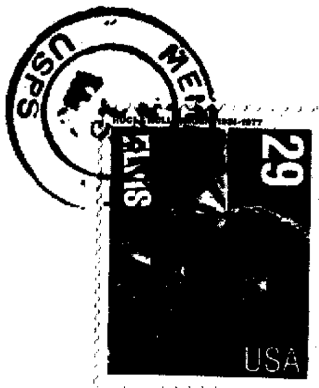
Mid-South 99 Users Group
P. O. Box 38022
Germantown, Tn. 38183-8022

FIRST CLASS MAIL

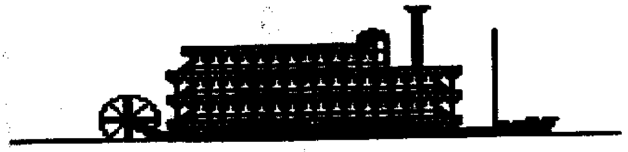
UG 2/86
DALLAS TI USER GROUP
P.O. BOX 29863
DALLAS, TX 75229

MEMPHIS TN
USPS



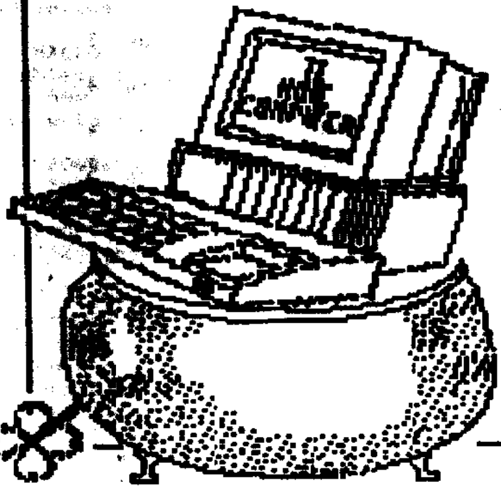
TIDBITS

MID SOUTH 99 USERS GROUP



MEMPHIS TENNESSEE

MARCH



1993

Newsletter of the MID-SOUTH 99 USERS GROUP-Vol 11, #3-MAR 1993

TIDBITS

OFFICERS

MAR. 1993 INDEX

RAG's RICHES

-- Mark Schafer
Bluegrass 99'ers, January, 1993
Louisville BYTEMONGER newsletter, February, 1993

I just recieved a flyer from R. A. Green of RAG Software detailing all of his fairware offerings after I sent him a fairware payment. The following is a synopsis of the list. I'll try to remember to bring it to the meeting if you want more details.

MULTIPLAN V4.03 - This is a greatly speeded up replacement for the Multiplan disk.

TI WRITER V5.1 - This is a disk based update of TI's word processing cartridge. New editor commands include top and bottom of file, Format File, and View File. New formatter commands include As is text and changing special characters. Most operations have been speeded up.

GPL Assembler V2 - This program produces object code from GPL source code. Since specs have never been released on GPL, it makes up its own syntax for all the commands.

GPL Disassembler V1 - This program produces source code from GPL code in GROM/GRAM. It is complete and allows you to assign real names to numbers. This and the program above require a GRAM device.

MESS - Memory Exploitation Supervisor System is a software package the RAVE 99 Memory Enhancement Card. It allows you to restart programs instantly, program the card with assembly, keep an editor resident, and use a buffer for program I/O.

GUMS - GRAM User Menu System extends the cartridge menu for GRAM device owners.

MACRO ASSEMBLER V8 - This is a replacement of TI's assembler that adds macros and a few other features. It adds three new directives and allows you to quote strings with '. It comes with a set of macros, and the user can make up his own.

PROGRAM LINKER V3 - This is that great linker I referred to in my "Link It Right" article. It allows the user to use libraries to resolve references. The source code is not required to turn object code into a program. It also comes with a utility that allows the user to add new libraries.

As you can see, Mr. Green seems to be dissatisfied with many standard TI programs, so he just programs his own! In the documentation to the programs of his that I have, he doesn?t mention a specific amount of money to send, but the flyer mentions $10. The address is:

RAG Software
R. A. Green
1032 Chantenay Dr.
Gloucester, ON, Canada
K1C 2K9

THE COMPUTER and VCR

(thanks to North County 99 UG)
from the pages of Spirit of 99 , February 1993

the following is an excerpt from Barry Traver's Classic Computer Column in the Computer Monthly.

