THE GUILFORD 99'ER NEWSLETTER

VOL.6 NO.9 SEPT 1989

, Pres. Emmett Hughes, Vice Pres.(584-5108)
Mack Jones, Secy/Treas(288-4280) Herman Geschwind,Pgm/Library(288-0015)
BBS: (919)274-7000(OPUS) ROS: (919)621-2623

The Guilford 99'er Users' Group Newsletter is free to dues paying members (One copy per family, please). Dues are $12.00 per family, per year. Send check to 3202 Canterbury Dr., Greensboro, NC 27408. The Software Library is for dues paying members only. (George von Seth, Ed.:292-2035)

OUR NEXT MEETING

DATE: Sept 5, 1989 Time: 7:30 PM. Place: Glenwood Recreation Center, 2010 S. Chapman Street.

Program for this meeting will be a demonstration of the TI/IBM connection and how to transfer programs from one to the other. Be sure to be there to see another view of computer compatibility

MINUTES

The Tuesday, August 1 meeting of the Guilford 99er Users' Group was held at the Glenwood Recreation Center on Chapman Street in Greensboro, N.C. There were 8 members present.

Since the President and Vice President were both absent, the meeting was called to order by the Secretary at 7:35 P.M.

OLD BUSINESS:

a. The Secretary informed members that a card has been sent to Mr. Peterson informing him we can no longer send him a newsletter since we have not heard from him since his March request for exchange.

NEW BUSINESS:

a. Scott called before the meeting and informed me that he would not be able to make the meeting. He also advised that since he was so pushed up with his work and also into programing with his clone, he is thinking about selling his TI equipment. He is also giving up his job as President. We hate to lose Scott but he said there just isn't enough hours in the day to handle both computers. He is also wanting to buy a 9600 baud modem and the TI equipment would help pay for it. Good luck to you Scott.

b. Herman donated a Sams' II 99A package consisting of a cassette tape with 24 programs on it. Tony Kleen bought it for $5.00. The money went into the treasury. Thanks Herm.

The program was presented by Tony Kleen on TI Base. It was a very good and informative program. Tony went through the entire program and made up a data base for mailing labels and showed how they could be printed out. It was a good job and well received. Thanks Tony.

Next month's program will be on the TI/IBM Connection. Bill will demo how you can copy an IBM disk to the TI and vice versa. We will all be looking forward to that demo.

The meeting was adjourned at 9:00 P.M.

Respectfully submitted,
L.F. "Mac" Jones, Sec./Treas.
Guilford 99er Users' Group
Greensboro, N.C.

FORTH NOTES

By Lutz Winkler

ARRAYS IN FORTH

In Tidbit #9 I introduced the word 2D-ARRAY to illustrate the use of the <BUILDS...DOES> construct. It seems that arrays have kindled as much interest as the subject I was dealing with. So here I am once again at the keyboard to continue what had been left unsaid about arrays.

The word to create a two-dimensional array was given as

```
: 2D-ARRAY ( #rows #columns --- ) <BUILDS 2DUP ( dup the parameters ) , , ( store them ) * ALLOT ( multiply them and allot the space ) DOES> ;
```

and I had indicated that it would create a byte array, i.e., each member of the array could only store values up to 255 (>FF). This puts a limit on its usefulness and it would be better to modify it so it can accommodate 16-bit numbers (up to >FFFF).

