Guilford 99'ers Newsletter

Merry Christmas

Volume 6 Number 12 December 1989

Emmett Hughes, Vice President (584-5108)
George von Seth, Editor (292-2035)
Mac Jones, Sec/Treasurer (288-4280)
Herman Geschwind, Pgm/Lib (288-0015)
BBS: (919) 274 7000 (opus)
ROS: (919) 621-2623

Dues are $12.00 Per Family, Per Year
Send Check To: 3202 Canterbury Drive
Greensboro, N.C. 27408

Software Library is for dues paying members o

MERRY CHRISTMAS
AND A
HAPPY NEW YEAR !

BOB CARMANY
1504 LARSON ST.
GREENSBORO, N.C. 27407

Guilford 99

OUR NEXT MEETING

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

There is no formal program this month. Bring your favorite music programs and other "sounds of the seasons" to share with the rest of the Users' Group. It is a good chance to visit, renew old friendships and enjoy the Christmas spirit.

MINUTES

The November 7th meeting of the Guilford 99er Users' Group met Tuesday at the Glenwood Recreation Center on Chapman Street in Greensboro, N.C. There were 7 members present.

Due to the absence of all Officers except the Secretary, the meeting was called to order at 7:40 P.M. by the Secretary.

OLD BUSINESS:

a. The secretary was asked if the new part had arrived for the Macinker and since it had arrived, Bill Woodruff made the request to re-ink 2 ribbons that he had brought in. Bill donated $1.00 to the Club for it's use. Bob Carmany also donated $1.00 for his use of the inker.

b. Herman Geschwind donated $10.00 to the Club for using the IBM program for some downloading of TI files which he said would be cheaper than using Genie to do the same thing.

NEW BUSINESS:

a. There were no new nominations from the floor as to the choice of officers for the coming year. The new officers will take over at the January meeting.

b. The Secretary asked members what they would like to do as to Christmas goodies, but there was not much interest shown for any type of party.

c. Members were advised that Ellen Kramer has sent us 6 disks of programs for use and Herman suggested that they be uploaded to ROS. Thanks Ellen.

d. It was decided by present members to wait until December to auction off the Compute! book donated by Herman for auction.

The program was given by Bill Woodruff which was on the IBM/TI Connection. The cart plugs into the grom port and requires a DSDD controller card and disk drive to work. The disk must be formatted on an IBM or Clone since the TI format will not work. The transfer is done in ASCII format. Bill, who uses both computers in his college teachings, gave a good demo on just how one could use the program to one's advantage if owning both types of computers or if using an IBM at work and needing to do homework on a TI. A very good and interesting program and many thanks to Bill.

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

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

RAMBLING BYTES

By "Mac"

I guess by the time you get this you will be into a good Thanksgiving bird or already enjoyed one, depending on when I receive the master copy from the Editor. Anyhow, we have a lot to be thankful for Folks. I hope you can find time to think about just how much you do have that some countrys don't.

Well, I made a mistake I sometimes do when I re-printed the 1-800 numbers that I put in the November newsletter. I

didn't check out what I passed on! The other day I called a bunch of the 800 numbers and thankfully, I used the phone instead of the modem! Most of them were no longer in use and the ones that were in use said they either didn't have a BBS or never had one! Some answered with a growl and must have been bothered before by someone calling for a BBS. Sorry 'bout that guys and gals. Next time I borrow something from another letter, I will make sure and check it out first. This is not to say that at the time it was printed that it was not correct, but in a years time, anything can and usually does, happen.

It was great to find a video tape program on ROS the other day and I quickly downloaded it as I have been trying to write my own program for that very thing for some time. I probably don't have as many tapes as most of you do but I have 42 and when you have to pull each one and look at the little yellow "stick-on" paper I use to tell what is on each one, it can be a problem. The program was written by Bob Pomicter of Hackettstown, NJ, and, like me, loves to write for free. It is a good program but has one short coming and that is, you cannot go back to a tape and edit it after it has been entered. I wish there was a way to do this as sometime I do tape a program over an old one that has served me for a few years. Bob says you can go into TI Writer and pull up the database, but it is in a D/F 80 format. If you make any corrections, you can not download back to disk directly as that would make it a D/V 80 file and the program crashes. I know, I did it!!! After talking to Bob Carmany about this, he told me just to PF (print file) to a D/F file name and it would save it in the D/F 80 format that I needed. This I haven't tried to do as yet, but I will the first time I need to go back to a tape and change it.