"Have thought about introducing your to you VCR? Although the two may be, in some sense, competitors (e.g. for consumer dollars) they actually work very well together.

"Example: suppose you have a minimal system (say, a console and a few cartridges, including the Terminal Emulator II), and you've gotten interested in telecommunications. You've purchased an inexpensive modem and become a subscriber to CompuServe, Delphi, or GEnie. Since it costs you dollars per hour for the time you're on line, is there any quick and easy way without a disk system, cassette system or printer that you can capture your sessions on while on-line and then view later at your leisure what you saw earlier in the screen?"

"It's simple if you have a VCR. Just hook up your VCR between your computer and your TV, set your VCR to record channel 3 or 4 (whatever is appropriate in your area), and you're in business. After you are off-line, you can play back the tape just as you would any other videotape and, unlike other capture methods, you have captured the audio as well as the video, the sounds as well as the sights. This is exactly what I did myself in the early days before I had a printer or a disk system and it accomplished it's purpose well."

"I should note that the hook up connectors for recording are a bit different, depending on whether you are using a TV or an monitor. If you are using a TV, you will hook up your RF modulator to the 'antenna in' connection on your VCR, setting your VCR to channel 3 or 4. Hook up the VCR to the TV in the normal fashion . . "If you are using a monitor, the situation changes somewhat, but it's still rather simple. Instead of using a RF modulator, you'll be using a video cable, hooking it up the the 'video in' and 'audio in' connections on your VCR. In this case, you'll need to set your VCR to 'line' rather than to a TV channel. Again, you would hook your VCR to the monitor in the normal way.

ANOTHER 96 CHARACTERS?

by Jim Folz
Charlotte U G

from the Great Lakes Computer Group, January, 1993

Many Gemini printer users don't realize that they can access another 96 characters in their printers. The 99/4A uses seven data bits to send characters to be printed. The additional characters in the Gemini printer (and some others) become active when the eighth bit is turned on. Gemini engineers have provided an escape sequence that turns the eighth bit ON and OFF.

In TIW FORMATTER, .TL60:27,62 will make the eighth but go ON when a < is found in the text. If you also put in a printer

manual to make sure your codes are not different.)

Many users will find it easier to use the CTRL-U sequence to access these codes, when working from the EDITOR, so as not to have to use the FORMATTER. In that case, the sequence would be: CTRL-U, FCTN-R, CTRL-U, >, to turn the bit ON and CTRL-U, FCTN-R, CTRL-U, =, to turn it OFF.

When the eighth bit is turned ON, the characters change as shown in the table.

The eight bit simply adds 128 to the ASCII value of the character you type. Consult your printer manual for the characters you want to print, subtract 128, and use the ASCII character of the resulting number.

For a complete printout of the ASCII characters and the resulting numbers, enter and RUN the following BASIC program.

```
100 REM PRINT CHARS 128+ !2-COL
105 OPEN #1:"PIO"
110 PRINT #1:CHR$(15) ! COND
115 FOR X=32 TO 80 ! 2-COL
120 PRINT #1:" ";X;" ";X+128;" ";CHR$(X+128);
    "          ";X+48;" ";X+48+128;" ";CHR$(X+48+128)
130 NEXT X
140 CLOSE #1::END
```

FOR SALE

2 TI99/4a consoles, PEB with 32k, RS232, Extended BASIC and various software. Best reasonable offer, call Ralph Wilson at (901) 327- 3702.

2 Seagate ST-225 20mb MFM hard drives - $50 each, TI PEB with 32k, TI disk controller, SS/SD drive, like new rarely used, best reasomable offer. One 286 16mhz system board with 1mb memory - $50. Call Gary Cox (901) 358-0667.

WRITING PROGRAMS
A PRIMER PART 01

by Wesley R. Richardson
Northcoast 99ers
Cleveland, Oh.
September, 1991

PURPOSE

The purpose of this article is to encourage those of you who want to write a computer program but say "I don't know where to start or how to start." This article will provide some guideance to answer those questions.

SCOPE

The focus of this article will be on the Texas Instruments (TI) computer, TI-99/4A, using the Extended BASIC (XB) language, but the concepts will apply to any type of computer and language.

