register, or a carriage return to get the period prompt.

The final command that you will need to know is Q the quit command. Entering a Q command will quit the debugger program.

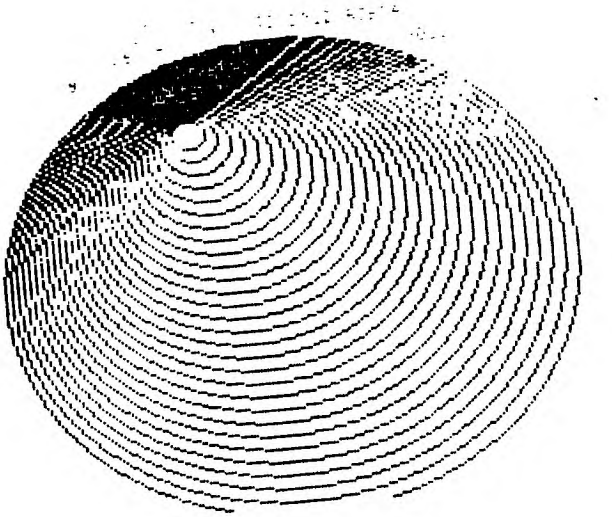
Other commands that are usefull are the '>', '.', and 'X,Y,Z'. '>' and '.' are convert to decimal and convert to hex, respectively. '>' followed by a hex value will return the decimal conversion. '.' followed by a decimal value will return the hex conversion. X, Y, Z, are bias variables. These variable can be set to a specific hex value and used with relative addresses. Ex.- X=A000 then 10X will be interpreted as A010 by the debugger.

This concludes a somewhat brief description of the use of the E/A DEBUG utility. On a final note there are other more ssophisticated debug utilities around but these programs occupy much more memory than TI's DEBUG. Debugging an assembly program need not be a trial and error process and can be made much easier with evan a small utility like DEBUG.

**FORTH:**

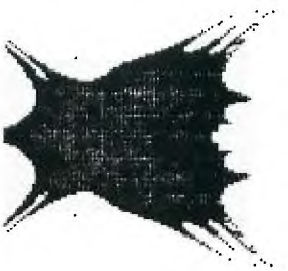
The following is a TI-Forth GRAPHICS2 mode screen dump program. To use this screen dump load -PRINT and the screen listed below, and insert the word HDUMP into your graphics word at the appropriate place. The speed of this dump is rather slow (two to five minutes per dump) and can be improved by writing it in Forth Assembly language. I will leave that as an exercise for the willing.

```
SCR #41
  0 ( HI-RES SCREEN DUMP) DECIMAL
  1 0 VARIABLE VA  0 VARIABLE X
  2 0 VARIABLE PZ 0 VARIABLE PW 0 VARIABLE CD 14 ALLOT
  3 : P-CODE 0 64 75 27 4 0 DO EMIT LOOP ;
  4 : CR-CODE 8 65 27 3 0 DO EMIT LOOP CR ;
  5 : ?BREAK ?TERMINAL IF TEXT UNDER ABORT THEN ;
  6 : CNVRT  VA ! 256 PZ ! 0 X ! 8 0 DO 0 CD I 2 * + ! LOOP
  7   8 0 DO VA @ I + VSBR X ! PZ @ 2 / PZ ! 256 PW !
  8   -1 7 DO PW @ 2 / PW ! X @ PW @ / 0 > IF
  9       X @ PW @ - X !  CD I 2 * + DUP @ PZ @ + SWAP ! THEN
 10   -1 +LOOP LOOP -1 7 DO CD I 2 * + @ EMITB -1 +LOOP ;
 11
 12 : HDUMP SWCH 24 0 DO 4 0 DO J 256 * I 64 * + P-CODE
 13   8 0 DO DUP I 8 * + 8192 + CNVRT ?BREAK
 14   LOOP DROP LOOP CR-CODE LOOP UNSWCH ;
 15
```



