# THE GUILFORD 99'ER NEWSLETTER

VOL.6 NO.10 OCT 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: Oct 3, 1989 Time: 7:30 PM. Place: Glenwood Recreation Center, 2010 S. Chapman Street.**

**Program for this meeting will be a demonstration of some XB and BASIC programming tricks and tips to help you write more compact and smoother programs. If you are interested in writing your own programs be sure to stop by!**

## RAMBLING BYTES

By "Mac"

My apologies to the members for my absent last meeting, but I had to attend the funeral of my Uncle and just couldn't make it. According to Bob, not many of you could make it either! I understand there were 2 faithful members that showed up but soon left. I got that info from a message on the ROS board from Scott who said he showed up for a goodbye hug for everyone.

As some of you know, Bob had a bad break with his printer as it went out on him while he was at the beach. He gave it to me to play with and see if I could get it working again. If you have never had your printer go out on you, just hope it never does! There are very few chips on the board that you can go to the local electrical supply house and buy. Most are dedicated chips that are special to the printer they are used in and there is no way to tell the "players" without a program. The program being a schematic diagram or electrical "road map" for your printer's circuit. I learned this by calling the STAR 800 number that was listed in the user manual Bob gave me with the printer when I picked it up. The Tech-help number, however, has been changed to a 900 number so that you have to pay $2.00 for the 1st minute and 40 or 50 cents for each additional minute.

I thought I would try the free 800 number first so after dialing it I was told that I could purchase the Tech Manual for the NP-10 for about **33** dollars. She also advised me that I could order any parts that I needed for the printer from STAR. She advised me that they were in the process of renovating and it would be the last of September before anything could be sent. I called the Tech number and was told to order the manual! How 'bout that, $2.00 to tell me to get the manual! No wonder the Japs are making a killing, they sell you a printer and when it blitzes, the charge you around **5** % of the price of the printer for a manual to fix it by! They need to put Japs in charge of the Post Office! It sure wouldn't be losing money like they claim they are now!

Talk about TI brotherhood, last Tuesday night I was watching a late show when about midnight the phone rang. It was Scott Copeland from EAR in England! You could have knocked me over with a feather. Scott said it was 5:30 AM over there and he was kind enought to call all that way to try and help me fix my printer!! Can you believe that? There are folks 50 miles

away that wouldn't do that, but Scott did. You never know where true friendship can come from, but I treasure Scott and Jo Ann's. Thanks again Scott and as you suggested, I just guess I will order the manual.

Now is the time for you parents to get back to the TI and help the kids to learn to program for their school lessons. My three girls enjoyed having the TI around while they were studying, and now, my little grandson is enjoying the same "puter" as his mom did.

I hope that October will bring the members out in force. I would like to see all members try to attend the Oct. meeting. Perhaps Bill can still give us the demo on the IBM/TI connection. We are still looking for input from members for the newsletter, how 'bout it? Well, gotta run so until the meeting on the 3rd, enjoy the good Times.

# XB TUTORIAL PART 2

## III. SUBPROGRAM PARAMETER LISTS

In the last chapter we saw how subprograms fitted into the overall workings of Extended Basic. In this chapter we are going to go into the details of writing subprograms. Most of the fiddly detail here concerns the construction of the parameter lists attached to CALL and SUB statements, and some of the little traps you can fall into.

Any information can be transmitted from the CALLing program to the CALLed subprogram via the parameter list, and anything not transmitted this way remains private for each program, with the exception of the DATA pool which is equally accessible to all. If something is mentioned in the parameter list then it is a two-way channel unless special precautions, provided for in XB, are taken. In this case the CALLing program can inform the subprogram of the value of a variable or entry in the parameter list, but not allow the CALLed program to change the value of the variable as it exists in the CALLing program. Arrays however, numeric or string, can't be protected from the follies of subprograms once their existence has been made known to the subprogram through the parameter list.