INTRODUCTION

The reason many people bought home computers or personal computers was to learn what this computer craze was all about. What was not so apparent when the computer hardware was purchased, brought home and set up, was that the hardware was useless without software. The software consists of the programs that do almost anything that you would like the computer to do, yet new programs are being written every day. One reason that computers are powerful is that they are flexible and can do a variety of tasks depending upon the program that is controlling the computer.

In spite of the vast nmber of programs available, new programs are being created because of new applications for the comuter, or faster or better ways of doing previous programs. Programs are also being written because it is fun, creative, and a learning experience to write programs. Assumnming that you have the desire to write a program, read on.

CONCEPT STAGE

The program concept stage is to me the hardest part of writing a program. The concept stage is where the idea for the program is created. this is the initial definition of what the program will do. Note that the functionality of the program usually is different than the concept, because of constraints due to time, speed, memory, hardware, or knowledge of the programmer. The greater the detail of the concept before programming, the closer the final program will reflect the concept. Regardless of your desire to write a program, you must have some idea as to what the program will do to enable you to start writing the program.

PROGRAM LANGUAGES

For the TI-99/4A, we have several languages in which a progam can be written, including: BASIC, Extended BASIC, SUPER Extended BASIC, Assembly, c99, FORTH, FORTRAN, G, LOGO, PASCAL, TURBO PASCAL, PILOT, and the Graphics Programming Language

(GPL). Many of these panguages support calls to another language such as to Assembly from Extended BASIC. Your selection as the program language will depend upon what the program is intended to do. The factors in this decision include what types of screen displays you will use, required speed, availability of the required hardware and software, and yur programming knowledge. Because of the ease of learning, BASIC and Extended BASIC are the most popular for the beginner to advanced programmer. Assembly is the closest to machine code and has the fastest potential and greatest access to machine capability. You can also write the program in one language and prove the concept and then rewrite the program in another language for speed, for example. Finally, writing a program in a language which is new to you should also be considered as that will expand your programming knowledge.

THE COMPLETE PROGRAMMING BOOK

It doesn't exist. An often heard question is what it the "best" book for learning Assembly programming. The best programming book and the complete book on how to play golf, tennis, or fishing do not exist. Programming is a learned skill in which you never stop learning. Just as there are always new words for you to learn in a language, your programming knowledge and for that matter, your computer knowledge can continue to grow for a lifetime, yet you will never know everything about programming. This is not meant to discourage you, but to invite you to step into the vast world of programming. It it were so simple as to be able to fit all programming knowledge in one book, we would all be bored with it very quickly. The beauty of programming is that each programmer can add their personality to their creation, even with the same outputs. the way to start programming is to start with the basics such as printing to the screen, and then increase the program complexity as your programming skill expand.

The other reminder is that the end product that you see as a finished program, has gone through many revisions to refine the prgram. A program which I recently wrote, MAGNA-CALC, had 82 revisions in the XB portion and about 20 reisions in the Assemby portion before I was ready to release it. Each routine that you need and each error that you make in writing a program become small puzzles for you to solve.

PROGRAM OUTLINE The standard guidelines for writing a program say that you should flowchart the program before ever starting to write code. This is sound advice, but is not how I write programs. The way I start is to start at the top level of the program and define a set-up section (100), a main menu (200), the options sections (300, 400, 500), closing routines (800), and a subroutine section (900). The numbers refer to the line numbers in the PROGRAM01X listing at the end of this article.

Build the program top down and bottom up. What does that mean? The overall program outline (top down) can be created, even if the program cannot do anything except go through some branches. The bottom up programming comes when you define the very specific routines which do some specific task, and these are called from higher levels when required. An example of a specific routine might be one which gets a filename, verifies it, and then opens that file. The top level routines allow the user to make selections from a category of options and then work down to the more specific function desired. When you write the program, you should set up the framework for the program, and then write the programming commands down to the detail level in one section at a time.

SUBROUTINES

The subroutines are programming steps which have similar functions but which are called from different sections of the program. The advantages of subroutines are that they save time by not writing the same program steps several times for different sections of the program, and they save memory by avoiding duplication. Library subroutines are routines written by someone else, which you call using specific formats. They are particularly useful for complex operations and calculations such as SIN(X) and CALL SPRITE().