On the print out, it takes a little longer than if it was in assembly but I have no place to go right now so time don't bother me that much. It prints an alphabetic letter and then prints all information in lines beneath that letter. I think it is a great way of keeping up with what you have on tapes. Thanks Bob for a good program. If anyone figures out a way to add the feature of going back to a tape for this program, I sure would like to hear from you. Or does anyone even read this? Oh well..

I would like to thank Ellen Kramer for the disks she so graciously sent for us to share with her. Ellen, who is our member from New Jersey, says she met Barry Traver at the CPUG fair she attended in October and gave her the low- down on his uploading the files to Compuserve. It's nice to have members like Ellen who contribute their time and money to see a group prosper.

I hope Mother Nature painted a prettier picture in your local than she did in the Piedmont this fall. Ours' is a bummer. Just a few maples showed their colors but most gave out with a burnt brown before plunging into the yards.

Until the December meet, keep hacking and enjoying the good Times. See you there the 5th.

Ho, ho, hum....

How would "A Visit from St. Nicholas" sound if a government agency had written it?

Several years ago, Bob Chambers, then of WKEW in Greensboro, translated Clement Moore's famed Christmas poem into bureaucratic jargon. Here's how it begins:

"Twas the nocturnal segment of the diurnal period, preceding the annual yuletide celebration, and throughout our place of residence..."

("TWAS THE NIGHT BEFORE CHRISTMAS, WHEN ALL THROUGH THE HOUSE...)

"Kinetic activity was not in evidence, among the possessors of this potential, including that species of domestic rodent known as Mus musculus." (NOT A CREATURE WAS STIRRING-NOT EVEN A MOUSE.)

"Hosiery was meticulously suspended from the forward edge of the wood burning caloric apparatus..."

(THE STOCKINGS WERE HUNG BY THE CHIMNEY WITH CARE...)

"Pursuant to our anticipatory pleasure, regarding an imminent visitation from an eccentric philanthropist, among whose folkloric appellations is the honorific title of St. Nicholas."

(IN HOPES THAT ST. NICHOLAS SOON WOULD BE THERE.)

And so on...aren't we glad Clement Moore got to it first!!?

XB TUTORIALS PART 4

By Tony Mcgovern

V. XB STYLE WITH SUBPROGRAMS

Let's now stand back a bit and look at the best way to construct XB edifices. Assume at this stage that we are in the process of developing a program, but not yet to the point where scrunching program length has become important. The first thing to note is that by giving the subprograms good descriptive names you have already gone a long way to making your program self-explanatory.

How big should individual subprograms be allowed to get ? After all, one of the reasons for using them is to break up big programs into manageable hunks. We will use the term 'line' to refer to a multi-statement XB line identified by a line number. My own prejudice is that, except in special circumstances, subprograms should be no more than about 10 lines long, and mostly rather less than that. What makes an exceptional circumstance? An obvious one is in title blocks, like that in SIMPLIST which was left as an almost bare stub. A full version would provide graphics and advice screens, which can be tediously long to write, but contain very little in the way of branching decisions or variable assignments. Another example is where a familiar routine, that already works, is used with little variation as in COLIST where the disk directory routine from the Disk Manual is incorporated as a subprogram with only minor changes. In any such situation where long subprograms are justified, the lists of parameters passed will be short or non-existent.

The other extreme is short one or two liners which are frequently CALLed for small special tasks, more or less your own customized extension of the built-in set of subprograms. In the middle there are middle-length subprograms with extensive parameter lists and the logic which carries the burden of program flow.

