THE GUILFORD 99'ER NEWSLETTER

VOL.5 NO.3 MARCH 1988

Larry Spohn, President
Janice Snider, Vice President
Mack Jones, Secretary/Treasurer
Herman Geschwind, Program Library
BBS: (919)274-5760(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, Editor)

OUR NEXT MEETING

DATE: March 1, 1988. TIME: 7:30 PM PLACE: Glenwood Recreation Center 2010 S. Chapman Street.

Program for this meeting will be a roundtable discussion of what is available for the TI99/4A five years after "Black Friday". Bring in any information concerning software, hardware or repair facilities. We hope to compile a software/hardware directory and a list of local repair agencies.

MINUTES

The February 2, meeting of the Guilford 99er Users' Group was called to order by President Larry Spohn at 7:50 PM. The January minutes were read and accepted as read.

Old Business: There was no old business discussed.

New Business: President Spohn announced that he may pretty likely be leaving North Carolina, but will let us know in plenty of time so a new Pres. can be elected to fill out his term. Larry also told of a project by the Winston Salem group, of which he is also a member, of building Super Carts. It was also brought up that Data Bio techs has now started shipping the Grand Ram, but prices are higher than origionally advertised due to an increase in chip prices. The Sect. was asked for the number of members who has not paid their 1988 dues, and was told the number to be eleven.

The program was conducted by Bob Carmany on the use of the Mini Memory. Bob gave an excellent demo on the ability of the Mini Memory to store files, much like a ram disk. At the end of his demo, Bob also gave members information on the German version of the GPL assembler that we have received from Germany. The text, which is over 200 pages, has been translated into English, however most of the commands in the assembler are still in German.

It was suggested by the Secretary that the club purchase a 32K expanded memory for club use, and was given the permission to spend $30.00 of the club money for same. (Sec. note: a kit has been since bought for $15.00.)

There was an extended discussion on GPL and other subjects pretaining to the TI and its uses.

The meeting was adjurned at 9:30 PM.

Respectfully submitted, L.F. "Mack" Jones, Sect./Tres.

PRES/POKES

Thoughts of Departing President
By Larry Spohn

Regretably this is my first, last and only column as your president. Unfortunately for us, but fortunately for my family, I have accepted a science writing position with the Albuquerque Tribune. I will be leaving this month to begin primarily covering the development of Star War technologies at government labs in New Mexico. My family will join me later. Although the chances are slim, it is possible I may be back in town during one of the monthly meetings and if I possibly can will stop by.

However, clearly you need a new president as it would be quite a drive for me every month from Albuquerque.

I certainly would recommend Janice Snyder to you as my successor, although I'm sure that will make her uncomfortable. Yet as a relative newcomer, she brings a much needed fresh approach that I think will invigorate the group and keep its feet on the ground.

I wish I was going to be around for what may be a pivotal year for the group. Of course, in our history, what year hasn't been pivotal. But clearly, given Mac's membership report at the last meeting, our ranks continue to dwindle. As president, I thought that the primary objective of the coming year would be membership and I would suggest (easier said than done) that any reasonable effort to bolster the ranks should be tried. We've discussed computer fairs to direct mail appeals. Perhaps the best approaches are word of mouth, notices in the newspapers "things to do" columns and posting notices in computer stores, schools, libraries and recreation centers.

Certainly, given Herman's and Bob's reports last meeting the user groups are integral to the machine's and its users' future and increasingly they have much, much to offer. I continue to be impressed with the international efforts being made to fully utilize this outstanding computer. I can tell you that whereever I am, I will do my best to participate in and support these efforts.

I do plan to continue my turtle-paced efforts in exploring LOGO although the move probably will preoccupy me over the next several months. I also hope to keep in touch with both the Guilford and Forsyth groups and hopefully to fullfill (at some point) my promise to at least start up a LOGO program section in the two libraries.

I also would reiterate that you will do yourselves well to establish a continuing liason between the two groups. You have much to offer each other, even if you never merge. The Guilford groups strengths are its extensive international connections, telecommuncations and Forth programming expertise, while Forysth users have been exploring hardware alterations, extensive use of data bases, unique programming efforts and the compatibility of the TI with the DOS and IBM compatible systems.