WRITING CODE

To begin writing code or programming instructions, you will need a reference guide which lists the commands and syntax required for statements in the programming language which you selected. When you purchase a program language, it is expected that you will be provvided with such a reference list.

Your first task will be to start learning the commands you need by looking them up. In well documented manuals, the commands will work exactly as described. Only through your use of the commands will you determine if you understand the syntax and application of the command and if the documentation was correct. Some languages also have hidden commands which they do not list in the documentation, but which you may discover by accident or by looking at the programming code of the language itself.

In a manner very similar to learning a foreign language, you will need to develop your vocabulary in the selected programming language. A way to accelerate this learning process is to study programs written by other people. By study I mean work through the program instruction by instruction so that you understand how the program functions. On a copy of the program, not the original, make modifications to the program to change the displays, inputs, outputs or whatever improvements are important to you. Be sure to add a reference near the beginning of the program as a reminder that this is a modified version, and save it using a modified file name.

When you are creating something new, like a new or modified program, plese make frequent backups. I believe in saving my work to disk every 15 - 20 minutes, with a printout about once per hour. Also when you are ready to trun off the computer, be sure to save to disk and print your work up to that point. It is much easier to re-enter a program from a listing after the power goes out unexpectedly than it is to recreate your work.

ERROR DETECTION AND CORRECTION

There are two levels of errors in programmong: syntax and logic. The syntax type of error will often be detected when you try to run or compile the program. This type of error is relatively easy to detect as well as correct. The logic error is often very difficult to identify and may go undetected even

by the program users. More complex programs increase the difficulty of detection of the logic error, and even if detected, may be even more difficult to correct. It is an obligation of the program writer to test the program before releasing it. Users will assume the program fuctions properly untill proven otherwise, and could come the incorrect conclusions using your program if it has errors in it.

I believe it is fundamental that each revision of the program is clearly indicated. This can be done by revision number or a date. Any user has the right to know if two copies of your program are the same, or if different, which is the later version.

When writing a new program, if possible, test each section you complete. It will be easier to make corrections when that section of program is fresh in your memory. If you have a large program that you are adding a routine to, it is helpful to first create the routine as a seperate small program, using only the minimum supporting lines required. Once the routine works, it can then be incorporated in the main program.

I don't believe that even the most experienced programmer sits down and writes an error free, working program, without making corrections or modificatins. As I said earlier, the fun of programming is solving each of the individual problems which eventually leads to the completed program. A corollary to this is that no program is ever competed. It is always possible to make improvements, but at some point the decision is made to share the program with others. The first distribution of the program should be on a limited basis, called a beta test. During beta testig, people that are similar to your end user will use your program and provide feedback to you. You should make every attempt to limit distribution at this point so that corrections which are made as a result of the beta testing can be distributed to all people who have copies of the program.

The additional pleasure from writing programs is acceptance by others. This can come from your friends, user groups, or selling your program. It is also the point which can be frustrating if you set your expectations too high. All programmers fall in love with their creations after they have spent hours working on the program. To them, the use of the program is obvious and user friendly. My recommendation is to find enjoyment from the time you spent learning and programming. If one or two other people like your program, then so much the better. Whether it is accepted or not, use the feedback for improvements when you write your next program.

GETTING STARTED NOW

Using Extended BASIC, type in the program listing shown below. Save it to disk using SAVE DSK1.PROGRAM01X. Then run the program. If you encounter any errors, check your program statements against the listing. If you have a printer, this may be easier to do from a printed copy of the program. This program is very simple, but the format can be used on complex programs. With larger programs, each of the options sections could be further divided into menu selections. This process can be extended down through as many layers as are needed to select all of the desired functions. It there are commands in the program which you do not understand, then please look them up in the Extended BASIC manual.

