

President Ira Lieberman 820-6332
Vice-pres Jeff Bleam 346-7590
Secretary Ann Halko 262-8206
Treasurer Barbara Rejician 767-9679

Vol. V, No. 5 September, 1987
Editor Jack Zawediuk 821-1043

LEHIGH 99'ER COMPUTER GROUP

Next meeting: 7:30 PM, Monday
October 19, 1987

Conference Room A-D, Second Floor
Sacred Heart Hospital
4th and Chew Streets
Allentown, Pennsylvania

PRESIDENT ■

For those of you who may have missed some of our meetings over the summer due to vacations, etc. let me try to bring you up to date on a few of the things that have happened.

First of course, I took over as President at the July meeting. Mark DeNardo who has been heading our group for the last few years felt his job was taking more of his time and wanted to limit his involvement. He will continue to bring the clubs' system to the meetings and provide or co-ordinate demos for us as much as possible. Jeff Bleam, who has been handling one of our disk libraries, will also now be our Vice-President. Ann Halko and Barbara Rejician agreed to remain as our Secretary and Treasurer for another term. Jack Zawediuk has graciously agreed to be the newsletter editor.

At the July meeting a motion was passed to establish a separate class of membership for cassette users at \$12 with no initiation fee. These members will have access to the cassette library only and can move up to the regular (disk) membership with the payment of an additional \$12.

This should help us to get some of those younger TI owners who have never expanded their systems to join and make use of our organization. Remember we have several hundred programs on cassette and almost no one making use of them. PLEASE pass the word on to others you know who have TI's or may have put them up on the shelf.

Attendance at the August meeting was small but Mark DeNardo gave a demo of TI/Funlwriter and showed how to make use of these programs without going back and forth between the editor and formatter.

A group of us are planning to attend the TI Computer Expo in Harrisburg on Sat Sept. 12 and will report on it at the Sept. 21st meeting.



Remember, we're still meeting the 3rd Monday of each month at the address listed at the top of this newsletter. Also up there are the names of the officers, the editor and our phone numbers. Use them for any questions. I hope to see most of you returning to the meetings this fall.

EDITOR ♦

I first want to apologize for not writing a newsletter last month. Having two weeks vacation, both of which I spent in the mountains and a week of car repairs, ate up the month kind of fast. One thing about the mountains you get time to catch up on your reading. The many newsletters I read brought a lot of new information and programs being offered. One program I sent for just came this week. It's PRINTIT by Rodger Merritt. It does banners, titles (or letterheads) in fancy fonts with graphics, catalogs any disk, and prints a catalog or label in subscript. It will also print graph paper. Included on the disks are programs to design your own fonts and graphics. I haven't used it much yet but so far I find it easy to follow the menus and get it to print out the samples. I know you've all heard this before but,

It's not impossible for you to write something for this newsletter! It doesn't take THAT LONG to write down what you think about a program. We don't need a punch by punch review. A few lines do help others decide if they can make use of a program. If you are not into reviews how about a comment on anything computer related. You can share the your knowledge on disks, drives, modem, console, or anything, good or bad. It helps others very much to know before they buy.

Don't tell yourself your going to do it, DO IT!!!

SAMPLE DATA  **JGMSJ** 

FILES			FREE 692	USED 28	
FILENAME	SIZE	FORMAT	FILENAME	SIZE	FORMAT
ERR/92	7	D/V-80	TEST	7	D/V-80
SAMPLE	2	D/V-80			

HOME SECURITY SYSTEM

How about a home security system for under \$100.00 ? How about under \$50.00 ? Do I have your attention now ? Well you may be well on your way to accomplishing it.

The hardware and software project I'm propos.ing will start with a bare console (we don't want to tie up your expanded system), and a few locally available parts. If you, like I have a spare console or maybe you're one of the many who bought your TI during the big sellout and still havent found a use for it, this series may be for you.

The basic system I have in mind will require a console, some sort of entry keypad , switches at required doors and-or windows and an amplifier and speaker to sound the alarm.

This might sound a bit like we're sarting to get in deep but just a keypad at the front door will start a burgler thinking.