Certainly there is plenty of opportunity here for at least some occassional joint programs over the year and these two might be useful in attracting new members across the Triad. Another idea I would suggest that might improve communication within the groups and between them, would be an alphabetical listing of your members in your newsletters, including phone numbers, addresses, computer expertise and interests. Such a listing might help bridge some gaps and certainly would help the novices, newcomers (and old-timers as well) with specific problems. Let's face it, while the opportunities for this machine continue to blossom the number of us staying tuned is declining. We ought to be as supportive as possible.

Well, before I get too excited, perhaps I better just say BYE and write off into the sunset. Take care all and good luck.

ATRAX TRACKS

By Bob Carmany

Last month we took a general view of FUNNELWEB 4.0 with a brief discussion of some of the more interesting additions and improvements. Since we touched on using the CONFIG program to set up a menu, let's take a look at what one might look like. As an example, I'm going to use the system that I have configured for myself.

The first three options available on the menu are the pre-set paths that load, respectively: 1)TI-Writer, 2) Editor-Assembler, and 3) Return to XB. I have configured my system to offer two distinct sets of programs -- the next five selections are a series of programs used for XB program development.

Option 4 loads XLATE which is an XB program that converts D/V 80 files into runnable programs. That allows me to write programs with a word processer like the F'WEB editor and take advantage of the superior editing capabilities. I simply use a template of line numbers and go from there. When I am finished, I run the resultant text file through XLATE and it is SAVEd as a runnable program. XLATE has been altered so that instead of just QUITing, it RUNs "DSK1.LOAD" and thus restarts F'WEB.

Option 5 loads another XB program called COMPRESSXB. This program removes all of the REM statements from a program and combines some of the program lines to make a more compact, faster running program. It has also be altered to reload F'WEB when it is finished.

Option 6 loads FILE/READ, a program that I developed to read (and optionally print) ANY file except for those in program image format. It is menu-driven and has some brief on-screen instructions with it. It has also been altered to re-enter F'WEB when it is finished. Although it is written in XB, it loads almost instantly.

Option 7 loads SYSTEX by Barry Boone. Systex is a utility for combining XB and A/L programs into a hybrid program that will load and run in XB. The result is like the F'WEB LOAD program. It has also been altered to reload F'WEB when it it finished.

Option 8 loads PRESCANIT by J. Peter Hoddie. PRESCANIT is an XB program that will process a program and turn off and turn on the XB prescan function, replace variables and otherwise "crunch" an XB program to allow it to load almost instantaneously. I used it on my FILE/READ program. It has also bee altered to re-enter F'WEB.

Option 9 loads Will McGovern's DISKHACKER. It is a hybrid XB and A/L program that allows the user to examine disk contents far beyond what is available with normal sector editors. It allows you to view (but not alter) protection schemes and look at some of the "strange-sectored" disks that appear from time to time.

This takes care of the first half of the 18 options that the user has available on the first F'WEB screen. The second half, designated A through I, contain a variety of other programs that I use frequently.

Option A loads John Birdwell's DISK UTILITES 4.0. These files are loaded as an A/L program image file load and contain a series of sector editing utilities and disk manager utilities. The package comes in handy when you have made some alterations and want to move the files to another disk under a different name. It is more powerful that the DISKO program furnished with F'WEB. I use it mainly for "heavy duty" sector editing and use DISKO for the quick jobs. It is the program that I use to change the screen prompts in F'WEB.

Option B loads Barry Boone's TRACKCOPY. It is an excellent track copier that will copy virtually any disk. It loads as a program image file(s) and comes in handy when you want to make a back-up copy of a protected disk that DM1000 won't handle.

Option C loads ARCHIVER II Vn 2.4. This just-released version loads from the GPL environment as a program image file. It will pack and compress files (all of our BBS stuff is ARCHIVED) which will save you a bunch of time when uploading and downloading programs from a BBS. It will also save you a vast amount of space on a disk --- I recently compacted the contents of 6 SSSD disk into two! It makes sending programs across the country (or the world) much cheaper also. It is now fully F'WEB re-entrant and searches for UTIL1 when you exercise that option.

Option D loads an XB Dis/assembler that is really a "budget model" for those short dis-assemblies. It doesn't have a lot of the conveniences of some the longer programs but it IS efficient. It has also been altered to re-enter F'WEB when it is finished.