Once your program is running correctly, start expanding one of the options sections to do something more. This could be to take an input sentance and print it to printer, for example. That is all there is to it, now why don't you give it a try?

```
100 REM PROGRAM01X
110 REM TI-99/4A EXTENDED BASIC
120 REM WESLEY R.  RICHARDSON, JULY 1991
130 REM NORTHCOAST 99ERS, CLEVELAND, OH
140 REM VARIABLES K,S
200 REM MAIN MENU
210 GOSUB 950
220 DISPLAY AT(4,4):"1 = OPTION 1"
230 DISPLAY AT(6,4):"2 = OPTION 2
240 DISPLAY AT(8,4):"3 = OPTION 3
250 DISPLAY AT(10,4):"4 = END
260 GOSUB 910 :: K=K-48 270 IF (K<1)+(K>4) THEN 260
280 ON K GOTO 300,400,500,800
300 REM OPTION 1
310 GOSUB 950
320 DISPLAY AT(10,4):"OPTION 1"
330 GOXUB 910 :: GOTO 200
400 REM OPTION 2
410 GOSUB 950
420 DISPLAY AT(10,4):"OPTION 2"
430 GOSUB 910 :: GOTO 200
500 REM OPTION 3
510 GOSUB 950
520 DISPLAY AT(10,4):"OPTION 3"
530 GOSUB 910 :: GOTO 200
800 REM END
810 GOSUB 950
820 DISPLAY AT(8,4):"THE END" :: STOP
900 REM SUBROUTINES
910 DISPLAY AT(22,4):"PRESS A KEY"
920 CALL KEY(0,K,S) ::A IF S=0 THEN 920
930 RETURN
950 CALL CLEAR
960 DISPLAY AT(2,2):"PROGRAM01X DEMO"
970 RETURN
999 END
```

_________910719 WR PROGRAM01_________

WRITING PROGRAMS
PART 02 VARIABLES

by Wesley R. Richardson
Northcoast 99ers
Clevaland, Oh.
NOVEMBER, 1991

PURPOSE

The purpose of this article is to help programmers understand the use of constants and variables, including arrays, in TI Extended BASIC.

SCOPE

While many of the concepts regarding varibles apply to any programming language, the specific commands, functions and statements in this article refer to those on the Texas Instruments (TI) computer, TI-99/4A, using the Extended BASIC (XB) language.

INTRODUCTION

Constants and variables are parameters. The advantage of using parameters is they allow a program to be written which will work even if the person writing the program does not know which numbers or words you will use in the program when you run it. Examples of this concept are the handheld calculator which uses numeeric variables that you entered when you multiplied two numbers, and a word processor which will work with any words you type.

After the constants and variables are entered in a program, the parameters are used by an expression to create an output, or to make a decision based on your input. Since most computer programs do not modify themselves while running, the use of variables in the program gives it the flexibility and power to be used in many different applications.

STRING CONSTANTS

A constant is a parameter whose value cannot change during the running of the program. A string constant is a group of alphanumeric characters up to 153 characters long. They are usually enclosed in quotes. String constants have two primary uses in programming, one in relational expressions to evaluate a string variable, see a), and the other in forming new strings for displays or printing, see b).

a) IF A$="Y" THEN 300

b) B$=C$&" PAGE NUMBER IS "

The letters and spaces between the quotation marks comprise the two string constants in these examples. I am already getting ahead of myself, because I have used string expressions which I will discuss later.

NUMERIC CONSTANTS

Numeric constants are similar to string constants but numeric means they are numbers. Since they are constants, again they cannot change during the program, because if they did, we would call them variables instead of constants. How can something that is constant be useful in a program? Here are two examples of how constants could be used:

d) IF A=4 THEN 300

e) B=5*C

The numeric constant A is compared with the numeric constant 4 to make a decision. the numeric variable C is multiplied by a constant 5 and B is set equal to the result. I guess I just can't write about one part of parameters without introducing the next.

STRING VARIABLES Each programming language has different rules which define allowable variable types. Extended BASIC on the TI has one of the most flexible requirements of all computers. Variables can have names of up to 15 characters long, with the first character any of the letters of the alphabet, the "at" (@) symbol or the "underline" (_) symbol. After the first character, the remaining characters may also include any of the numbers. For string variables, the last character must be a dollar ($) sign. This limits the number of string variable names to about 10, 000, 000, 000, 000, 000, 000, 000 while numeric variables can be selected from a number a little larger than 38 times that number. Examples of string variable <u>names</u> are:

f) A1234567$ and _A1B2C3D4E5F6G$

Note that the TI converts all lowercase letters to uppercase when they are used in a variable name.

