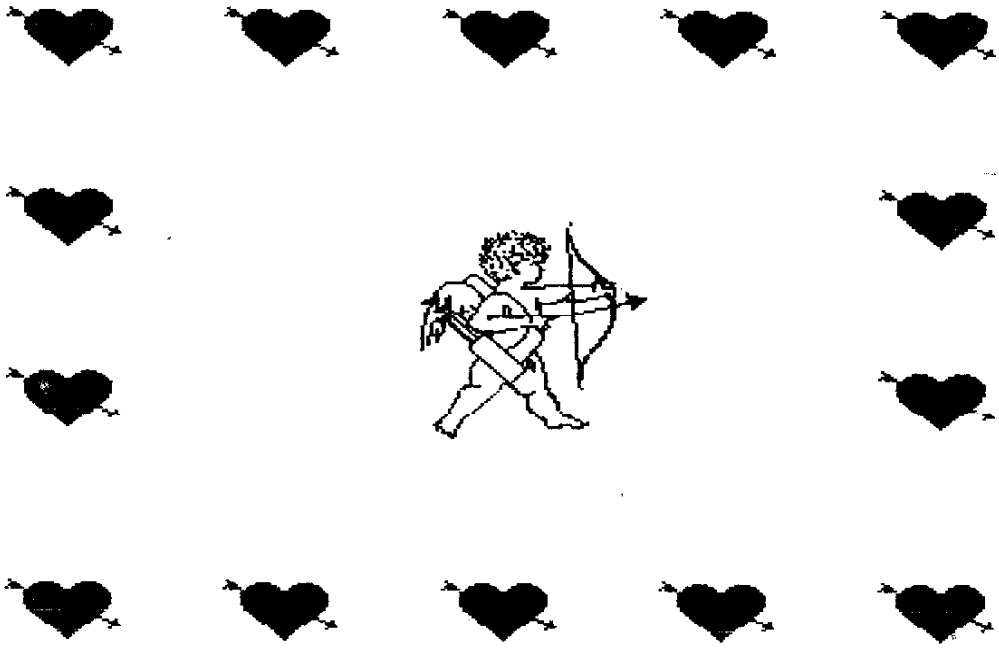
GULLFORD 99'ERS

NEWSLETTER



VOLUME 7 NUMBER 2 FEBRUARY 1990

BOB CARMANY, PRESIDENT (855-1538)
EMMETT HUGHES, V. PRESIDENT (584-5108)
MAC JONES, SEC/TREASURER (288-4280)
GEORGE VON SETH, EDITOR (292-2035)
BILL WOODRUFF, PGM LIBRARY (228-1892)
BBS: (919) 621-2623 (ROS)

DUES ARE $12.00 PER FAMILY PER YEAR
MAKE CHECKS PAYABLE TO: L. F. JONES
3202 CANTERBURY DRIVE
GREENSBORO, NC 27408

SOFTWARE LIBRARY IS FOR DUES PAYING MEMBERS ONLY

GULLFORD 99'ERS

OUR NEXT MEETING

DATE: Feb 6, 1990 Time: 7:30 PM. Place: Glenwood Recreation Center, 2010 S. Chapman Street.

Program for this meeting will be a demonstration of the TI-WRITER Utilities package from the Chicago Users Group. Bill Woodruff will conduct a workshop on how to use this series of programs to make writing documents easier.

MINUTES

The January 2 meeting of Guilford 99er Users' Group was held at the Glenwood Recreation Center on Chapman Street in Greensboro, N.C. There were 6 members and one visitor present.

The meeting was called to order at 7:45 P.M. by President Bob Carmany. A short business meeting was held and consisted of the minutes being read and a copy of the Treasury report given each member present.

OLD BUSINESS:
There was no old business brought up.

NEW BUSINESS:
Pres. Carmany asked for a demo for February's meeting and Bill Woodruff said he would demo the TI/Writer Utilities he has been working with.

Pres. Carmany also reminded members that the Feb. Newsletter would be the last for non-paying members. He also mentioned the new version (4.21) of Funnelweb that he has. Also his program (Program Writer Ver. 4.0) is now out and as soon as he returnes from Australia, he will upload it to the BBS.

After the short meeting, goodies and drink was made avaiable to members. I would like to thank each one attending for the fellowship and good food.

The party was over at 9:30 P.M.