Option E loads PGM CONVERT. This program is used to convert D/F 80 A/L files into program image. The result is a savings in space and the program image A/L programs also load and run faster. It loads as a "Load and Run" D/F 80 file with the F'WEB assembly loaders.

Option F loads another A/L program image file called SBUG. This is basically a de-bugging tool for assembler programs. It comes in handy when you have a bug in something you have put together with the F'WEB E/A option and it just won't run properly.

Option G loads WYCOVE 4TH. It provides a reminder to insert the disk in the correct drive and then loads the tow program image files that make up the WYCOVE 4TH kernel.

Option H loads TI-FORTH. Since I "dabble" in both of them, I wanted to be able to load either of them without having to resort to another loader.

Option I loads an XB program called GEMINI that is just a short program to set up my printer in the font and character style that I want. It has also been altered to re-enter F'WEB when it is finished. Since your printer will retain whatever options you send it until it is either reset or turned off, this program can be used to get different font styles when using the F'WEB Editor.

That takes us through the 18 options available on the initial menu screen. You can see that F'WEB can be configured to load just about any type of program imaginable as a menu option. On my copy, I have loaded XB programs, E/A program image files, GPL environment program image files, and E/A object code (Load and Run). All of them load quite neatly and easily with F'WEB. The fact is, there are VERY few programs that can't be loaded as a menu option with one of the F'WEB loaders.

There is no reason why you couldn't set up your own initial menu of programs that you use most frequently as menu options on the Main Menu and have them instantly available when F'WEB boots. All you have to do is follow the instructions for using CONFIG and then follow the prompts as they appear on the screen.

From here, we are going to examine the TI-Writer option (#1 on the main menu) and all of the options that it presents including the Users List option that gives you 8 more A/L programs immediately available. That is where we are going to go next month ---- detailed TI-Writer information and its supporting menu. The following month, we will look at option #2 on the Main Menu --- the E/A options and the intermediate menu screen that it presents. That will include all of the A/L loaders and using the Editor/Assembler as well. In the mean time, read your F'WEB docs and "play around" with a back-up copy using the CONFIG file. If any questions come up, send them in to the UG mailing address or bring them up at our meetings. 'Til next month . . .

XB TUTORIAL

By Tony McGovern

WELCOME to the first Extended Basic Tutorial from Funnelweb Farm to appear in the new HUNTER VALLEY 99 NEWS. This will be a continuation of the series of articles which appeared earlier in the TISHUG Newsdigest, but the HV99 NEWS will now be the primary source. The series will go on in the same vein as before, intended neither as an elementary course for 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, news items, and ramblings on.

Some copies, and disk-files, of previous Tutorials will be available at HV99 meetings for newcomers or ex-TISHUGers who don't have all the previous ones in their Sydney file. The Tutorials may even show up in User Group News letters around the world. I have seen one in the TI*MES from the UK. Now HV99 members will be the first to see them, for whatever that is worth. If other User Groups wish to reprint these Tutorials, please get in touch with me, either directly or via the HV99 group, so that you can get a corrected and updated version on disk in the user group spirit of exchange. I usually do an edit on the file after it has appeared in print to correct the little goofs which hide so cunningly before printing, and sometimes to clarify, correct, or extend what had appeared in print. The printed version has not always been precisely what was on disk either.

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. Please let me know of areas you would like covered. My policy so far has been to concentrate on those parts of XB which are especially powerful, not already included in console Basic, not well documented, and not as widely appreciated as they should be. The next few tutorials will be on getting the most into and out of the machine while using XB. On the other hand 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.

There has been a gap of a few months in appearance of Tutorials, mainly due to pressure of work. The time spent on the TI-99 has been almost entirely devoted to Assembly language programming, much of it in association with XB, and this will provide some real substance for future HV99 News articles, either in this series or separately. One of these projects has been to get TI-Writer running from XB. Why do that ? Well... TI have always been good guys in that most serious disk software can be backed up on disk as often as needed, but they were of course relying on the infamous cartridge GROMs for protection and exclusion of others. Now cartridges are a lot less fragile than disks, but they can die too. So I don't want ever to have to suffer Imagic Australia's less than impressive service and/or rapacious pricing policy if our TI-Writer module ever claps out. May save some module swapping on occasion too. Yes, we do have a spare XB module!