One further classification of variables is whether they are scaler or array type of variables. Array variables are discussed later. A scaler variable means a "real" number in mathamatics. On the TI-99/4A, a scalar string variable if a string variable whose value is any <u>single</u> string of alphanumeric characters up to 256 characters long.

Examples of scalar string <u>values</u> are:

g) "1234567" and ABCDEF123456

NUMERIC VALUES

The names of numeric variables in Extended BASIC have the same specifications as do the string variable names, except the dollar sign ($) is not used. Thus, a numeric variable name can be up to 15 letters or numbers long, but cannot start with a digit. Examples of numeric variable <u>names</u> are:

h) A1234567 and _A1B2C3D4E5F6G7

The values which a numeric variable may be are determined by the numeric range of the computer. In Extended BASIC, the largest number which can be printed is 9.99999999E127 and the smallest is 1.00000000E-128. Normally XB will only print or display numbers up to E99, with a maximum of 10 characters, however if you use an IMAGE statement or a PRINT USING or

DISPLAY USING, with 5 circumflex symbols ( ) for the exponential term, numbers with exponents of +-100 or more can be displayed or printed. Examples of numeric <u>variables</u> are:

i) 9.345 and -123.456 and 3.45678E-37

STRING ARRAYS

Arrays are a method of notation which allows a programmer to refer to or use a group of variable names and values in such a way that it greatly simplifies programming. There are many analogies of arrays which we use in our everyday life, and I would like to present one of those, which will perhaps help you to understand arrays.

If I wished to direct someone to a piece of information which was on a certain page of a document, I could say look on page 4, for example. The variable name is the name of the document, so the reader knows which document I am referring to. I might also say to read pages 4 through 7 of that document. As long as I gave the document name and the page numbers, the reader should be able to read the pages I indicated. The document is then like a one dimensional array, with each page being a different, but ordered element of that array.

Now what if I had several documents, all of which related to a particular subject. I could give them each a different name or variable name, or I could assign each a sequential number, and give them a common name such as TAXFILE. If I then told you to read TAXFILE, document 1991, page 3, you would know exactly where to look. This is like a two-dimensional array. The year sequence is one dimension, and the page is the second dimension. If all of my tax records are in drawer one of a file cabinet, and all of your records are stored in drawer two, and other records are in drawers three and four, then I have created a three dimensional array of tax information. I can uniquely identify which page I refer to by giving drawer number, year, and page number. Of course this can be extended to as many dimensions as we wish to use.

To use an array in Extended BASIC, you first define it with a DIM statement such as DIM TAX$(23, 5, 4). The name for an array must follow the same conventions as do any other string or numeric variables. In the DIM statment you also tell the maximum number of elements for each of the dimensions. This is slightly modified by the OPTION BASE statement which defines whether you start counting elements at zero or one. The default is 0 if you do not use OPTION BASE. When you use the array, you tell the program which element you want to use by the values within the parenthesis. TAX$(2, 3, 4) is thus referring to the string value contained in the second drawer, the third document, and the fourth page.

Arrays in XB can be up to seven dimensions. The most common in programming is one dimension, followed by two, three and higher dimensions. The other constraint is the memory space available. For the TI, the memory limits arrays to about 150 string elements for most programs, but this greatly depends upon how many characters are in each element. Large computers can be programmed with arrays of hundreds and even thousands of dimensions. This many dimensions is useful for demographic analysis. A way to determine how much memory you are using is to run your program, fill the array with typical strings, and then do a FCTN4 break, then type SIZE to see how much stack and program memory is free. When your program gets under 1,000 bytes of stack free, you will need to be careful about adding any more variables or increasing array sizes.

An element of a string array contains a string value which can be used in exactly the same way that a scalar string variable returns a string value.

NUMERIC ARRAYS

Numeric arrays use the same conventions of numeric variables, with the additional numbers determining which element we are referring to. As stated under string arrays, all arrays must be first defined with a DIM statement before they can be used. Numeric arrays evaluate or return a number whose value was stored in the element or position indicated.

Now back to the advantages of arrays for the programmer. I can write a loop to check each element to see if it has a certain value or to print all of the values to the screen. A loop like: FOR I=1 to 10 :: PRINT WEIGHT(I) :: NEXT I provides a shorthand notation to refer to all 10 elements or values of weight. This same method could be used to see if your name appeared on any page of TAXFILE from above.