I'm going to try to keep the basic system as simple and the cost as low as possible. As we progress I hope to expand with LED'S at the keyboard, to tell you when the systems armed. Maybe even speach to tell you it's armed. Hopefully I or someone can work out an internal clock with the possibility of turning lights on and off. Another project will be battery backup for power failures.

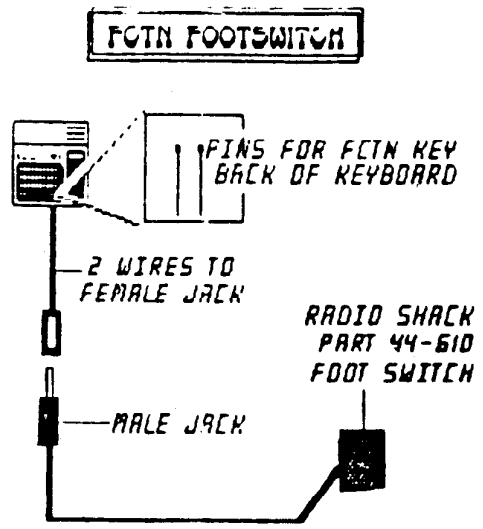
What I have in mind is tapping into the keyboard, primarily the numeric keys and hooking up a keypad at the front door. You don't even have to use all the digits just four can give you many combinations. 2314 134234231 as many as you like. The door and window switches can be connected to any other key. We will also need an amplifier and speaker. We will use the audio out, amplify it to produce a alarm sound at the speaker.

Other security systems don't give you the programmable options we will have. With this system you can change your access code as often as needed. If you go on vacation and have a frend or neighbor watching your house you can easily change the code to one they want and change it back when you return. Or how about speech? When someone is punching in the wrong codes the computer can tell them (THIRD TRY-ALARM WILL SOUND IF YOU TRY TO ENTER-WAIT TEN MINUTES BEFORE RETRY). What about an automatic dialer to the police?

Next month hopefully, I'll have the basic program writen , then I'll start working on the hardware. (no I don't have a system built yet but I have confidence it will work)

Any comments or ideas please don't hesitate to call.

Jack Zawediuk
719 N. 12TH. ST.
Allentown, PA.
18102
(215) 821-1043



Computer Survival Column

You own a computer (A TI99/4a) and the manufacturer decides to stop making that model. What do you do?

Well first you should look at why you bought the computer in the first place. Did you buy it to learn? Or did you buy it to write programs or letters? Or did you just buy it (or receive it) because they were really cheap at the time?

Well now that you answered that question, you can now answer some more. If you said something other than because it was cheap, or you've now used it for more than because it's cheap, then let's continue to explore how to survive with your Orphaned Computer.

I'll zero in on the TI and my thoughts on how to survive, even after several years being orphaned.

If you have an expanded system (at least a disk system), then consider making sure it has all the capabilities you'll need in the future before the devices start to disappear. The following list is probably the hardware you want to get.

- 1) printer port (PIO) 2) Serial port (RS232) normally with 1) 3) 32K Expansion

These devices will open your system to any printer, any modem and the full power of the TI system (Extended Basic) respectively. These are important to be able to use your computer to its full capability.

Now that you have decided to expand your system, we'll talk about software. TI Extended Basic (or any of the other mfg's new EXB's) is the essential primary software you'll need. It will open you system to many of the good software programs you that will make your computer useful.

Next you'll need a word processor to write letters. This according to surveys is the primary use of a home computer. The most popular for the TI are TI-Writer, Funnel Writer. Both require 32K and a Disk System. Funnel Writer requires EXBasic.

Continued on Page 8

Charlotte TI 99/4a Users Group

FAST TERM OVERLAY

by Robert Simms

I have often thought I would like a larger overlay of various programs in which keys other than the top row have special functions. Fast Term is such a program, having numerous key combinations around the board which for some are difficult to remember. I had a 3XS card near the computer for a while, until I mastered the codes. But it is easy to forget, especially if you don't run a program very often. Strips are fairly easy to come by. I thought I would go them one better and make a full overlay. The diagram at the side can be cut right out of this page. I suggest you paste it on a piece of heavier paper--cover stock, etc.--then cut it out. Snip the dotted lines at the ends of the FCTN/CTRL strip, and when you put the overlay in place, you can slide the bottom edge of this strip down into the computer's slot for the regular overlay. The rest of the overlay should now lie in place. The "tails" on the sides are for whatever other information you might want to put there--you could even put phone numbers on it.