The accompanying graphics were printed using the above screen dump program. The ink blot picture is of a the Mandelbrot set between (1.5,-1.5) real axis, and (1.5,-1.5) complex axis, at a resolution of 100x100 points. Five iterations were made at each point. The other dump is of circles of increasing radius and moving center.

```
100 CALL CLEAR
105 REM P. 152 MARCH 1986 COMPUTE!
110 CALL SCREEN(15)
120 PRINT TAB(7);"SHAPING TI SOUNDS"
130 FOR T=1 TO 6
140 PRINT
150 NEXT T
160 PRINT "CHOOSE:"
170 PRINT
180 PRINT
190 PRINT TAB(4);"1) SHAPED MUSICAL NOTES"
200 PRINT
210 PRINT TAB(4);"2) ECHO"
220 PRINT
230 PRINT TAB(4);"3) QUIT"
240 PRINT
250 INPUT A
260 IF (A<1)+(A>3)THEN 250
270 ON A GOTO 290,520,690
280 REM THIS PART PRODUCES SHAPED MUSICAL NOTES
290 CALL CLEAR
300 CALL SCREEN(13)
310 PRINT TAB(3);"* SHAPED MUSICAL NOTES *"
320 FOR T=1 TO 10
330 PRINT
340 NEXT T
350 PRINT "ENTER RISE AND FALL TIMES -"
360 PRINT "USE VALUES GREATER THAN ZERO";
370 PRINT
380 INPUT R,D
390 IF (R<=0)+(D<=0)THEN 380
```



```
400 FOR F=110 TO 880 STEP 30
410 FOR DB=30 TO 0 STEP -5/R
420 CALL SOUND(-10,F,DB)
430 NEXT DB
440 FOR DB=0 TO 30 STEP 5/D
450 CALL SOUND(-10,F,DB)
460 NEXT DB
470 FOR T=1 TO 50
480 NEXT T
490 NEXT F
500 GOTO 100
510 REM THIS PART CREATES AN ECHO EFFECT
520 CALL CLEAR
530 CALL SCREEN(14)
540 PRINT TAB(8);"* ECHO EFFECT *"
550 FOR T=1 TO 12
560 PRINT
570 NEXT T
580 FOR F=110 TO 880 STEP 30
590 FOR DB=1 TO 30 STEP 1.75
600 CALL SOUND(-10,F,DB)
610 FOR T=1 TO 10
620 NEXT T
630 CALL SOUND(-10,990-F,DB)
640 FOR T=1 TO 9
650 NEXT T
660 NEXT DB
670 NEXT F
680 GOTO 100
690 END
```


NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA, R3C 2Z6

EDMONTON TIERS
PO BOX 11983
EDMONTON ALBERTA
T5J 3L1

# QUED PRO QUO

BY PAUL DEGNER

It was nice to contribute a little more than I usually do to last month's newsletter. I want to thank Mike for giving me the opportunity to do so.

I wrote a review last month on Clint Pulley's c99 language in attempt to highlight what you can or cannot do with his subset version of C and I think I had overstressed the word CANNOT too much. Since this review, Clint has sent out version 2.0 of c99 and has added more CANs than CANNOTS such as relative file access, floating point operations, underscore, new assignment operators, and new statement keywords. This version will be available in the May monthly disk collection.

In the last newsletter, I published a c99 program that I wrote up to convince myself that I am still a programmer rather than a software collector. Programmers, no matter how good they may be, do make mistakes and I'm no exception! After having my initial version running, I decided to clean it up and in doing so I added a semicolon to where it shouldn't be as in the line while( i < leninbuf); so please correct it if you intend to use this program. If it was left as is, it could lock up your computer. Thanks to my sister's cat, Daisy, for pointing out the error!

I finally received the shipment of disk drives from B.G. Micro last month. The brand shipped was 142 LX Qumetraks. So far there isn't any apparent bugs though the 142s draw far too many watts to have them connected to a single power supply. Steve Zabarylo was kind enough to fix my problem. If you are thinking of buying Qumetraks and have a PEB, talk to Steve on how to beef up your power supply! By the way, I'm powering my Panasonic 1/2 height with a V2.2 console power supply, available at J&J. It seems a very reliable and better alternative to the other power supplies they sell.

Briefs:

While scanning Mike's stockpile of medium range newsletters for documentation on how to build Super Carte I found a article on another way to use your load interrupt switch as appeared in Rick Lumsden's February 85 issue of R/D COMPUTING newsletter.

An article by Jon Bannister of 9T9 users group in Toronto described a modification to the bus lines found in the speech synthesizer to utilize (by grounding) the LOAD interrupt line on the 44 pin I/O. This is similar to Bill Gronos' GROM BUSTER with the switch debounce through hardware--rather than software. This causes the computer to do a BLWP to vector >FFFC where >FFFC contains the Workspace Pointer and >FFFE the Program Counter. So at any time in the execution of a program (like when it inevitably locks up) I press the button and branch to the debugger. Jon's device is pretty easy to make. You need a momentary contact, normally open bush button switch (Radio Shack #275-1547), a .1 mF bypass capacitor (#272-135) and a 2.2K resistor (#271-1325). Solder the capacitor across the switch keeping the leads as short as possible. Solder one lead of the resistor to one side of the switch and the other lead to a 7" insulated wire. Connect the other end of the wire to LOAD pin on the speech synthesizer. This is pin 13 on the I/O buss. Looking at the edge of the card at the upper right of the console, pin 13 is the seventh pin from the left on the bottom. Flip the board upside down so that you cannot see any components and place the black female connector on the right side. Pin 13 LOAD is then seventh from the bottom.