Respectfully Submitted,
L. F. "Mac" Jones, Sec./Treas.
Guilford 99er Users' Group
Greensboro, N.C. 27408

PRES PEEKS

This is my last column before my trip to view the "Wonders Downunder" first hand. After saving for a year for this adventure, I'm looking furward to "getting amongst 'em" in Newcastle. Newcastle is the home of the Hunter Valley UG and all of the brilliant programmers who are numbered among its members. I'll be spending a little over a week in the Hunter Valley region with plans to visit with many of the friends that I've made over the years. It should be interesting to talk to the likes of Tony and Will McGovern (Funnelweb), Ron Kleinschafer (OED cartridge and hardware hacker), Richard Terry (Forth programmer par excellence), Joe Wright (Geneology program), and the rest of my 'mates'. I don't think that there is a higher concentration of truly brilliant programmers in any other UG anywhere in the world. There will be a full account of my "adventures" in the April issue of the newsletter and I expect that there will be a rather large text file on ROS as well. Who knows, I might be some "goodies" brought back from Oz as well.

We are still soliciting articles for the newsletter. This seems to be a rather universal problem among TI Users groups. Filling the pages of a newsletter is not as easy as it might seem and we would like to get as diverse a range of articles as

possible. The articles don't have to be professional quality or even typewritten. We will accept articles in virtually any format --handwritten, on disk, or printed out. The topics can be anything related to the TI --reviews, programming tips, software and hardware modifications, or what uses your TI serves. Questions about the TI are also welcome. We will try to find the answers to any questions that you have about how to use a particular program or just about anything else that you want to know. Send any material to: George von Seth, 608 Candlewood Dr, Greensboro, NC 27403.

We are going to try to have an interesting graphic coversheet for the newsletter for the remainder of the year. The coversheets are created with PAGEPRO. The program is easy to use and creating an attractive graphic application is both quick and straightforward. Besides, you can use TI-Artist character fonts and instances to put together whatever you want.

RAMBLING BYTES

By "Mac"

Well gang, at least 6 of us enjoyed the companionship and good food last Jan. 2. I was surprised that only that few showed up, but I guess it was a little too close to the "big weekend" for some. Anyhow, you missed a real treat.

As you probably noticed in the President's remarks, there will be no more Newsletter for you who have not paid the dues. As you remember, the constitution states that there will be 30 days grace period from December 31st and now has passed. I am surprised to say the least that we will be losing so many members. To date, only Bob Carmany, George von Seth, Mrs. Jerry Jones, Tony Kleen, Bill Woodruff and yours truly have paid for another year. I will be glad for any of you that wish to continue the Newsletter and Library privileges to send a check for $12.00 to L.F. Jones at 3202 Canterbury Dr. in Greensboro, N.C. 27408. As we have no checking account for the group, I must cash the check in order for us to get the cash.

As I wasn't able to speak to a lot of you at the meeting, I hope Santa was good to you and stuck you something real TI'ish in your stocking. Unfortunately he must have lost the letter I sent him asking for the Horizon ramdisk for it was left at someone else's home instead of mine. Oh well, maybe someday!

Most of you know by now that Bob is leaving for Australia in February so if you are wanting to give him any messages for Tony and Will and the othere mates over there, you had better be at the meeting. Bob will unfortunately miss the March meet but you will probably see the results of his trip on the Groundstar BBS as soon as he lands home.

Remember, if you are having problems with TI/Writer, be sure to be at the February meeting and pick up on the good information Bill Woodruff will be demoing for us then. Until then, enjoy the good TImes.

XB TUTORIALS PART 6

By Tony McGovern

VIII XB PROGRAM SCRUNCHING

Well, where do we go in the future ? So far the series has had a detailed look at user SUBprograms and the ACCEPT AT statement, the two most powerful features of TI's Extended Basic, and also at the prescan switch commands lurking in the V110 manual addendum. For the next few sessions we will continue with topics which are of immediate relevance to console-only users, namely squeezing programs to fit in memory, and extracting maximum speed from XB. I can see no point, and have even less interest in writing about, say, SOUND or SPRITEs from the very beginning, as these are fairly well documented in the manuals and the subject of many books and articles. That isn't to say that subtleties in using them won't come up from time to time.