There was another word (CELL) which simplified access to the array and it was intended for use with byte arrays:

```
: CELL ( row# col# --- addr ) DUP >R ( dup address and save it to the return stack ) @ ( fetch number of columns ) ROT ( bring row# to top of stack ) * + ( multiply #col by row# and add to col# ) R> ( retrieve address from return stack ) 4 + ( address of first member ) + ; ( add offset )
```

Instead of showing you how to change them, let us define a new word which not only can create an array but at the same time provides the convenience of CELL and is word (16-bit) oriented:

```
: ARRAY <BUILDS 2DUP ( dup the parameters ) , , ( store them ) * 2 * ALLOT ( multiply them and allot the space ) DOES>
DUP ( dup address ) >R ( save copy on return stack ) @ ( fetch number of columns ) SWAP DUP ROT 1- ) ( check column parameter
) IF ." Column out of range!" ABORT THEN SWAP R 2+ @ SWAP DUP ROT 1- ) ( check row parameter ) IF ." Row out of range!" ABORT
THEN SWAP R @ ( calculate offset ) * + 2 * ( " " ) R> ( address from return stack ) 4 + + ; ( add offset )
```

Creating an array is still the same: number of rows, number of columns, ARRAY and a name. If we stay with the example from the previous Tidbit, we would enter 4 26 ARRAY SCORES. Accessing a cell, however, has been simplfied. Where before you had to type

```
75 0 2 SCORES CELL ! or 0 2 SCORES CELL @ ( or ? )
```

to store or retrieve scores now you only need

```
75 0 2 SCORES ! or 0 2 SCORES @ (or ?)
```

And as you probably noticed, some error checking has been added also. It is rudimentary inasmuch as it only checks that the high limits for rows and columns are not exceeded (assuming that it is unlikely somebody would enter a negative value). But it will keep you from crashing your system if by chance you do make a mistake and send SCORES looking for an invalid address, i.e., beyond the boundaries of the array.

Before we go on, let me remind you that ARRAY can create a byte array by removing 2 * just before ALLOT and in the DOES> portion just before R>.

Getting back to errors, most of them occur when you forget that the first row of an array is row 0 and likewise for

columns. We can cut down on errors and at the same time facilitate access to the array if we make full use of what Forth offers. Our example SCORES was intended to record the scores of four players over a period of 26 weeks. This means row 0 for player number 1, row 1 for player 2 and so forth. The easiest way to set the row number for each player is to use constants:

```
0 CONSTANT TOM 1 CONSTANT DICK 2 CONSTANT HARRY 3 CONSTANT JOE
```

Now the first scores of the season can be recorded with:

```
150 TOM 0 SCORES ! 189 DICK 0 SCORES ! 93 HARRY 0 SCORES ! 134 JOE 0 SCORES !
```

A step in the right direction but we still have to remember to use the column parameter at a value of one less than what comes naturally and typing SCORES ! (or SCORES @ or ?) is not elegant. To take care of this lack of style we define a couple of words:

```
: SAVE 1- SCORES ! ; ( n r c --- ) : GET 1- SCORES ? ; ( r c --- )
```

Now we can use names and the real number of the week to record scores for any of the players. Tom's score of 299 in the second week of the whatever season (notice how he has improved in just one week's time!) would be entered with just

```
299 TOM 2 SAVE
```

or could be checked on with

```
TOM 2 GET 299 ok
```

and the same applies to Dick, Harry and Joe and any week of their 26-week season.

When you create an array you can not be assured that its space is alloted in a part of the memory that is "clean", i.e., all bytes are zero. To see for yourself, enter ' SCORES 212 DUMP after creating our sample array SCORES (and booting -DUMP). The first two locations should show 001A 0006 but most likely you won't find many 0000s. You can use the interactive mode and the FILL (addr cnt byte ---) command to set things right:

```
' SCORES - put starting address on stack 4 + - we don't want to wipe out the dimensions 4 26 * 2 * - calculate count (skip 2 * if a byte array) 0 FILL - execute (fill with zeros)
```

Now enter ' SCORES 212 DUMP again and see that everything has indeed been changed to zero except the locations containing the array's dimensions.