Let's for starters take a very simple but useful example, where a program needs to invoke a delay at various points. Now some BASICs (and TI LOGO) have a built-in function called WAIT. XB doesn't have this command (though a cynic might suggest that BPL gives it to you all the time whether you want it or not) so you have to program it. It can be done by a couple of CALL SOUNDs or with a FOR-NEXT loop. Let's use an empty loop to generate the delay, about 4 milliseconds each time around the loop, and place the loop in a subprogram.

```
230 CALL DELAY(200)
670 CALL DELAY(200/D)
990 CALL DELAY(T)
3000 SUB DELAY(A):: FOR I=1 TO A :: NEXT I ::SUBEND
```

This is easier to follow when editing your program then using a GOSUB, and you would need to enter the subroutine in every subprogram since GOSUBbing or GOTOing out of a subprogram is verboten. Also it's less messy than writing the delay loop every time. The example shows several different CALLs to DELAY. The first supplies a number, and when DELAY is CALLed, the corresponding variable in the SUB list, A, is set to 200. This is a particular example of the kind of CALL from line 670 where the expression 200/D is first evaluated before being passed to DELAY to be assigned to A. Variable D might for instance represent the level of difficulty in a game. The CALL from line 990 invokes a numeric variable T, and A in the subprogram is set to the value of T in the CALLing program at the time when the CALL is executed.

Nothing untoward happens to T in this example, as the DELAY subprogram does nothing to change A. Now it may not matter in this instance if T did not retain its value after the subprogram CALL. Suppose instead the delay was to be called out in seconds. Then a subprogram on the same lines DELAYSEC might go

```
230 CALL DELAYSEC(2)
990 CALL DELAYSEC(T)
4000 SUB DELAYSEC(A):: A=A0

4010 FOR I= 1 TO A :: NEXT I :: SUBEND
```

Now after DELAYSEC has been executed with the CALL from 990, T will have value 250 times its value before the CALL. This

won't be a bother if you don't use T again for its previous value. If the CALLing program specifies a numeric constant as in line 230, or a numeric expression, the change in A in the subprogram has no effect on the main program. Suppose you can't tolerate T being changed in line 990 (and this kind of thing can be a source of program bugs). You will find that XB allows for forcing T to be treated as though it were an expression, thus isolating T from alteration by the subprogram, if T is enclosed in brackets in the CALL (not SUB) list. Suppose DELAYSEC is also called from line

```
970 CALL DELAYSEC((T))
```

If this CALL in line 970 is followed by the CALL from line 990, T not having been altered in the meanwhile, the same delay will be obtained, but if the order of CALLs were reversed the second delay would be 1250 times the first. In the language of XB this is known as "passing by value" as distinct from "passing by reference". This can only be done for single variables or particular array elements, which behave like simple variables in CALL lists. Whole arrays cannot be passed by value, but only by reference. Expressions and constants can only be passed by value, and its hard to see what else could be done with them. In the example as written, a different variable name was used in the SUB, but if you remember the little experiment in the last chapter you'll see that it wouldn't make any difference if T had been used in the SUB list instead of A.

Now let's complicate things a little by flashing up a message on the bottom line of the screen during the delay interval.

```
200 CALL MESSAGE(300," YOUR TURN NOW")
270 CALL MESSAGE(T,A$)
3000 SUB MESSAGE(A,A$):: DISPLAY AT(24,1):A$

3010 FOR I=1 TO A :: NEXT I :: DISPLAY AT(24,1):""

3020 SUBEND
```

The SUB parameter list now contains a numeric variable and a string variable in that order. Any CALL to this subprogram must supply a numeric value or numeric variable reference, and a string value or string variable reference, in precisely the same order as they occur in the SUB list. In the little program segment above, line 200 passes constants by value and line 270 passes variable references. There is no reason why one cannot be by value and one by reference if so desired.