Two arrays can be used together to maintain data which is related. For example a string array may have last names and forst names, ad a parallel numeric array may contain wages, ages, and hours worked. When used in a program, the array index can point to an element in each array which contains information about the same subject.

STRING EXPRESSIONS

Now we know what string constants and variables are, how can we use them? I will use A$, B$, and C$ to represent any parameter which evaluates to a string value for the following discussion.

Of course we can simply print or display the string A$. We can also combine strings using & for concatenation. If A$="HI " and B$="BOB", and C$=A$&B$, then C$="HI BOB". This provides a way to combine any strings which we select. We can repeat a string A$ using C$=RPT$(A$,3) and get C$="HI HI HI ". The opposite of combining strings is to parse them. The SEG$ function combined with the LEN and POS functions allow selection of any subset of letters from another string. LEN is useful to tell us how long the string is, and POS to locate a particular character, such as a space character. If A$="FRIDAY", then SEG$(A$, 1, 3) returns "FRI"

The ASC(A$) function will return the ASCII code of the first character of the string A$. We can also use the VAL(A$) to convert a string to a number assuming the string contains the proper characters.

NUMERIC EXPRESSIONS

Numeric expressions operate on any numeric parameters to return another numeric parameter. The primary numeric operators are the plus (+), minus (-), multiply (*), divide (/), and exponentiation ( ). In addition, the following functions operate on numeric functions to return a numeric expression: ABS, ATN, COS, EXP, INT, LET, LOG, MAX, MIN, PI, RANDOMIZE, RND, SGN, SIN, SQR, and TAN. Both numeric variables and numeric

expressions can be used in loops, as array element pointers and in statements and functions which require numeric expressions as inputs.

The functions CHR1$ and STR$ use numeric inputs and return string values.

RELATIONAL EXPRESSIONS

The relational expression operators are: = < > <> <= >= which stand for equal, less than, greater than, not equal, less than or equal, and greater than or equal. These operators may be used on both numeric expressions and string expressions, but not at the same time. The result will be -1 if the test is true and will be 0 if it is false. The output value of a reational expression can be used as a numeric value or in a branching test like an IF-THEN statement. Examples of the use of relational expressions would be to test a string to see if it matched the string you were looking for in a list or array. You could use a relational expression to speed up a game if the player's score exceeded a certain value.

LOGICAL EXPRESSIONS

The logical expression operators include NOT, XOR, AND, and OR, in decreasing order of precedence. This means that in the absence of parentheses, performs the NOT operator first, then XOR, then AND, then OR when evaluating a logical expression. Logical expressions form the basis for Boolean algabra. Logical expressions, like relational expressions are used to evaluate relationships and when used that way, return -1 if the test is true and 0 if false. Logical expressions can also be used on the integer numbers in the range of -32,768 to 32,767 and when used in this way, return the decimal result of performing the operator on the binary value of the initial number(s).

CONCLUSION

Parameters, including constants, scalar variables and array variables allow programs the flexibility and power to utilize numbers and strings in a variety of ways. These include decision making for program branching, operating on groups of similar information, and performing calculations with numeric data. Variables are the means for all input and output from a program.

______________911018_WR_PROGRAM02______________

SHAKESPEARE

Adapted from the Official Computer Haters Handbook
by Ed Machonis, Queensborough 99ers ,
(reprinted in the East Anglia 99 newsletter, Feb. 1991)

On the TI Pullout:
"The evil that men do
lives after them ....."
Julius Caeser, III, 1

Upon Reading the E / A Manual:
"Though this be madness,
yet there is method in it."
Hamlet, II, 2

Upon Writing His First Program:
"An ill-favoured thing, sir,
but mine own."
As You Like It, V, 4

On User Groups:
"Misery aquaints a man with
strange bedfellows."
The Tempest, II, 2

Upon Blowing His Last Back-Up Disk:
"If you have tears,
prepare to shed them now."
Julius Caesar, III, 2

On Programming Speech Synthizers:
"Speak the speech, I pray you,
as I pronounce it to you,
Trippingly on the tongue."
Hamlet, III, 2

On The Price of Peripherals:
"Costly thy habit
as the purse can buy ... "
Hamlet, I, 3