Some subprograms may be CALLed only once from within another subprogram but are of value in making your code easier to read and modify. These are associated with the branching of program flow by means of IF..THEN..ELSE statements. In either TI BASIC or XB, FOR-NEXT loops may extend indefinitely with NEXT acting as delimiter. Unfortunately in extending BASIC to XB, TI did not provide an "ENDIF" statement as in TI-FORTH, but only the 'endif' implied by the end of a XB line. This means that any alternative actions determined by the IF.. condition have to fit within that XB line or involve a GOTO somewhere else unless the usual simple drop-through to the next line is enough. The XB manual already explicitly forbids inclusion of FOR..NEXT loops within IF..THEN..ELSE statements. No doubt you are already used to getting around this little deficiency by placing the looping code in a subroutine and using a GOSUB. Subprograms can be used instead, following THEN and ELSE to give more complex alternative possibilities, but still staying within the confines of a single line with a minimum of leaping about with GOTOs.

This brings us to the subject of the 'dreaded GOTO'. A great deal of heat, and not necessarily much light, has been expended on this subject. It is after all just another statement available in many languages, and has perfectly predictable immediate consequences. The real objection is that it leaves no trace as to where the program "came from". At the machine code level, jumps enable the computer to do more than just chomp along a single track of instructions. The question is whether it is help or hindrance in high level languages, and whether other ways of controlling program flow can replace its explicit use to advantage. TI-FORTH does without it, but that most procedural of languages, TI-LOGO, still finds it useful. Pascal tries to do without it. What we do have is XB, and XB can't do without GOTOs. If anything should be considered as reprehensible in a high-level language, it is any need to provide PEEK and POKE.

The great weakness of GOTO as a language element is that it is so readily abused, because undisciplined use makes the program code inefficient and hard for people to follow. The genuine message from 'structured programming' ideas is not that BASIC is bad for having GOTOs, but that most BASICs (TI console Basic is typical) make it necessary for the programmer to exercise real restraint if terrible tangles of GOTOs are to be avoided.

Once you use XB subprograms to chop up a program into small hunks, then you have automatically eliminated great leaps around with GOTOs. All you need then is to remember the comments on using subprogram CALLs as statements in IF..THEN..ELSE and take a little care in laying out the logic flow, you will find it very much easier to debug or develop programs. Backwards GOTOs over more than one or two lines of code, or any forward GOTOs at all, should only occur under the most regular of logical layouts, as in SUB BASICLINE in the SIMPLIST example. Single recursive lines such as in line 620 of SIMPLIST are very effective. It's a pity that the designers of XB didn't add the "MYSELF" function as in TI-FORTH to enhance such constructions.

One last little matter before we go on to other topics. Many languages with local procedures also allow specification of global variables, accessible from any part of the program. XB does not allow for separate global variables, and it can be quite tiresome when a parameter defined at the end of one subprogram chain is only needed at the end of another chain, and has to be passed all the way up and down in parameter lists. A way around this is to use the static value feature of XB subprograms.

```
3000 SUB PAGELENGTH(A,B):: IF A THEN C=B ELSE B=C
3010 SUBEND
```

If the write flag is set as CALL PAGELENGTH(1,66) the value 66 is stored in the subprogram local variable C, while CALL PAGELENGTH(0,PL) will retrieve that value into PL. This is clumsier than having global variables, but is also more protected from unwanted interference. XB does not enforce any hierarchy of subprogram levels, so PAGELENGTH can be written to, or read from, at any level in the program. The example is for one parameter only, but is easily extended.

Using subprograms does carry some overhead expenses, both in the size of the program and in the time taken for XB to do a CALL as distinct from a simpler GOSUB. Unless you are absolutely desperate for bytes, the benefits of subprograms outweigh that price and they should always be used liberally in the early stages of program development. The speed argument is a mixed one, as in a long program the extra time cost of a subprogram CALL can be more than outweighed by the savings in time for the interpreter because it has only a short local list of variable names to search instead of a much larger global list.