This process can be extended to any number of entries in the parameter list, provided the corresponding entries in the SUB and CALL lists match up entry by entry, numeric for numeric, string for string. The XB manual does not say so explicitly, but it appears that there is no limit apart from the usual line length problems, on the number of entries in the list. This is the only apparent difference between the parameter list in XB subprograms and the argument lists for CALL LINK("xxxxxx", , ... ) to machine code routines in XB, and Minimemory and E/A 9asics.

One little freedom associated with built in subprograms is not available with user defined subprograms. Some built-ins, such as CALL SPRITE permit a variable number of items in the CALLing list. Parameter lists in user defined subprograms must match exactly the list established by the SUB list or an error "INCORRECT ARGUMENT LIST in CALLs allow whole arrays, numeric or string, to be passed to a subprogram. Complete arrays may be passed by reference only. Individual array elements may be used as if they were simple variables and may be protected from alteration by bracketing in the CALL list. An array is indicated in the parameter list by the presence of brackets around the array index positions. Only the presence of each index need be indicated as in A(). MATCH(,,) indicates a three- dimensional array MATCH previously dimensioned as such, explicitly or implicitly. Don't leave spaces in the list. If the subprogram needs to know the dimensions of the array these must be passed separately (or as predetermined elements of the array). TI Basics are weaker than some others in that they do not permit implicit operations on an array as a whole, a very annoying deficiency.

Arrays may be DIMensioned within subprograms. This will introduce a new array name to the program, and an array or variable name from the SUB parameter list can't be used or an error message will result. In the following code the main program passes, among other things, an array SC to subprogram BOARD (perhaps a scoreboard writing routine in a game).

```
100 DIM SC(2,5) :: ....
450 CALL BOARD(P,A$(),SC(,))
4000 SUB BOARD(P,A$(),S(,)) :: DIM AY(5):: ......... ::
CALL REF(P,AY(),S(,)))
4080 SUBEND
```

5000 SUB REF(V,A(),B(,)):: .... :: SUBEND

BOARD generates internally an array AY() which is passed to another subprogram REF (maybe this resolves ties) along with SC(,), which BOARD knows as S(,), and REF in its turn as B(,) -- the same name could have used in all places. There is however no way that the main program or any subprogram whose chain of CALLs doesn't come from BOARD can know about the array AY(). This would hold equally well for any variable or array, string or numeric, first defined within BOARD and whose value has not been communicated back to the CALLing program via some other variable mentioned in BOARD's parameter list.

By following this line of reasoning you can check out the conclusion that there is no way for a subprogram whose chain of CALLs does not come through BOARD to know about array AY(). The only way around this is for AY() to be DIMensioned in the main program (even if this is its only appearance there) and the message passed down all necessary CALL-SUB chains.

This idea of DIMensioning an array only within a subprogram is particularly useful if the array is to READ its values from DATA statements and to be used in the subprogram. This could be done again from any other subprogram needing the same data, without having to pass its name up and down CALL-SUB chains. Remember that DATA statements act as a common pool from which all subprograms can READ. If the array values are the results of computations then these values must be passed through the CALL parameter lists.

For completeness note that although the XB manual has nothing to say about it, IMAGE statements for formatting PRINT output are accessible from any part of a program in the same way as DATA statements and not confined to the subprograms in which they occur as are DEF entries.

It is not necessary to have any parameters in the list at all. Subprograms used this way can be very helpful in breaking up a long program into more manageable hunks for ease of editing. We shall also see in later chapters that there can be other benefits as well.

One more XB statement for subprograms remains, the SUBEXIT. This is not strictly necessary as it is always possible to write SUBEND on a separate line and to GOTO that line if a condition calling for an abrupt exit is satisfied. Like a lot of the little luxuries of life however, it is very nice to have and makes programs much easier to read and edit. It does not replace SUBEND which is a signal to the XB pre-scan to mark the end of a subprogram. SUBEXIT merely provides a gracious and obvious exit from a subprogram (awkward in some Pascals for instance). The next chapter will demonstrate typical examples of its use.