Enough raving on for now and on to the real business. Let's now look at how to face up to that 'MEMORY FULL' message. This even comes up when you have memory expansion with a total of 48K of RAM in various guises. Programs always seem to end up needing more memory than is available! I do feel some unease in discussing this topic as many of the things that are done in compacting programs can only be regarded as poor program practice otherwise, as they make the code obscure and difficult to modify or develop further. The other great trade off that must be considered when scrunching programs is speed of execution. Given an equal level of skill in program writing, coding for speed usually results in a longer program than would otherwise be written. Perhaps the easiest example to see is unrolling of a short loop which is repeated a fixed number of times. A

FOR-NEXT loop gives compact code but carries a penalty of the loop overhead which could be avoided by writing out the contents of the loop the appropriate number of times. The subject of coding for speed will be taken up in detail in later Tutorials, and speed sacrifices with compacted code will only be noted in passing. The richer the language the more opportunities there are to optimize code one way or the other. Console Basic offers many fewer ways to do this than does XB and is much less fun.

At what stage should you bother trying to make your code compact ? Remember that XB can only OLD or RUN one program at a time, so apart from loading time from cassette, or disk space, there is no reason at all to scrunch a program that runs in the smallest memory it is intended to run on. Most users with disks now have the 32K memory expansion, so this means the bare console. Minimem Basic programs to store in the module's RAM are the only ones you have real incentive to make smaller still. Unless you know from the start that you are going to run short of space because of large arrays of numbers, or a need for maximum string storage room, be expansive -- document your program thoroughly with REMs, use lots of SUBprograms, use obvious explanatory names for variables, avoid reusing variables for unrelated uses ..... and then you run out of room.

Now first of all a program has to be short enough to load. This is purely a function of program length. Next it has to be able to complete prescan when RUN. For prescan to succeed there must be enough room left over after the prescan for variable pointer and subprogram tables to be set up, and room set aside for numeric values, at 8 bytes per number. String variables are not assigned space until it is actually required, so it is possible for a program to crash later because it can't find enough room for strings. The well known hiccupping of long Basic programs occurs while Basic scratches around to reclaim string space when it has run out of new space. XB does it too, but it is a lot faster at 'garbage collection'. Now let's look at how to squeeze programs in, starting with things that affect the program length only.

The most obvious thing to do is to remove REMs from your program. I would suggest that this be left till later in the development process as you put them there in the first place to help. At the least keep some for now. If you have been following earlier Tutorial advice to use lots of clearly named subprograms then you don't need many REMs. For the same reasons you should not abbreviate subprogram names beyond recognition at this stage. Basic as an interpreted language, where the source code is also the run-time code, has this problem that commentary and explanation are not eliminated by a compiler or assembler and compete for memory space with the executing program. One way round the problem is to restore REMs to a file copy after intensive development is over, even if it does make it too long to RUN. The REMS can always be removed later.

Now it's time to look at what makes an XB program as long as it is. To get started let's look at two very short programs to clear the screen.

```
100 CALL CLEAR
```

Before entering this clean up the machine with NEW and SIZE it. Then enter this program and SIZE it again. The difference will be the length of the program 13928-13914 = 14 . I will mostly quote SIZEs on the basis of a console only machine for simplicity, but there are some interesting differences. With memory expansion XB lists high memory and stack separately, and ignores low memory altogether. XB stores the program and numeric variables in high memory (24K), while the stack - 12K of VDP memory - contains variable descriptions, subprogram details, PABs, and the string storage space. This ALL has to fit in 14K of VDP RAM with XB/console only. Console Basic doesn't use memory expansion for Basic at all. Now try a second program which does almost the same thing

```
100 DISPLAY ERASE ALL
```

and SIZE it. Only 9 bytes now! Although the LIST of this second program is longer, the computer thinks it is shorter. Consult your XB manual p40 where you will find all three words DISPLAY, ERASE, ALL are listed as reserved words, as is CALL but not CLEAR. Reserved words are treated differently -- when you enter the line they are recognised and "tokenized" as one byte symbols with ASCII values above 127. 'CLEAR' takes 7 bytes, one the token for a string without quotes, one for a length byte, and 5 for the string itself. Why use tokens ? For one thing it shortens the program length, and also makes it easier for the interpreter to recognize them when the program is running. XB's range of tokens is very limited and built-in subprograms are the way XB gets around this.

Now you don't have to take my word for this. If you have an expanded system you can write programs using CALL PEEK to explore stored programs, or better still use the E/A DEBUG (reassembled as uncompressed object code so the XB loader can handle it) for a quicker look. With console XB you can at best get an indirect insight by entering <CTRL+various keys> in a REM statement and LISTing that. Be careful, you can crash the computer in ways wondrous to behold that way. Someone forgot to tell the computer not to try to turn token values back into reserved words when LISTing REMs. Ever notice when writing file specifications that keywords that do extra duty elsewhere LIST with the extra space, but the others do not. EASY-BUG in

Minimem also allows you to look directly into VDP RAM or cartridge RAM to see Basic programs in their internal state.