Of late I have been working with Microsoft Basic and Turbo Pascal on CP/M machines with Z-80 processor in science laboratory applications. I must say that the more I see of Microsoft Basic the more I regard it as a cancer that should have been eradicated years ago when computers grew up to have more than 8K of memory. It is only now with their Apple Macintosh version, judging that from reading magazine hype, that they have at last surpassed the level of expressiveness that XB had years ago. TI did a lot of things to screw up this machine, some of historical origin, some quite intentional, but it takes coming from the engineered TI-99/4a file and device handling system to CP/M to make you realize how weak and primitive CP/M is in this area. On the other hand Turbo Pascal almost makes that Swiss straight-jacket feel comfortable, and even CP/M doesn't seem so bad with Turbo. An excellent product at a realistic price that makes one realize that pirates are only the minor league of brigands in the software business. We can only dream that someone will bring out a native code Pascal anywhere near as good as Turbo for our machine. The TI P-code version is most unattractive at Imagic's past and present exorbitant price (eight times that of Turbo). Now that more compact consumer type products are replacing the massive PE box, P-code cards will fade into oblivion. Maybe I'll be proved wrong but I get the feeling that Imagic Aust. is the sort of outfit that would rather use things like that as hard-fill in a swamp than let committed users have them at a realistic price.

It is also another example of how TI had this death-wish to hobble the most powerful micro-processor in any home micro available here, with interpreted languages. Shed a tear for TI-99 Basics with their two layers of interpretation (Basic and GPL), on top of working indirectly from the byte oriented VDP memory and GROMs.

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 >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 C/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 SUBEND 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,A2345678
9012345=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.

That should be plenty to chew on for this inaugural issue of the HV99 News. The next Tutorial will continue with the principles of program scrunching, getting more into the program writing end of things.

LOGO

By Larry Spohn

Little BIG LOGO

While I have been praising LOGO, virtually without reservation over the last several months, you ought to be aware if you haven't discovered it on your own, that LOGO has limitations.

Among these are its foreign or unorthodox programming freedom, which if you been used to other languages can take some getting used to. I enjoyed your surprised looks when I told you that the main calling program of the Christmas Story program was just seven lines long and that I didn't matter where it was in the program as a whole.

Second and despite the above example, by comparision with other languages LOGO can seem inefficient, that is it usually takes more LOGO code to accomplish what some other languages do with less code. And, at least in TI LOGO, you aren't likely to set any program speed records although I've yet to test that argument language by language.

Third, the previous limitation is compounded by limited memory. This is particularly cumbersome in the TI LOGO configuration because--as near as I can tell--there is no provision for accessing other storage devices, such as a disk or ram disk, directly from a program.

Fourth, I know of no way to output LOGO program "results" to the printer. LOGO does have a procedure to print the programs themselves on a printer, but not the results (answers, pictures, designs). While I suspect that this printing procedure was designed for memory reasons or to make LOGO easiest for youngsters to use, I consider this the most serious application drawback of an otherwise delightful, free-spirited language.

There may be some tricks I'm not aware of that might ameliorate these limitations, particularly this last and most depressing block to creativity. If you know or can think of any please enlighten us. Otherwise, we LOGO lovers may have to wait for some experienced Robin Hood to liberate the cartridge and modify TI LOGO to eliminate these shortcomings and reveal LOGO's maximum power.

There IS one trick I would like to pass along that I learned from George Browne and Ken Hedrick of the Forsyth 99'er's

Group. I had been complaining about the problem of converting LOGO programs into ASCII for publicaton in the newsletters. TI LOGO does have a third device option in its structured Save/Load format, which will accept and I presume dump to the RS232 card for modem output. This is fine if the intended party has a modem and a TI LOGO cartridge. Tony Kleen, Forysth's "99 LINES" editor has the former but not the later. This presents a monumental road block, presuming that a transmitted TI LOGO program is received by the other TI-99/4A as a program and not an ASCII file. Tony would have no way to output the program without the LOGO cartridge, which as I mentioned at least does allow programs to be printed by your printer.

Enter Ken and George. They suggested that when LOGO prompts for the printer device that I enter the disk drive designation, ie, "DSK1. 2 or 3." I did and it worked. The program is sent to the disk in as Dis/Var 80 file, readable by TI-Writer. Thank you Ken and George. Now if only someone could suggest how we might reconfigure the ASCII file into a TI LOGO program!