IV. USEFUL SUBPROGRAM EXAMPLES

In the previous chapter we used as an example a DELAY subprogram which could, with a little refinement, be used to substitute for the WAIT command available in some other languages. You can extend this idea to build up for yourself a library of handy-dandy subprograms which you can use in programs to provide your own extension of the collection of subprograms that XB offers. The MERGE facility with disk based systems makes this particularly easy. See Jim Peterson's Tigercub Tips for many further examples.

For our first example let's take one of the more frustrating things that TI did in choosing the set of built-in subprograms. If you have Minimemory or E/A you know that the system keyscan routine, SCAN, built into the console ROM returns keyboard and joystick information simultaneously, while XB forces you to make separate subprogram CALLs, KEY and JOYST, to dig it out. Since these GPL routines are slow it is difficult to write a fast paced game in XB that treats keyboard and joysticks on an equal footing as is done by many cartridge games. On the other hand in games where planning and not arcade reaction is of the essence there is no reason why the player(s) should be forced to make a once-and-for-all choice and not be able to use either at any stage of the game.

The subprogrammers approach to this problem, once it realised that it can be done (and we have seen commercial XB games where the writers haven't) is to write the game using joysticks, but replacing JOYST by a user defined sub-program JOY which returns the same values as JOYST even when keys are used.

The first step in telling whether keys or joysticks are being used is to check the keys, and if none have been pressed then to check the joysticks. If a key has been pressed then its return, K, has to be processed so that the direction pads encoded in the keyboard split-scan return the corresponding JOYST value. A subprogram along the lines of the one used in TXB does just this.

```
900 SUB JOY(PL,X,Y):: CALL KEY(PL,K,ST):: IF ST=0 THEN CALL JOYST(PL,X,Y):: SUBEXIT

910 X=4*((K =4 OR K=2 OR K=15)-(K=6 OR K=3 OR K=14))

920 Y=4*((K=15 OR K=14 OR K=0)-(K=4 OR K=5 OR K=6))

930 SUBEND
```

PL is the player (left or right joystick or side of the split keyboard) number and is unaltered by the procedure. The simple-minded approach for converting K to (X,Y) values by using the XB logic operators (one of the more annoying omissions from console Basic) seems to work as well as any. The subprogram as written checks the keys first but balances this out by putting the processing load on the key return.

This is as good a time as any to sharpen your own skills by working out alternative versions of this procedure, and also by writing one for mocking up a substitute CALL KEY routine to return direction pad values even if a joystick is used.

## BEANSTALK ADVENTURE

This Adventure game had me going for several weeks. In fact I would still be groping around for the secret word to kill the giant rat in the Giants parlor if it hadn't been for Jo Ann in England. She is a wizzard at solving all adventure games and if you ever get stuck, she's the one to write to!

The game starts with you waking up in your bedroom. A mirror attests to the fact that you are Jack as you look into it.

After some words in the kitchen with your mother, you are told to go take the cow into town and sell it for money for food. You must first find the cow and then you must find a rope to use to lead the cow into town. You sell the cow and then you find a way to get her back so you can trade her for some magic beans.

Of course you will get a scolding from your mother for the trifling beans you bring home and you are told to throw them out in the garden to get rid of them. There is a way you will find to make them grow and grow they do! Right up into a beautiful blue sky.

By climbing the beanstalk, much to the dismay of your mother, you will find you are standing on a fluffy cloud overlooking a huge castle. You can enter this castle and you will find evidence of a huge being living there. There are four halls consisting of the main hall, the South hall, the great hall, and the North hall. Rooms leading off from these halls are a storage room on one side of the South hall and a kitchen off the other side of the same hall. From the kitchen you can go to a sewing room or a dining hall. From the great hall you can either go to the well or to the parlor. At the end of the North hall you will find a staircase going up to two more rooms. A wrong move here will surely get you eaten by a giant who's bedroom you stumble into. The other room upstairs will be the maid's room.