VI. PRE-SCAN SWITCH COMMANDS

The little supplementary booklet that comes with the current Version 110 of Extended Basic introduces a new pair of reserved words, !@P+ and !@P-. These have the form of a tail remark (XB manual p38) and so are ignored entirely by the earlier V.100 of XB. If the XB interpreter finds an exclamation mark ! outside any DATA string or string enclosed by quotes, it treats the rest of that line as though it were a REM statement. The V.110 interpreter has the added ability to recognize this pair of words beginning with ! as being distinct from normal tail remarks when used as a single word statement. Their use is allowed only at the end of a line so that V.100 just ignores them, not creating any incompatibility problems between versions, something that TI was always conscientious about. TI then couldn't let these commands actually do anything! So why are they there ?

The XB manual addendum, p7, tells the story. These switch commands allow you to control the operation of the pre-scan through the program by the interpreter -- that agonizing time interval after RUN is entered before the program starts executing. The interpreter is grinding its way through your program, byte by byte, ignoring only the messages in DATA, REMs and tail remarks. Other than these there is nothing that it can afford to ignore until it has actually looked at it. The pre-scan sets up the storage areas and lookup procedures for variables, arrays, data, sub-programs and DEFs used by the interpreter as the program runs. Of course once it has set aside space for a variable and its lookup linkages, then it doesn't need to do it again or even to have to decide it has already fixed it up earlier. The pre-scan switch commands allow the programmer, from a superior vantage point, to turn the pre-scan off and on throughout the program so that it only looks at what it really needs to look at to do its job.

What does the programmer gain by going to all this extra trouble? The most obvious result is a reduction of pre-scan time. This can be significant in long programs. The 6 to 7 seconds for TXB, a 12K program, may still seem long but beats 4 times that. In a later chapter we will see how it can be used to fine tune run time behaviour as well. What price does the programmer pay for these benefits? The necessary penalty is the memory space taken by the extra statements. The hidden penalties, incurred while writing programs, are the inscrutable bugs that may be introduced into the code and the loss of some program checking during pre-scan such as FOR-NEXT nesting.

Let's work our way through the XB manual's prescriptions. Some of these help give insight into the way XB conducts its affairs. My experience is that some of the restrictions need not be followed strictly as laid down, as long as the essential spirit is observed, while some are absolute, and others are in between. These last are the ones where it is possible to imagine another version of XB doing things differently while still being according to the book. This is always the danger in using unspecified properties or "undocumented features". It is not such a problem with XB since TI pulled the plug on the 99/4a and made XB a language as dead as Latin. In retrospect this last statement is no longer quite true, and though people continue to use the original XB there have been enhanced alternatives, still GPL based using various GROM simulation schemes. There have also been utility programs to do automatically some of tasks we are discussing here and more. What has been sorely lacking has been any published exegesis of just how XB goes about its business internally. I have never seen any source code for XB, either original TI or reconstructed. I don't know whether TI source has leaked out, or even if it still exists, but if it has it is being closely held, and the writers of XB processing utilities do not seem to have felt any urge to share around the details of what they found.

(1) DATA statements :-

The pre-scan locates the first DATA statement and sets XB's data pointer for the first READ operation to use. If the first DATA is skipped in the pre-scan, then RESTORE must be invoked before the first READ to set the data pointer correctly. If this is done, the XB manual's advice can be ignored.

(2) Variables :-

Each variable must be scanned once, otherwise XB won't have it in its linked list of pointers to names and storage locations. This can be the source of some truly evil program bugs, where a syntax error message results from a line of code which looks perfectly correct. The reason can be that injudicious positioning of pre-scan switch commands has left the interpreter with something that should be a variable, but can't be located as such. Being a non-variable is a much worse fate than merely being set to zero.

OPTION BASE 1 affects how storage is allocated and normally precedes any array references. It hidden from the pre-scan by !@P- then the default 0 will apply.