The lesson illustrates a couple of points, one of which has been most often made in the two Triad groups by Herman Geschwind of Greensboro, and Roy West, of Lexington: without the user groups we are lost. It might never have occurred to me to try the option which was second nature to Ken and George. They may not be into LOGO as much as I am, but obviously they know a lot more about the 99/4A and computers generally than do I.

The other point is that LOGO is such a "new" language to most of us--includi., me--that there may well be various ways to get around the image limitations it has been saddled with, as well as those I mentioned herein.

Meanwhile, my previous praises for LOGO stand and I urge you to do some LOGOing.

FORTH TUTORIAL

By Lutz Winkler

FORTH TO YOU, TOO ! SESSION 1

Introduction

According to our source there are quite a few people out there who got the TI-FORTH disk and documentation when TI made them available to user groups. But not very many do much with it. Why? Well, the TI manual is not a tutorial, it assumes that you know something about FORTH. Though packed with useful information there are no "HOW TO" instructions for the beginner. We will try to get you started from the very beginning. Hopefully we'll strike a happy medium, somewhere in between teaching and providing info that's useful to you.

What is FORTH?

There was much hype when it was made available, some of it was much overdone, but Forth is faster than BASIC. Yet some of the advantages of Forth will not immediately be evident to a beginner. FORTH is a TIL (Threaded Interpretive Language) and it'll be hard for you to believe that there is no GOTO command. If that is hard to swallow, there is more : It uses RPN or post-fix notation (RPN = Reverse Polish Notation). In other words, it's not 2 + 2 that equals 4 but 2 2 + . We'll find out more as we go along, for now let's just say that FORTH is very powerful, quite a bit faster than BASIC, compact, but perhaps more difficult to learn than BASIC. As a matter of fact, knowing BASIC may make it harder on you, because you'll be thinking BASIC until you get the hang of FORTH.

Getting started.

Before you do anything with your FORTH disk, get out a DISK MANAGER and make a backup copy. Do all your work and experimenting with this copy unless you are prepared to get a new FORTH disk. Now plug in the EDITOR/ ASSEMBLER, opt for 3 (LOAD AND RUN) and enter DSK1.FORTH. After a moment the screen shows "BOOTING..." which is soon replaced by a menu. These are the LOAD options. For right now you need to concern yourself with only 2 of them: the normal or the 64-column editors. Your choice will depend on several factors: 1) your eyesight, 2) your monitor, and 3) how well you have adapted to using 'windows'. So jump right in and enter -64SUPPORT. After your disk drive is through you will see a tiny 'ok', meaning the 64 column editor has been booted. To see what your screen will look like type 34 EDIT <enter>. If you can read what is displayed on your screen, you'll want to stay with -64SUPPORT. If it's hard on your eyes, settle for the 40 column editor. To get an idea what it looks like, hit FUNCT-9(ESCAPE), then enter TEXT COLD. FORTH will re-boot and when it is done, enter -EDITOR. (From now on, 'enter' will mean to type in the word followed by the ENTER key.) Again enter 34 EDIT to see what your 40 column editor looks like.

Programming in Forth consists of editing SCREENS, such as that number 34 screen you called up for editing. But we are not ready for that, yet. Hit ESCAPE (F-9) and enter FLUSH and do this: Make yourself an overlay strip so you can edit easily. Keys and their functions are explained on page 5, chapter 3, of the TI-FORTH manual. Now here is another thing you might want to find out right now: a display color that suits you. Since you are still in the so-called 'interactive' mode of FORTH (no program is running) you can enter this ditty :

```
: SEE 252 22 DO I DUP . 7 VWTR
KEY 2 = IF ABORT ENDIF LOOP ;
```

After you get the 'ok', type SEE. Don't worry if you can't read anything, at times the FG and BG colors match and there's nothing to be read, keep hitting any key and the colors will change. When you see a combination which gives you a good screen display, write down the last number (bottom of the screen) and continue to step through the loop (or exit via FUNCT-4).

You have accomplished 2 things:

1) you know which editor you'll want to use
2) you have chosen a screen color

*** END SESSION 1 ***