The maid's room will yield a stool. The parlor, which has a giant mouse hole, has the golden harp. In the dining hall you will find a giant table which contains a hen that lays golden eggs. The kitchen holds a large oven and a piece of cheese. The sewing room yields a large needle and thimble which will be sorely needed to complete this adventure. In the storeroom you will find a smelly fish barrel.

There will be a need to return to earth a couple of times to trade items for other items needed to win the adventure. I am sure you will enjoy this adventure as I did. You will be surprised what the 4th treasure will turn out to be! Mc

## TONY'S CORNER

Confessions of a Small-Time User. by Tony Kleen. Guilford (NC) Users Group.

Well shucks, we've got another 1149-4A system for sale. A prior member of the Winston-Salem group notified me this month that his TI is collecting dust, and dying of lonliness. He has a console and PEB; with RS232, disk controller, SDSD disk, and 32K cards. Cartridges include TI-WRITER and MULTIPLAN. Also included is an OKIDATA m92 printer.

One can contact Larry Youngren at (W)760 4070 or (R)761-3147. He has had an ad in the Journal, and was wanting 300 for all of it. I believe he is open for negotiation, though.

Hope ya'all enjoyed the TI-BASE presentation last month. I certainly enjoyed getting up front and letting the package demonstrate itself. The HELP and TUTOR presentations are to be commended. We've been having some rainy evenings again, so I was able to write a few dbase programs. I have listed them at the end of this article. I know that I learn by example better than I can learn from a reference manual, so I thought I might provide a couple TI-BASE examples and explanations.

The first program is a command file called SETUP/C. This is the file automatically called for execution by TI-BASE as soon as one has booted and entered the current date. This shows that one can immediately call your own master menu to the screen. The first line will SET the TALK mode OFF. If one does not set this off, each directive line is displayed prior to it's execution. This makes for an untidy screen when one is trying to display a master menu. The next directive will SET the CURSOR speed to '2'. This is the recommended speed for the TI. A speed of 10 is recommended for the Geneve. I haven't read anywhere as to the allowable cursor speeds. I can't see where anyone would much care, either..

Prior to the use of any variable in a directive, one must define that variable to the system. I would assume that one would want to keep the variable name at a minimum number of characters, ie. one or two. One only has a certain amount of dynamic memory, 2200 bytes with E/A or XB cartridges, and 6300 with the M/M cartridge. Anyone know for sure?!? Now then, let's describe the variable definition directive. One is declaring a LOCAL variable named 'L0' as a 'C'haracter type, with a length of '1' byte. The program is using this label variable to determine the operator's screen selection.

The statements between the WHILE and the ENDWHILE (3rd directive from the end) will be executed as long as the expression resolves to a "true" condition. In this case, WHILE our local variable 'L0' is not equal to ( ie. <> ) the character string "0", the program will execute the directives that will () paint the master menu on the screen, () retrieve the operator's selection, and () execute the operator's selection.

Now I've already given away what the program does. Painting the screen is the CLEAR directive (which CLEAR's all information from the screen) and all the following WRITE directives. The READSTRING directive allows for operator data entry of his/her selection. The program interogates the selection variable 'L0' and executes the selection via the DOCASE/CASE structures. In our example, all the valid L0 values are listed, followed by a "DO C?" directive, then followed by a BREAK directive. The DO directive requests the execution of another command file, ie our case, command files C0, C?, etc. The directive to BREAK off the CASE structures is issued to advance to the ENDCASE directive.

The only way to get to the CLEAR screen, and RETURN to the calling module; is to have the operator enter a "0" at the selection prompt.

The command program C1/C is similar to SETUP/C. The program continues to () paint the screen, () accept the operator's selection, and () act upon the selction; WHILE the operator does not enter a selection "0". You will notice one diference, in that one has a CASE L0="0" within the DOCASE and WHILE structures. This is improper 'structured' coding, but saves on execution time, ie. is faster to execute than placing the RETURN statement at the end of the command file.