On His Subscription to home Computer Magazine:
"Oh what a goodly outside
falsehood hath."
Merchant of Venice, I, 3

On TI's Packaging a SSDD Drive With a DSSD Controller:
"Something is rotten in the state ...
of the art."
Hamlet, I, 4

On Computer Crashes:

"The rest is silence."
Hamlet, V, 2

Whilst Playing an Adventure Game:
"Is this a dagger
which I see before me, ... ?"
Macbeth, II, 1

After Losing a Nights Work To The Quit Key (FCTN=):
"O villan, villian, smiling,
dammed villian."
Hamlet, I, 5

On The Turbo XT Keyboard:
"What's in a name?
That which we call a rose
By any other name would smell ...."
Romeo and Juliet, II, 2

Upon Buying a Geneve with DOS .9:
"Have we eaten of the insane root
That takes the reason prisoner?"
Macbeth, I, 3

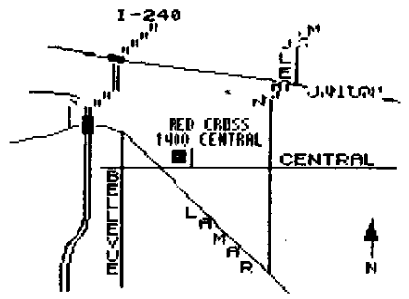
On the TI-99/4A Grom Port:
"The Gods are just,
and of our pleasant vices
make instruments to plague us.
King Lear, V, 3

On Debugging Source Code:
"When sorrows come,
they come not single spies,
But in battalions."
Hamlet, IV, 5

On The Speed of the TI-99/4A:
"My ewes breed not,
My RAMs speed not,
All is amiss."
The Passionate Pilgrim

On Purchasing Software:
"I am a man more sinned against
than sinnning."
King Lear, III, 2

On Software Pirates:
"One may smile, and smile,
and be a villian."
Hamlet I, 5



LOCATION MAP

WORKSHOP : to be announced.

PROGRAM BIT - third Thursday

MARCH 18th , 1993

MEETING: 7:00pm - Red Cross Building - 1400 Central.

6:45pm - Doors Open

7:00pm - Meeting begins, general discussion.

7:30pm - Demonstration to be announced.

9:00pm - Meeting ends.

9:15pm - Late dinner at location to be announced
at meeting.

NOTICE

Information contained in TIdbits is accurate and true to the best of our knowledge. Viewpoints and opinions expressed in TIdbits are not necessarily that of the Mid-South 99'ers. We welcome any opinions/corrections from our readers. Articles may be reprinted elsewhere as long as credit is given to the author and newsletter.

GROUP INFO

Visitors and potential members may receive 2 free issues of TIdbits while they decide if they wish to join (no obligation) On the top of your label is a code. A Y means you are a member, N means 2 free list, UG means user group and S means a business. Beside the Y is a date, one year from that date your dues are due. A dollar sign ($) on the label will indicate that your dues are due. The library is open only to members. Library list is $1. Mail order disk library access is $2 for the first disk and $1 for each additional disk - max of 5 disks per month. Order by disk number only. At meetings, library access is FREE if you exchange your disk for ours or $1 per disk for our disks. Send all mail order library requests to librarian's address! Send dues and correspondence to group address.

CALENDAR

MEETINGS:	MAR. 18, (3rd Thursday!)
WORKSHOPS:	TO BE ANNOUNCED

24HR TI BULLETIN BOARD

The 9640 NEWS BBS 300/1200/2400/4800/7200/9600/12000/14400
Hayes. 901-368-0112

GROUP MAILING ADDRESS

Mid-South 99 Users Group
P.O. Box 38522
Germantown, Tn. 38183-0522

LIBRARY ADDRESS

Jim Saemenes
46 Higgins Road
Brighton, Tn., 38011

MEMBERSHIP APPLICATION

NAME ______________________________ |_| $18.00 FAMILY
ADDRESS ______________________________
CITY __________________ ST ____ ZIP __________
PHONE(_____)___-_______:INTERESTS ______________________________
____
EQUIPMENT,ETC. ______________________________
____

Detach and mail with check payable to: Mid-South 99 Users Group, P.O. Box 38522, Germantown, Tn, 38183-0522.