The manual says that the first occurrence of any variable or array must be included in the pre-scan. This would seem to be necessary for arrays, in the DIM statement, unless you are using the default (no DIM) dimensioning. Simple variables can be pre-scanned anywhere as long as it's at least once. Try the little sample program

```
100 CALL CLEAR :: !@P-
200 I=1 :: PRINT I
300 !@P+
400 I=2
```

Run this program and there will be no problems. Delete line 400 and see what happens. Now you will have a syntax error in a line that by itself is perfectly correct.

(3) Sub-programs :-

The XB manual recommends that the first CALL to any sub-program be included in the pre-scan. It would appear that if the first CALL to a user defined sub-program occurs after its own SUB (from within a later sub-program) then the necessary inclusion of the SUB and SUBEND markers suffices.

Built-in sub-programs of course do not have associated SUB statements, so a CALL must be included in the pre-scan if the program is to run normally. Try this example.

```
100 FOR I=1 TO 1000 :: !@P-
200 CALL SCREEN(12)
300 !@P+
400 NEXT I
500 SUB ANYTHING :: CALL SCREEN(3):: SUBEND
```

This will run even though SCREEN is pre-scanned only in a subprogram. Delete line #500 and it will crash if you are running XB with the 32K memory expansion. In VDP RAM (console only) it still executes but only at about 1/3 the speed.

What happens if an array is referenced in the parameter list of a sub-program, but not dimensioned until a later sub-program? If you recall the discussion on passing arrays by reference, you won't be surprised to find that XB is smart enough to hold over assigning space for the array until it comes across a genuine program reference. Try this little example

```
100 CALL SECOND
200 SUB FIRST(A());: PRINT A(20):: SUBEND
300 SUB SECOND :: !@P-
400 DIM A(20):: CALL FIRST(A())
500 !@P+
```

```
600 SUBEND
```

This program crashes with a syntax error in 400 in SECOND. Now delete the pre-scan commands and the program will run. If you further delete DIM A(20):: in line 400 the program will crash in 200 with a subscript error.

(4) DEF,SUB and SUBEND :-

Do as the book says. XB needs these in the pre-scan to set things up correctly.

The pre-scan switch doesn't have much effect unless the program is of substantial size, so it isn't worth worrying about too much in the early stages of a program's development beyond being prepared for the possibility. The XB manual supplement (p10) shows how all variable and sub-program declarations may be gathered together to minimize the range of the pre-scan, by using a GOTO to jump over the list to the first executable statement. This can be gotten away with since XB does not do a complete check for correct syntax until it comes to execute the line. This is the only virtue one can ascribe to XB's failure to reject all invalid lines at entry time. The same technique can be used within a sub-program, and I have found it very convenient for this same GOTO to reserve a hiding place in which to tuck away the subroutines accessed by GOSUBs within the sub program.

TOKENS

By Bob Carmany

A month or two ago in MICROpendium, J.L. Stern came up with an interesting little program called LINESAVER that extracted and saved chunks of XB code to be used as subroutines. One of the areas that was briefly covered in the article and program was the idea of CHR$ token values saved in "crunch" format in a D/V 163 MERGE file. Of course, there have been partial lists of the CHR$ tokens published from time to time and they have been used to create programs that write other programs. The idea of creating a program to write code as a D/V 163 file that can be MERGEd into memory or with existing programs leads to almost endless possibilities. If only there was a list of what the CHR$ values above CHR$(127) represented in "crunch" form! With that idea in mind, I set out to see what I could do about providing a usable list.

To be of any real value, the final list almost had to be in a D/V 80 format that could be manipulated by TI-Writer or F'WEB so it could be easily printed. I discovered quite quickly that characters above CHR$(127) aren't printable as token values in a D/V 80 file. So, we had to trick the computer into thinking that it was writing a bona fide MERGE program and go from there. The final result wasn't the best programming effort but here it is anyway.

```
100 CALL CLEAR :: ON WARNING NEXT

110 DISPLAY AT(2,2):"CHARACTER AND TOKEN LISTER" :: DISPLAY AT(4,7):
"By R.M.  Car many"

120 DISPLAY AT(14,1):"FILENAME: DSK1." :: ACCEPT AT(14,14)BEEP VALIDATE(UALPHA,
DIGIT,".")SIZE(-12):FN$

130 OPEN #1:"DSK"&FN$,VARIABLE 163
```