The final command program C11/C simply ()paints the screen with column header lines, and () lists the LBLOWNER file, in it's entirety to the screen. Just an example of SCROLL'ing; a handy directive!!

Some notes I thought of while writing. The first three lines of each command file is documentation, ie., comment lines. I like to know where the file resides (DiSK), what is the disk's name, what is the file's name, and when was this file last updated. The extention '/C' stands for (C)ommand. When one wants to execute a command file, one ignores the extention, eg. DO C1, not C1/C!

See ya'all at the next meeting. Goo'day.

```
* ==================== =====
===
* DSK.LABELS.SETUP/C   89/08
/15.
* ==================== =====
===
SET TALK   OFF
SET CURSOR 2
LOCAL L0 C 1
WHILE L0 <> "0"
  CLEAR
  WRITE 02,13,"LABEL PROCESS
ING"
  WRITE 04,03,"1-Append owne
r"
  WRITE 05,03,"2-Append labe
l"
  WRITE 06,03,"3-Edit owner"
  WRITE 07,03,"4-Edit label"
  WRITE 08,03,"5-Delete owne
r/label(s)"
  WRITE 09,03,"6-Delete labe
l"
  WRITE 10,03,"7-Print owner
's labels"
  WRITE 11,03,"8-Print membe
r label(s)"
  WRITE 12,03,"9-Define labe
l's format"
  WRITE 13,03,"0-EXIT"
  WRITE 15,03," -SELECTION"
  READSTRING 15,03 L0
  DOCASE
    CASE L0="8"
      DO C8
      BREAK
    CASE L0="9"
      DO C9
      BREAK
    CASE L0="1"
      DO C1
      BREAK
    CASE L0="2"
      DO C2
      BREAK
    CASE L0="3"
      DO C3
      BREAK
    CASE L0="4"
      DO C4
      BREAK
    CASE L0="5"
      DO C5
      BREAK
    CASE L0="6"
      DO C6
      BREAK
    CASE L0="7"
      DO C7
      BREAK
    ENDCASE
  ENDWHILE
CLEAR
RETURN
* ==================== =====
===
* ==================== =====
===
* DSK.LABELS.C1/C      89/08
/15.
* ==================== =====
===
WHILE L0 <> "0"
  CLEAR
  WRITE 02,11,"APPEND OWNER
PROCESS"
  WRITE 05,03,"1-Display Own
er/Purpose"
  WRITE 06,03,"2-Display Own
er/History"
  WRITE 07,03,"3-Proceed w/A
ppend"
  WRITE 07,22 "process"
  WRITE 08,03,"0-EXIT"
  WRITE 10,03," -SELECTION"
  READSTRING 10,03 L0
  DOCASE
    CASE L0="0"
      REPLACE L0 WITH " "
      RETURN
      BREAK
    CASE L0="1"
      DO C11
      BREAK
    CASE L0="2"
      DO C12
      BREAK
    CASE L0="3"
      DO C13
      BREAK
    ENDCASE
  ENDWHILE
* ==================== =====
===
* ==================== =====
===
* DSK.LABELS.C11/C     89/08
/11.
* ==================== =====
===
CLEAR
WRITE 01,02 "0"
WRITE 02,02 "w   LBLOWNER db
ase"
WRITE 03,02 "n"
WRITE 04,02 "e"
WRITE 05,02 "r Purpose"
WRITE 06,02 "- ------------
-----------"
WRITE 07,02 "- ------------
-----------"
*
USE LBLOWNER
WHILE .NOT. (EOF)
  SCROLL 20,07
  WRITE 07,02 LBLOWNER
  WRITE 07,04 LBLPURPOSE
  MOVE
  ENDWHILE
CLOSE
RETURN
* ==================== =====
===
^^^^^^^^^^^^^^^^^^^^^^^^
```