BAUD	:SPoolER	:PARITY	:PTR.PORT	:PTR.PAR.	:PTR.PORT	:PTR.BAUD	:HOME	CUR	:40/80	QUIT
DEL	:BUFFER	:REV.OFF	:BREAK	:WINDOW-R	:COLOR F	:COLOR B	:CAT.DSK	:WINDOW-B		
Backspace	CTRL G	IF DUPLEX	TOGGLE	FCTN, SHFT	DIALOG	TOGGLE	FCTN B		TIMER	LEFT
Clear Log	FCTN Y	FILE X	FERS:	FCTN, SHFT	LOG START	/STOP	FCTN :		WINDOW	
Clear Screen	CTRL Z	XMODEM		FCTN, SHFT	FILE		FCTN, N			
CR/LF Toggle	FCTN J	TE 2	ASCI	FCTN, SHFT	TIPRINT	SCREEN	CTRL, SHFT	P	F A	B T E R M
				FCTN	REVERSE	ON	CTRL, D			overlay

FUNNELVEB FARM EXTENDED BASIC TUTORIAL PART 1

Extended Basic Tutorials from FUNNELVEB FARM

1. INTRODUCTION

In this series of notes on TI Extended Basic for the TI-99/4A we will concentrate on those features which have not received due attention in User-group newsletters or commercial magazines. In fact most of the programs published in these sources make little use of that most powerful feature of XB, the user defined sub program, or of some other features of XB. Worse still is to find commercially available game programs which are object lessons in how to write tangled and obscure code. The trigger for this set of tutorial notes was a totally erroneous comment in the TI.S.H.U.G. Newsdigest in June 1983. Some of the books I have seen on TI Basic don't even treat that simpler language correctly, and I don't know of any systematic attempts to explore the workings of XB. The best helper is TI's Extended Basic Tutorial tape or disk. The programs in this collection are unprotected and so open for inspection and it's worth looking at their listings to see an example of how sub programs can give an easily understood overall structure to a program.

Well, what are we going to talk about then? Intentions at the moment are to look at: (1) User-defined sub programs (2) Prescan switch commands (3) Coding for faster running (4) Bugs in Extended Basic (5) Crunching program length (6) XB and the Peripheral Box (7) Linking in Assembler routines

Initially the discussion will be restricted to things which can be done with the console and XB only. Actually, for most game programming the presence of the memory expansion doesn't speed up XB all that much as speed still seems to be limited by the built-in sub programs (CALL COINC, etc) which are executed from GROM through the GPL interpreter. The real virtue of the expansion system for game programming, apart from allowing longer programs, is that GPL can be shoved aside for machine code routines in the speed critical parts of the game, which are usually only a very small part of the code for a game. Even so, careful attention to XB programming can often provide the necessary speed.

As an example, the speed of the puck in TEX-BOUNCE is a factor of 10 faster in the finally released version than it was in the first pass at coding the game.

Other topics will depend mainly on suggestions from the people following this tutorial series. Otherwise it will be whatever catches our fancy here at Funnelveb Farm.

II. Sub programs in OVERVIEW

Every dialect of Basic, TI Extended Basic being no exception, allows the use of subroutines. Each of these is a section of code with the end marked by a RETURN statement, which is entered by a GOSUB statement elsewhere in the program. When RETURN is reached control passes back to the statement following the GOSUB. Look at the code segments.

```

290 ...
300 GOSUB 2000
310 ...
2000 CALL KEY(Q,X,Y):: IF Y=1
    THEN RETURN ELSE 2000

```

This simple example waits for and returns the ASCII code for a fresh keystroke, and might be called from a number of places in the program. Very useful, but there are problems. If the line number of the subroutine is changed, other than by RESequencing of the whole program (and many dialects of Basic for microcomputers aren't even that helpful) then the GOSUBs will go astray. Another trouble, which you usually find when you resume work on a program after a lapse of time, is that the statement GOSUB 2000 doesn't carry the slightest clue as to what is at 2000 unless you go and look there or use REM statements. Even more confusingly the 2000 will usually change on RESequencing, hiding even that aid to memory. There is an even more subtle problem -- you don't really care what the variable "Y" in the subroutine was called as it was only a passing detail in the subroutine. However, if "Y" is used as a variable anywhere else in the program its value will be affected.