The first four program lines are very simple. Basically, they clear the screen and present the program title, and display the author's attribution. Next, a prompt appears for the output filename and the appropriate disk file is opened. No problem so far!

```
140 A$=" = " :: DISPLAY AT(14,1):RPT$(" ",28)

150 FOR TOK=32 TO 255 STEP 1

160 DISPLAY AT(13,12):TOK :: IF TOK<127 THEN DISPLAY AT(11,6):
"LISTING CHARACTER " ELSE DISPLAY AT(11,5):"LISTING TOKEN VALUE"
```

The next line sets A$ equal to "=" with a space on either side and then blanks out the line used to enter the disk file

name. Line 150 sets the loop parameters from CHR$(32) to CHR$(255). By changing these values, one could print out any range of character or token values. Line 160 checks to see what the current value of TOK (the character value and line number) and displays the appropriate message on the screen. If the value is less than 127 (a printable character), the "CHARACTER" message is displayed otherwise the "TOKEN" message appears. The current character value is also displayed.

```
170 PRINT #1:CHR$(INT(TOK/256))&CHR$(TOK 256*INT(TOK/256))&A$&CHR$(TOK)&CHR
$(0)

180 NEXT TOK
```

Line 170 is the algorithm for printing the information to the disk file (see J.L. Stern's article in the Jan '89 MICROpendium). In this case, the line number and the CHR$ value are the same because we are using the same value. The line number is followed by A$ and then the actual value of the character string. The computer has been tricked into creating a "program" starting at line number 32 and extending to line 255 with the corresponding character strings for each. Line 180 increments the loop to the next character and line number.

```
190 PRINT #1:CHR$(255)&CHR$(255)

200 CLOSE #1 :: DISPLAY AT(17,1):"Type 'NEW' to clear memory " :: DISPLAY AT
(19 ,1):"and MERGE DSK"&FN$&" and"

210 DISPLAY AT(21,1):"then LIST 'DSKx.filename'" :: DISPLAY AT(23,1):"to create
a D/V 80 file" :: FOR DELAY=1 TO 1000 :: NEXT DELAY :: END
```

The next line sends the end of file marker when the loop limit is reached and line 200 physically closes the file and begins the display of the final instructions. If you follow these instructions, you will have a D/V 80 file that can be manipulated and printed with TI-Writer or F'WEB. You will find, however, that some of the lines have "garbage" immediately following the "=". You can delete it and print the CHR$ value and "=".

If you wish, you can look at the intermediate D/V 163 file before you LIST it to disk and get some idea of what your D/V 80 file will look like. Here is the entire program with checksums after processing with the CHECKSUM program that appeared in MICROpendium some time ago:

```
100 CALL CLEAR :: ON WARNING NEXT !042

110 DISPLAY AT(2,2):"CHARACTER AND TOKEN LISTER" :: DISPLAY AT(4,7):
"By R.M. Carmany" !207

120 DISPLAY AT(14,1):"FILENAME: DSK1." :: ACCEPT AT(14,14)BEEP VALIDATE(UALPHA,
DIGIT,".")SIZE(-12):FN$ !018

130 OPEN #1:"DSK"&FN$,VARIABLE 163 !114

140 A$=" = " :: DISPLAY AT(14,1):RPT$(" ",28)!153

150 FOR TOK=32 TO 255 STEP 1 !043

160 DISPLAY AT(13,12):TOK :: IF TOK<127 THEN DISPLAY AT(11,6):
"LISTING CHARACTER " ELSE DISPLAY AT(11,5):"LISTING TOKEN VALUE" !237

170 PRINT #1:CHR$(INT(TOK/256))&CHR$(TOK-256*INT(TOK/256))&A$&CHR$(TOK)
&CHR$(0)! 007

180 NEXT TOK !132

190 PRINT #1;CHR$(255)&CHR$(255)!084

200 CLOSE #1 :: DISPLAY AT(17,1):"Type 'NEW' to clear memory " :: DISPLAY AT
(19 ,1):"and MERGE DSK"&FN$&" and" !162
```

```
210 DISPLAY AT(21,1):"then LIST 'DSKx.filename'" :: DISPLAY AT(23,1):"to create
a D/V 80 file" :: FOR DELAY=1 TO 1000 :: NEXT DELAY :: END !208
```