In TI Basics, unlike those which store programs as ASCII files, the line number is always stored as a 2 byte integer, and it makes no difference to program length to use line #1 or line #10000. Try various line numbers in one of the examples above. If you are peeking around in the program, don't expect to find the line number at the start of its program line. It is in a separate table below the program, and each 4 byte entry has the line number followed by the location of the line itself. The line # table is sorted into order, but new or edited lines are always added to the lower address end of the program block. The program lines themselves are preceded by a length byte and terminated by a null (>00) byte. I won't go into it here but you can use this general information to interpret the various time delays when you edit a line or enter a new line.

From this you can see that there is a 6 byte overhead associated with every new line number. Now enter the program lines above as lines #100 and #200 and SIZE. Next combine them as a single line

```
100 CALL CLEAR :: DISPLAY ERASE ALL
```

and SIZE again. There is a saving of 5 bytes. The reserved word "::" has cost 1 byte, but 6 bytes have been saved by having one line fewer. Now if you scrunch a 500 line console Basic style of program into 200 XB multi-statement lines you have gained 1500 bytes. Of course you can't do this to every line because line numbers, as well as being line editor markers, are also where GOTOs and GOSUBs go, so you will usually end up with a few short lines you can't condense. FOR-NEXT loops work perfectly well within or across multi-statement lines. The use of prescan switch commands is costly because you end up with !@P+ and SUBFND on separate lines at the end of each subprogram so treated. Still, it's usually worth doing even though a long program may have several hundred bytes tied up in prescan switching. In desperation at the end you can always remove prescan switches starting with the shortest subprograms.

How much room does a variable take up ? Take a simple numeric variable. There are 8 bytes for the radix-100 floating point form that both TI Basics use for all numbers (they even do 1+1 to 14 significant figures every time - another reason they are slow). Next the interpreter has to be able find where this value is stored so there's 2 bytes for a pointer to the value, and 2 more to point to the name associated with this value. Further it has to record the nature of the variable, whether it is numeric or string, simple or array, DEFed or normal. Also in a Basic language which allows long variable names a length record is also likely, though not absolutely necessary. All told there is a practical minimum of 14 bytes of overhead for every simple numeric variable.

As I have noted in other connections in this series, TI in its self-defeating secretive way, never explicitly specified the details. TI Basic is most likely highly consistent in this from model to model, because any console can be called on to work with separate E/A or Minimem Basic support utilities such as NUMREF. On the other hand each XB module contains its own set of support utilities, and only has to be internally self consistent. There is information in TI's published data (XB, E/A, Technical manuals), giving details of VDP stack entries built by the E/A CALL LINK with some hints as to changes for the XB version. So to use XB LINKs at this level of detail you have to work by implication. Now it is done from time to time, but TI does not seem to have guaranteed explicitly to programmers that such procedures would work with all XB modules, or that the LINK stack entries are similar to internal table entries. Most likely they do and are. Only TI knows for sure. Then again TI lost big while Apple and IBM make lots of money being more open about their machines, though Apple seems to be developing more secretive ways as it gets older and more arrogant.

Time for some little program experiments again. Enter the miniscule program

```
100 A=0
```

Before you do anything else work out how many bytes this uses. The answer is 11. In accepting the line the editor has already figured 'A' for a variable (because it starts with an allowable character) and not a reserved word and it is represented exactly as it occurs, no token involved. On the other hand it doesn't yet care that '0' is meant to be a number and treats it as an unquoted string. If it isn't an honest number, say 2N, it will only find out later when it RUNs and tries to convert it to a floating point number.

SIZE the program, then RUN it and SIZE again. XB does not reset everything until you have made an editing change, as you know from debugging efforts after BREAKing (fctn-4) program execution. At this stage you get more information from an expanded system, which will show 8 bytes of memory used and 9 bytes of stack. Now repeat the process with a longer variable name. The length is reflected both in the original program length and in the stack used. The stack usage is 2 bytes plus the

variable's name length more than the minimum we figured out before. Most likely the 2 bytes are for a linked list structure to help table searching, and there is a symbol table entry of the variable name. Now turn off your expansion system and be like everybody else with console only, and repeat the above. Now you will find the increase over the program length is always the 14 bytes we figured earlier no matter how long the variable name is. Now try

```
100 A23456789012345,A23456789012345=0
```

Still 14 bytes RUNtime overhead! Change the second A to a B to make a distinct variable name, 15 bytes long. Only another 14 bytes overhead ! So how come ? Maybe it's doing without the full word list link and squeezing things up a bit, but what about the symbol table ? The only consistent conclusion is that it doesn't have one as such, but points to the first location of the variable name in the program as located by the prescan. Read the Tutorial on prescans again. XB always searches in VDP RAM for variable names even if the program itself and the numeric values pointed to are stored in expansion memory.