The internal workings of the subroutine are not separated from the rest of the program, but XB does provide four ways of isolating parts of a program.

- (1) Built-in sub programs
- (2) DEF of functions
- (3) CALL LINK to machine code routines
- (4) User defined BASIC sub programs

The first of these, built-in sub programs, are already well known from console Basic. The important thing is that they have recognizable names in CALL statements, and that information passes to and from the sub programs through a well defined list of parameters and return variables. No obscure Peeks and Pokes are needed. The price paid for the power and expressiveness of TI Basic and XB is the slowness of the GROM/GPL implementation.

DEF function is a primitive form of user defined sub program found in almost all BASICs. Often its use is restricted to a special set of variable names, FNA, FNB, ... but TI Basic allows complete freedom in naming DEFed functions (as long as they don't clash with variable names). The "dummy" variable "X" is used as in a mathematical function, not as an array index

```
100 DEF CUBE(X)=X#X#X
```

doesn't clash with or affect a variable of the same name "X" elsewhere in the program. "CUBE" can't then be a variable whose value is assigned any other way, but "X" may be. Though DEF does help program clarity it executes very slowly in TI Basic, and more slowly than user defined sub program CALLs in XB.

CALL LINK to machine code routines goes under various names in other dialects of basic if it is provided (eg USR() in some). It is only available in XB when the memory expansion is attached, as the TI/994A console has only 256 bytes of CPU RAM for the TMS9900 lurking in there. We will take up this topic later.

You should have your TI Extended Basic Manual handy and look through the section on sub programs. The discussion given is essentially correct but far too brief, and leaves too many things unsaid. From experiment and experience I have found that things work just the way one would reasonably expect them to do (this is not always so in other parts of XB). The main thing is to get into the right frame of mind for your expectations. This process is

helped by figuring out, in general terms at least, just how the computer does what it does. Unfortunately most TI99/4A manuals avoid explanations in depth presumably in the spirit of "Home Computing". TI's approach can fall short of the mark, so we are now going to try to do what TI chickened out of.

The user defined sub program feature of XB allows you to write your own sub programs in Basic which may be CALLED up from the main program by name in the same way that the built-in ones are. Unlike the routines accessed by GOSUBs the internal workings of a sub program do not affect the main program except as allowed by the parameter list attached to the sub program CALL. Unlike the built-in sub programs which pass information in only one direction, either in or out for each parameter in the list, a user sub program may use any one variable in the list to pass information in either direction. These sub programs provide the programming concept known as "procedures" in other computer languages, for instance Pascal, Logo, FORTRAN. Lack of proper "procedures" has always been the major limitation of BASIC as a computer language. TI XB is one of the BASICs that does provide this facility. Not all BASICs, even those of very recent vintage are so civilized. For example the magazine Australian Personal Computer in an older issue (Mar 84) carried a review of the IBM PCjr computer just released in the US. The Cartridge Basic for this machine apparently does not support procedures.

Perhaps IBM doesn't really want or expect anyone to program their own machine seriously in Basic. You will find that with true sub programs available, that you can't even conceive any more of how one could bear writing substantial programs without them (even within the 14 Kbyte limit of the unexpanded TI-99/4A let alone on a machine with more memory).

The details of how procedures or sub programs work vary from one language to another. The common feature is that the variables within a procedure are localized within that procedure. How they communicate with the rest of the program, and what happens to them when the sub program has run its course varies from language to language. XB goes its own well defined way, but is not at all flexible in how it does it.