Here is one more program that might be of interest. It may be true that an XB program can't read another XB program but there is more than one way to extract lines from a program. I believe that the author of this little beauty is Craig Miller who marketed EXPLORER, DISKASSEMBLER et. al. The program uses CALL PEEK to look into the line number table in memory expansion (line 1), calculates the range of lines to be extracted based on user input, and finally extracts the lines by effective deleting everything else (line 6). If you examine the calculations, you will see that the algorithm for calculating line numbers in this program is similar to the previous program and J.L Stern's program as well.

The advantage of this extraction program is that the result is an XB program that can be immediately added to, resequenced, or otherwise used without merging from disk. You can output the extracted lines to a disk file for later use if you wish, though. It doesn't have the "creature comforts" of J.L. Stern's program.

```
1 CALL CLEAR :: CALL INIT :: INPUT "LINE NUMBERS OF ROUTINE TO BE SAVED: FIRST,
LAST?":L,M :: G=256 :: CALL PEEK(-31952,H,I,J,K)!202

2 C=INT(M/G):: D=M-C*G :: F=(J-G)*G+K :: FOR E=(H-G)*G+I TO F STEP 4 ::
CALL PEEK(E,A,B):: IF A=C AND B=D THEN 4 !201

3 NEXT E :: PRINT :"LINE";0;"NOT FOUND!" :: STOP !@P- !219

4 H=INT(E/G):: I=E-(G*H):: H=H+G :: C=INT(L/G):: D=L-C*G ::
FOR E=E+4 TO F STEP 4 :: CALL PEEK(E,A,B):: IF A=C AND B=D THEN 6 !@P- !025

5 NEXT E :: PRINT :"LINE";N;"NOT FOUND!" :: STOP !@P- !048

6 E=E+3 :: J=INT(E/G):: K=E-(G*J):: J=J+G :: CALL LOAD(-31952,H,I,J,K):: STOP !@P- !161
```

To use this program, SAVE it in MERGE format on disk. Then, when you want to extract a chunk of lines, load your target program and MERGE this program into it and type RUN and follow the prompts. If you plan to extract more than one group of program lines, SAVE your target/extractor program after they are merged since it will have to be re-RUN. Also, you will have to SAVE each group of lines before you can process the next group since nothing will remain in memory except the extracted lines. Enjoy both of the programs!

DUES DUES DUES

Another year has come to a close and it is time to think about your annual contribution to the UG. It might be well to note before we go any further that the Guilford 99'er UG has been around longer than just about any other Users' Group in the country. In fact, with the January issue of the newsletter, we will be starting our 7th full year of production --- and we have yet to miss a month. So, you can count on getting a newsletter each and every month.

To continue, we need your support with dues (which are payable at this meeting) and an article or two written over the course of this next year. C'mon, don't be bashful! Anything will do that is related to the TI. Experiences (good or bad) with a software product, reviews of a TI-related item, or just ramblings about how you use your TI will do just fine.

Since this UG was formed in 1983, there have been a lot of changes --- from a 48K machine to a list of accessories that make the TI as powerful as a top-of-the-line IBM! Who would have thought such things were possible way back then. To be sure, TI is no longer avidly pushing our "orphan" but as Mac Jones will attest, they still support it with repairs and advice! Your only lifeline to all the information is our UG.

I guess the gist of this short article is that now is the time to pay your dues and write an article for the newsletter. We need all of you for 1990 and beyond!!!

GLITCHES AND GREMLINS

By Bob Carmany

Floppy disks and disk drives make a great combination for storing programs. The drives generally perform well and are reliable and the floppy disks are usually a durable storage medium. That is not to say that sometimes things don't go wrong. That's what this is about --- those times when a disk gets "eaten" or altered.