Any array - no matter how smart - which is used to store data is of no use if it can not be accessed again and again as is the case with our "scores" which we would want to have available for at least 26 weeks. The obvious solution is to keep our array on a diskette. Data from an array can be moved to a Forth screen (and back) with MOVE (or CMOVE if it is a byte array). Again, you can do it interactively:

```
' SCORES - addr1 n BLOCK - addr2 (n = screen number where to store it) 4 26 * - calculate n ( or b) 4 + - add the dimension cells MOVE - move 'em (CMOVE if byte array) UPDATE FLUSH - make it permanent
```

To retrieve the data from the screen and put it back into the array at a future date, you would first create the array (4 26 ARRAY SCORES), then reverse the order of the addresses and leave out UPDATE FLUSH:

```
n BLOCK ' SCORES 4 26 * 4 + MOVE (or CMOVE)
```

Since neither the array's dimensions nor the number of the data screen change, words like ZEROSCORES, SAVESCORES and GETSCORES could be defined to do away with the typing effort shown above. They could be placed on a screen along with ARRAY so they would be available whenever ARRAY is booted. This sounds like a good idea but it has a flaw: these words would be limited for use with only one specific array, i.e., SCORES. We have made ARRAY smart, so why not define similar smart words for this purpose?

Let's define a word which can move data from a screen into an array with a minimum of effort on our part. All that is

required of us is to furnish the appropriate number of the screen and the array's name:

```
: READ-ARRAY ( scr no.  --- ) BLOCK ( addr1 ) [COMPILE] ' ( addr2 ) DUP DUP @ ( fetch number of rows ) SWAP 2+ @ (  fetch number of columns ) * 2 * 4 + ( calculate count ) MOVE ; ( execute )
```

It is used in the form READ-ARRAY name. If we had stored our data for SCORES on screen 89, then 89 READ-ARRAY SCORES would read it into the array. We can do the same to save data from an array to a screen:

```
: SAVE-ARRAY ( scr no.  --- ) BLOCK ( this will be addr2 ) [COMPILE] ' ( addr1 ) DUP DUP @ SWAP 2+ @ * 2 * 4 +  SWAP  ROT ROT ( get addresses & count in order ) MOVE UPDATE FLUSH ;
```

As you can see, this word is patterend after READ-ARRAY so that the usage is the same: 89 SAVE-ARRAY SCORES saves the data from the array to the screen though we have to juggle the stack a bit with SWAP ROT ROT.

Finally, the shortest and easiest one to round out the set:

```
: ZERO-ARRAY [COMPILE] ' DUP DUP >R @ SWAP 2+ @ * 2 * R> 4 + SWAP 0 FILL ;
```

Since no screen data is involved, usage is ZERO-ARRAY name (like ZERO-ARRAY SCORES).

This does not exhaust the subject of arrays. But I will leave it up to you now to define a word to create arrays with more than two dimensions or to modify the above to store the screen number within the array also. These are just a couple of additional wrinkles that come to mind as I am typing this. You may think of others and with the basic tools at hand, why not try your hand at it?

RAMBLING BYTES

By "Mac" Jones

I guess my ramblings would not be very direct if I didn't choose as my opening ramblings to include Scott's message to me before the August meeting that told me he was giving up his job as President of the group. Although I have known Scott a short time as years go, he has been a good friend and a very knowledgable member of the group. Although we will miss him and his good humor in the group, I can understand his reasons for his decision. I have only been retired four years but I remember when it seemed there was just not enough time to do all the things I wanted to and get my work done also. Even after you retire this is true too! (Grin). As Scott holds a high position at his hospital I am sure he is pushed for time to spend doing a lot of the things he would like to do. Also, he works with the more advanced computers at work so it would be natural to also carry this over to the home. Hence the dropping of his trusty old 99 4/A. Scott hopes to get enough for his equipment to finance a new 9600 modem for his clone at home. I cannot fault someone for wanting sameness in equipment when you work with them at your job and also at home. We will miss you Scott, and thanks for the good job you have done for us as President.