Now let's look at how Extended Basic handles sub programs. The RUNNING of any XB program goes in two steps. The first is the prescan, that interval of time after you type RUN and press ENTER, and before anything happens. During this time the XB interpreter scans through the program, checking a few things for correctness that it couldn't possibly check as the lines were entered one by one, such as there being a NEXT for each FOR. The TI BASICs do only the most rudimentary syntax checking as each line is entered, and leave detailed checking until each line is executed. This is not the best way to do things but we are stuck with it and it does have one use. At the same time XB extracts the names of all variables, sets aside space for them, and sets up the procedure by which it associates variable names with storage locations during the running of a program.

Just how XB does this is not immediately clear, but it must involve a search through the variable names every time one is encountered to trade off speed for economy of storage.

XB also recognizes which built-in sub programs are actually CALLED. How can it tell the difference between a sub program name and a variable name? That's easy since built-in sub program names are always preceded by CALL. This is why sub program names are not reserved words and can also be used as variable names. This process means that the slow search through the GROM library tables is only done at pre-scan, and Basic then has its own list for each program of where to go in GROM for the GPL routine without having to conduct the GROM search every time it encounters a sub program name while executing a program. In Command Mode the computer has no way provided to find user defined sub program names in an XB program in memory even in BREAK status. XB also establishes the process for looking up the DATA and IMAGE statements in the program.

Well then, what does XB do with user sub programs? First of all XB locates the sub program names that aren't built into the language. It can do this by finding each name after a CALL or SUB statement, and then looking it up in the GROM library index of built-in sub program names. You can run a quick check on this process by entering the one line program

```
100 CALL NOTHING
```

TI Basic will go out of its tiny 26K brain and halt execution with a BAD NAME IN 100 error message, while XB, being somewhat smarter, will try to execute line 100, but halts with a SUBPROGRAM NOT FOUND IN 100 message.

The XB manual insists that all sub program code comes at the end of the program, with nothing but sub programs after the first SUB statement (apart for REMarks which are ignored anyway). XB then scans and establishes new variable storage areas, starting with the variable names in the SUB xxx(parameter list), for each sub program from SUB to SUBEND, as if it were a separate program. It seems that XB keeps only a single master list for sub program names no matter where found, and consulted whenever the interpreter encounters a CALL during program execution. Any DATA statements are also thrown into the common data pool.

Try the following little program to convince yourself.

```
100 DATA 1
110 READ X :: PRINT X :: READ X :: PRINT X
120 SUB NOTHING
130 DATA 2
140 SUBEND
```

When you run this program it makes no difference that the second data item is apparently located in a sub program. Images behave likewise. On the other hand DEFed functions, if you care to use them, are strictly confined to the particular part of the program in which they are defined, be it main or sub. During the pre-scan DEFed names are kept within the allocation process separately for each subprogram or the main program.

Once again try a little programming experiment to illustrate the point.

```
100 DEF X-1 :: PRINT X;Y :: CALL SP(Y)
:: PRINT X;Y
110 SUB SP(Z) :: DEF X-2 :: Z=X :: DEF Y-3
120 SUBEND
```

This point is not explicitly made in the XB manual and has been the subject of misleading or incorrect comment in magazines and newsletters. A little reflection on how XB handles the details will usually clear up difficulties.

TI BASICs assign nominal values to all variables mentioned in the program as part of the prescan, zero for numeric and null for strings, unlike some languages (some Basics even) which will issue an error message if an unassigned variable is presumed upon. This means that XB can't work like TI LOGO which has a rule that if it finds an undefined variable within a procedure it checks the chain of CALLing procedures until it finds a value. However, unlike Pascal which erases all the information left within a procedure when it is finished with it, XB retains from CALL to CALL the values of variables entirely contained in the sub program. The values of variables transferred into the sub program through the SUB parameter list will of course take on their newly passed values each time the sub program is CALLED.

A little program will show the difference.

```
100 FOR I=1 TO 9 :: CALL SBPR(0):: NEXT I
110 SUB SBPR(A):: A=A+1::B=B+1::PRINT A;B
120 SUBEND
```

The first variable printed is reset to 0 each time SBPR is called, while the second, B, is incremented from its previous value each time. Array variables are stored as a whole in one place in a program, within the main program or sub program in which the DIMENSION statement for the array occurs. XB doesn't tolerate attempts to re-dimension arrays, so information on arrays can only be passed down the chain of sub programs in one direction. Any attempt by a XB sub program to CALL itself, either directly or indirectly from any sub program CALLED from the first, no matter how many times removed, will result in an error. Recursive procedures, an essential part of TI LOGO, are NOT possible with XB sub programs, since CALLing a sub program does not set up a new private library of values.