A second 7 inch insulated wire should be soldered to the other side of the switch and then to Ground - leads 11 12 13 and 14 from the bottom (with the black connector on the right). You can easily recognize them because they are all soldered together.

All that remains is to mount the switch inside the speech synthesizer under the hood. You'll need 5/16th inch hole for the Radio Shack switch.

Now, if you've made it this far, put in your E/A module, connect the modified Speech Synthesizer. Place the E/A disk with DEBUG on it in drive #1 and run the following program:

```
100 CALL INIT
110 CALL LOAD(8228,96,0)
120 CALL LINK("DSK1.DEBUG")
130 CALL LOAD(-4,131,224,112,190)
140 CALL LOAD(8228,160,0)
150 PRINT "PRESS Q THE ENTER"
160 CALL LINK("DEBUG")
170 END
```

This will load the DEBUG utility. Now enter BYE to leave basic and select an option of Editor Assembler - e.g.: Load and Run. When you press the LOAD button on your speech synthesizer you should be in the debugger. To leave the debugger, use FCTN QUIT.

TIers on TimeLine say Unisource ceased operation. One less supplier of TI wares!

The April 26th TI Fest in Ottawa has come and gone. It supposedly was a smashing success with around three hundred and fifty people attending. Many distributers from North America came to show their wares. Myarc and Horizon Computers brought their wire wrapped 'Noah' computer and their 192K RAMdisk. More details as they come in!

Millers Graphics summer catalog are mentioned several interesting products. There is a book called The Orphan Chronicles written by a good friend, Ron Albright. The history and future of our computer is supposedly contained within. #B007 $9.95. A smart disassembler called DISkASSEMBLER and is said to disassemble DIS/FIX 80 or PROGRAM IMAGE Files right off the disk! #UT03 $19.95. A game called Night Mission which seems to me a lame duck. #G10D (or #G10C for cassette users) $19.95. If interested see Paul Degner for more information.

That's all for this month. Sorry for shortness but I've been playing with c99 to much.

April 23, 1986.

Winnipeg 99/4A Users Group
c/o Paul Degner
1105 Church Avenue
Winnipeg, Manitoba
204-586-6889

# ORDER FORM

| Quantity | Software title | Price Each | Total |
|---|---|---|---|

## Educational Software

| | | |
|---|---|---|
| ..... | Addition & Subtraction I ++ Command..... | $ 5.88..... |
| ..... | Meteor Multiplication Module..... | $ 2.88..... |
| ..... | Multiplication I ++ Command Module..... | $ 2.88..... |
| ..... | Scholastic Spelling Level 3..... | $ 5.88..... |
| ..... | Scholastic Spelling module..... | $ 2.88..... |
| ..... | Teach Yourself Extended Basic..... | $ 2.88..... |
| ..... | Zero Zap..... | $ 2.88..... |

## Games

| | | |
|---|---|---|
| ..... | Blasto +++ Command Modules..... | $ 5.88..... |
| ..... | Centipede..... | $15.88..... |
| ..... | Connect Four +++ Command Module..... | $10.88..... |
| ..... | Defender..... | $15.88..... |
| ..... | Dig Dug..... | $15.88..... |
| ..... | Donkey Kong..... | $15.88..... |
| ..... | Jawbreaker II..... | $15.88..... |
| ..... | Jungle Hunt..... | $15.88..... |
| ..... | Moon Patrol..... | $15.88..... |
| ..... | Ms. Pac Man..... | $15.88..... |
| ..... | Pac Man..... | $15.88..... |
| ..... | Protector II..... | $15.88..... |
| ..... | Shamus..... | $15.88..... |
| ..... | The Attack XXX..... | $ 2.88..... |
| ..... | Tombstone City 21st Century..... | $ 2.88..... |

## Home Productivity Software

| | | |
|---|---|---|
| ..... | Home Financial Decisions Module..... | $ 5.88..... |
| ..... | Personal Real Estate..... | $ 5.88..... |
| ..... | Physical Fitness Command Module..... | $ 2.88..... |

## Books

| | | |
|---|---|---|
| ..... | Beginning Basic Manual..... | $ 2.88..... |
| ..... | Creative Programming Alstar PR..... | $ 2.88..... |
| ..... | Creative programming Volume III..... | $ 2.88..... |
| ..... | Users Reference Guide..... | $ 2.88..... |

Sub Total _____

$ 2.00

COMPUCENTRE®

Add $2.00 Shipping/ Handling _____

Total _____

9001 Louis H. Lafontaine, Anjou, Québec H1J 2C5 (514) 354-3810