Can you believe it's almost September? Where does the time go when you are starting down the other side of the hill, so to speak. To young folks it seems like time drags on and on, but as you grow older, it does fly! Hopefully, all of the vacation is about over and we can get down to some serious club meetings as the fall days approach. It is hard to try to get your thoughts on computing when there is so much to do outside. Even I had rather swim in the warm sun than sit at a keyboard, but you had better believe I had rather hack than mow a yard!!

There is hardly anyone that contributes to the newsletter anymore. If you could just find time to give us a few comments it would make the newsletter so much more enjoyable for you. I am sure if you found just how easy it is, you would make it a monthly thing. It doesn't have to be pages long. Just a few paragraphs about a favorite game or utility would be of interest to someone else. To get an example of what I mean, check out some of the newsletters that we exchange with each month. I have made friends with some real nice folks in England and we correspond regularly and exchange software. Bob has many good friends he has correspondance with in Australia. He is even thinking of going over and meeting them. All this because of a little computer that seemingly no one wanted anymore! To really find out just what a user group can mean to you, volunteer for a job when it comes time for elections. You won't be sorry. After all, it is only once a month that you are counted on for your part as an officer. I owe quite a bit to our group and being a part of it. If you are holding back from being an officer because you think business or vacation would make you miss several of the meetings, not to worry! I have been to meetings where only 3 members showed up with no officers at all. It didn't stop us from having a wonderful time just the same.

I see no problems as to the newsletter being sent through December. However, we will have to wait until then to see how the renewals stack up. It is up to the membership as to whether we continue our newsletter or not. I hope we can and several others have expressed the same concern.

Bill promised to give us a demo on the TI IBM CONNECTION at our September meeting and that is one meeting I would not miss if any of you do have Big Blue or it's clone at work. You can format a disk with the IBM at work and bring the work home to your TI and work at home. That sounds great to me. Although I do not have access to a clone, there are many members who do. I had heard that this was possible but never had the chance to witness it. I hope we can have a good turn-out to give Bill support on his demo. 'Til then enjoy the good TImes.

XB TUTORIALS

By Tony McGovern

I. INTRODUCTION

These Tutorials were originally written several years ago to try to improve the abysmal standard of programming apparent in magazine and User Group Newsletters, and even commercial game programs at the time. They appearsd first in the Sydney TISHUG NewsDigest and later ones in the HVYY Newsletter after the formation of that group as a separate entity. The standard of published programs written in XB has on the whole improved since then, but not as much as it should have, and as advanced users move on, new beginners take their places on the basic console/XB computers and need to learn how to use them. XB is no longer the prime language on the expanded TI-99/4a, but it is still a very expressive and powerful language, with features that have only been caught up with in recent times on newer and more pretentious computers. There does seem to be some continuing demand for tutorial material on XB, so I hope this series will still be of value to new and old programmers.

The series is intended neither as an elementary course for absolutely raw beginners nor as a reference treatise. It is meant for the interested user who is willing to put some effort into understanding how Extended Basic works in order to make best use of it, and wishes to develop a feel for how the machine actually goes about its business. Plus assorted ravings and ramblings on. This issue of the Tutorials represents a mild going over of the original files, mainly to remove some of the outdated topical material.

The aim of this series on TI Extended Basic was always to concentrate on those features which had not received due attention in User-group newsletters or commercial magazines. In fact most of the programs published in these sources up to the original time of writing had made little use of that most powerful feature of XB, the user defined sub-program, or of some other features of XB. Part of the reason for this was the practice of some authors, a practice still sometimes in evidence in magazines, of rehashing the same material for several different machines and publications, forcing the use of the lowest common denominator of Basics. Worse still were the many programs which were object lessons in how to write tangled and obscure code. A much neglected source of help 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 ? Subjects covered are

(1) User-defined sub-programs
(2) Prescan switch commands
(3) Bugs in Extended Basic
(4) Crunching program length

Initially the discussion will be restricted to things which can be done with the console and XB only. Original intentions were to cover LINKing of assembly routines but the series petered out before that. Ross Mudie has written extensively on this topic in a series of TND articles in recent years so the gap has been filled. Actually, for most game programming in pure XB with no assembler help, 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 TXB, our first major exercise in XB coding, is a factor of 10 faster in the finally released version than it was in the first pass at coding the game. Curiously it has turned out that the coding of this game is so finely fitted to the way TI XB handles things that the puck speed has to be reduced for Myarc XBII to handle it without losing track, even though it is supposed to be much faster.