All of this discussion of the behavior of TI Extended Basic comes from programming experience with Version 110 of XB on a TI99/4A with 1981 title screen. Earlier Versions and consoles are not common in Australia, but TI generally seems to take a lot of trouble to keep new versions of programs compatible with the old. On the other hand TI has also been very reticent about the details of how XB works. The Editor/Assembler manual has very little to

say about it, less by far even than it tells about console Basic. I am not presently aware of any discussion of the syntax of the Graphics Programming Language (GPL), let alone of the source code for the GPL interpreter which resides in the condole ROM of every 99/4A.

Another simple programming experiment will demonstrate what we mean by saying that XB sets up a separate Basic program for each sub program. RUN the following

```
100 X=1 :: CALL SBPR :: BREAK
110 SUB SBPR :: X=2 :: BREAK :: SUBEND
```

When the program BREAKs examine the value of variable of X by entering the command PRINT X, and then CONTINUE to the next program BREAK, which this time will be in the main program, where you can once again examine variable values.

We will now summarize the properties of XB sub programs as procedures in complete XB programs, leaving the details of joining up the various procedures to the next section.

(a) XB treats each sub program as a separate program, building a distinct table of named (REFed) and DEFed variables for each. (b) All DATA statements are treated as being in a common pool equally accessible from all sub programs or the main program as are also IMAGE statements, CHARACTERS, SPRITES, COLORS, and file specifications. (c) All other information is passed from the CALLing main or sub program by the parameter lists in CALL and SUB statements. XB doesn't provide for declaration of common variables available on a global basis to all sub programs as can be done in some languages.

(d) Variable values confined within a sub program are static, and preserved for the next time the sub program is CALLED. Some languages such as Pascal delete all traces of a procedure after it has been used.

(e) XB sub programs may not CALL themselves directly or indirectly in a closed chain. Subject to this restriction a sub program may be CALLED from any other sub program.

(f) The MERGE command available in XB with a disk system (32K memory expansion optional) allows a library of XB sub programs to be stored on disk and incorporated as needed in other programs.

THE END FOR NOW.

Continued from Page 3

The Next most useful program is a spreadsheet program. There is only Multiplan (from TI) available for us. It functions exactly like it does for all other versions of Multiplan for other computers. Any books written to help learn it on any computer will work on the TI. If you look around you should be able to get it for under \$30.

The next most important thing to do is to join a Users Group. Here you can get much help understanding your Orphan. There also you will normally find libraries of programs. Copywrited programs that you can evaluate, and determine if a program fills your needs. If so you can buy for full time use or borrow it if you only need it occasionally. Shareware and Public Domain programs which you can get for a donation to the author if you decide to use it. (By the way, now that the manufacturer has abandoned you, other users are your best source for software)

You can do quite a bit with an Orphaned computer using the existing software and hardware. You don't always need the latest computer to be able to use use it.

As with any purchase, you must use common sense when deciding what to buy or how to add to your computer. Be sure you think any purchase out. "Buy with your head not over it." Be sure to check prices so you don't spend too much for your expansions.

>M. De Nardo
sprite one liner

by Barron Bartlett

Want to frustrate and amaze your Atari, Apple, VIC-20 and Color Computer friends? Type in the following in the command mode with Extended Basic.

```
CALL CLEAR :: CALL SCREEN(5):: CALL MAGNIFY(2)::  
FOR I=1 TO 28 :: CALL SPRITE(1,64+I,16,80,80,3*1,8)::  
NEXT I :: FOR J=1 TO 5000 :: NEXT J
```

Hit ENTER and watch all 28 sprites do their tricks. If you want to see it again, simply hit function REDO, then ENTER again as many times as you wish.

LEHIGH 99'ER COMPUTER GROUP
P.O. Box 4837 * 1501 Lehigh St.
Allentown, PA 18103