If you wanted to make a faster interpreted Basic, you would, in the prescan, replace all variable names by some token plus a storage pointer to eliminate table searches. Which is just what TI claim in their Software Development Handbook to have done with the Basic for their 990 minicomputers. Unfortunately they failed to make an honest machine of the 99/4.

The next tutorial will continue with the principles of program scrunching, getting more into the program writing end of things.

TIPS AND TRICKS

The following is a compilation of tips, tricks and general information gathered from issues of TI*MES.

This first item is a short bit of source code that can be assembled and loaded into XB with CALL LOAD. When a program breaks for any reason, it will auto-load "DSK1.LOAD"

```
DEF CHECK
CHECK MOVB @>8344,>8344
      JEQ NORUN
      B *R11
NORUN CLR @>83C4
      LI R1,>6372
      MOVB R1,@>9C02
      SWPB R1
      MOVB R1,@>9C02
      B @>006A
      AORG >83C4
      DATA CHECK
      END
```

"INFOCOM adventures are not all FULLY logical-there is a RANDOM element in some of them which means that sometime you die and sometimes you live... and an interesting command to type into your INFOCOM adventures is $VE--try it!! There is a $RAND command in Lurking Horror which expects a number before/after it, not sure what it does but I think it may determine the path when you come to a random choice. HITCHHIKER has a total vocabulary of 969 words-have you found them all yet? KILL ADAMS? Some odd commands include XYXXY and ZZMGCK- the latter may just be an end of file dummy. SUSPENDED has a vocabulary of 680 words, but you can complete it with just 35, that's reall overkill!!"

ROM CART PORT

This is a text file discussion of the ROM cartridge port for the TI-99/4A. It represents information I have been able to obtain from various references. Cartridge programs must operate from >6000 to >7FFF. When the computer is RESET or turned on, the power up routine looks for a Header or Control block at location >6000 in the cartridge port. This contol block establishes the linkage into your cartridge program and allows you to have multiple entry points. Here is an example contol block used to provide one entry point;

```
0000 AA01       DATA    >AA01     6000      ID FOR BOOT
0002 0000       DATA    >0000     6002
0004 0000       DATA    >0000     6004
0006 000C       DATA    CHAIN     6006      ADDRESS OF MENU LIST
0008 0000       DATA    >0000     6008
000A 0000       DATA    >0000     600A
000C 0000 CHAIN DATA    >0000     6010      CHAIN POINTER
000E 0020       DATA    SLOAD     6012      ENTRY POINT
0010   0F       BYTE    SLOAD-$-1 6014      LENGTH OF MENU TEXT
0011   54       TEXT    'CARTRIDGE NAME'
0020 0460 SLOAD B       START
0022 092E
```

Let's examine the control block. If the TI operating system finds >AA at >6000 it knows a cartride is plugged in the port. The next byte must be a >01 at location >6001. This informs the operating system that the code in the cartridge is executable machine language. Other codes are used for GROM, but that's another discussion. The data at location >6002 - >6005 is zero. Location >6006 must contain a word pointer to a list which identifies the menu text and associated entry point when that item is selected. This location usually contains a >600C. Locations >6008 - >6009 must be zero. The chain list at >600C contains the following:

Bytes 1 2 = chain pointer to the next menu list - or 0000 is this is the last list in the chain.

Bytes 3 4 = entry point associated with this menu selection.

Byte 5 = length of the menu text.

Bytes 6 - N = Menu Text - this is displayed on main menu. Craig Miller's newsletter has additional information on the power up routine for the computer. Remember all dynamic data must be in RAM usually in the >8300 area. This area is used for registers plus VDP RAM is used for variable storage. Cartridges cannot REFerence any lable or routine outside the cartridge. This means the cartridge program must provide it's own VSBW, VSBR, VMBW, and VMBR routines which are normally loaded from the Editor Assembler cartridge. Examples of what these routines look like may be found in the Tombstone City game or Craig Millers newsletter. Armed with this information, it possible to disassemble code to see how the program works. Hope you find this information useful. See ya around the boards...Mack McCormick 74206,1522.