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(0,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 weren't even that helpful at the time) 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 or tail remarks. 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 in which the powerful TMS-9900 CPU is hobbled by being forced to interpret another language (GPL) for an imaginary 8-bit processor.

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-99/4a console has only 256 bytes of CPU RAM for the TMS-9900 lurking in there. All we note now is that the TI does it in a very civilised fashion, LINKing by name to relocatable assembly routines. Ask your PC or Apple or C64 owning friends if their Basic supports that. The Funnelweb system supports all necessary assembly development functions with the XB module in an expanded system. TI XB contains a standard set of interface routines that support the details of linking. Unfortunately these have a great weakness in that transfers of strings between XB's VDP storage and the memory expansion is done byte by byte with a complete reset of the VDP read and write addresses for each byte instead of transferring a block at a time. The result is that assembly string handling doesn't always speed things up all that much and this has been a reason for lack of success of some XB support packages.

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 TI-99/4a 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. The lack of proper "procedures" has always been the major limitation of many BASIC dialects as a computer language. TI XB is one of the BASICs that does provide this facility. Not all BASICs, such as the GW-Basic supplied with IBM style ATs that we bought recently for the lab, are so civilised. Perhaps the suppliers of these machines don't really want or expect anyone to program their machines 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 localised 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 to be mentioned later. 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, and appears 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 from REMarks which are ignored anyway). Now this is a good way to do things as the main program is right up front there for you to inspect. One of the great annoyances about Pascal is that despite its pretensions to structured style it forces you to hide away the main program right at the end after all the procedures. 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 under that name. 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. Some recent Basics on other machines now allow subprogram variables to be specified either as static (can be very handy) or transient (which saves storage space in between CALLs), but XB is at the fully static end of the spectrum. 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 behaviour of TI Extended Basic comes from programming experience with Version 110 of XB on a TI-99/4a with 1981 title screen. Earlier Versions and consoles are not common in Australia, but TI generally seems to have taken 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 either, less by far even than it tells about console Basic.

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 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 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 variables and DEFed functions 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 does not 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.

PRINTERs AND SUCH

What worse fate can befall a person than to have a printer "crap out" for no particular reason? Even worse is one that can't be easily fixed! Anyway, for those of you with broken printers (or none at all), here is some information that might be of use to you. There was a review of the STAR NX-1000 in the March issue of MICROpendium that was certainly of interest. The NX-1000 is the latest in the line of "economy" printers offered by STAR. It is fast (144 CPS in draft mode) and versatile (both IBM and Epson compatible). It has 4 NLQ fonts and support italics in all modes as well as high resolution graphics capability. It even has "paper park" which allows you to use single sheets without removing your tractor feed paper. From there, the features go on and on: reverse and forward micro-feed, single, double and quadruple sized letters, condensed print, superscripts, subscripts, etc., etc.

The best part of the whole deal is that it can be had for $159!! Just pick up a copy of COMPUTER SHOPPER and look at the Mid-West Microperipheral ads. I ordered one from that outfit on a Monday morning and it was at my doorstep Wednesday. Amazing service!!

There was, at one time a problem with the printer EPROM. Whether it was just with a single printer or a whole batch, I'm not sure but the one to watch out for is Vn 1.5 (current Vn is 2.1). At any rate, all of the "bugs" seem to have been worked out and the NX-1000 will do just about everything except make coffee!!