There is one important axiom to remember throughout all this: Disk data remains on the disk until it is physically over-written or the sectors on which it is stored are damaged.

All that means is that what you put on the disk is still there even though you delete a filename. If you save a file to disk with the same name as one that already exists, the old file will be over-written and lost. The data on a disk will be lost if the sectors on which it is stored are physically damaged (ie. by magnetic fields, spilled coffee, etc.).

To start, you will need a disk sector editor and DM1000. The sector editor should be capable of reading in both Hex and ASCII. The two easiest ones to use are John Birdwell's DISK UTILITIES and DISK0 (the enhanced version is in the Funnelweb package). Also, for the purposes of this article ")" denotes the sector number or byte number in hexidecimal. Now we are ready to look at a disk and see what problems we can overcome.

SECTOR >00

This is the very first sector on the disk. It contains the disk name, protection, and initialization data which tells the controller whether or not the disk is initialized and in what format. It also contains the bit-map which tells the controller which sectors are used and which are unused. If this sector is blown or damaged, the results can be catastrophic!! Without the bit-map to guide it, the controller will write to previously used sectors and over-write existing programs. Usually, though, when you put the disk in the drive you will not be able to load a program. Attempting to catalog the disk with DM1000 will give you a "Disk Not Initialized" message. That alone is enough to panic most people!

Actually, this is one of the easiest problems to fix. Using one of the sector editors, simply copy sector >00 from a blank disk that has been initialized in the same format to sector >00 of the "bad" disk. With that done, you can recopy the individual files to a new disk with DM1000. If you get an error message when you try to write the good sector >00 to the suspect disk, it means that the sector is damaged rather than blown. The procedure is just a bit more complicated in that case. The "Copy Sector" selection in DISK UTILITIES works quite well here. All you have to do is copy sectors >01 to >167 to a blank disk initialized in the same format as the original.

SECTOR >01

Sector >01 contains the disk directory link information. Time for a little digression! When a file is saved to disk, it is saved in two parts. The first of these is the file directory. It contains the filename, attributes (ie. D/V 80, PROGRAM, etc.), the sector on which the file starts and the number of sectors that it occupies. All of this information is stored in alphabetical order in sectors >02 to >21. The actual file is stored on the disk at the location specified in the directory.

If you get a "File Not Found" error, the problem is that sector >01 has been blown or damaged. If you use the "Recover File" option with DM1000, you will probably get a "This File has been Over-Written" message. The 'fix' for this problem is a little more difficult than repairing sector >00 but only slightly.

The first thing to do is to "zero out" the sector. This can be done in one of two ways: either replace everything in the sector with "0" by manually editing sector >01 in the "bad" disk or by copying sector >01 from a blank disk to the suspect disk. Once again, if you get an error message when writing the replacement sector to disk it means that the sector is damaged rather than blown. The solution would be to copy the good sectors to a blank, initialized disk.

Okay, we have "zeroed out" sector >01 and we are ready to get to work on fixing the disk directory. Get out a pencil and paper because there is some writing to be done at this point. We know that the disk directory starts at sector >02 and may continue all the way up to >21 depending on the number of programs on the disk. If there haven't been any deletions or additions to the disk since it was created, everything will be in alphabetic order. Switch your disk sector editor to ASCII and start reading the sectors one-at-a-time starting with sector >02. Write down the sector number as a four-digit number (ie. 02 becomes 0002 and 03 becomes 0003, etc.). Then, next to it, write down the filename as it appears at the beginning of the sector. Keep doing this until you reach the end of the disk directory. Alphabetize the file names keeping the corresponding sector numbers next to them. Switch to Hex and write the sector numbers to sector >01 in the order they appear on your alphabetized list in the four-digit form (ie. 000200030004). The last step in the procedure applies if you also repaired sector >00. Use the list of filenames that you have prepared and use the "Recover File" option in DM1000 to re-write the bit-map in sector >00 for each of the files. then, re-copy the files to another disk.

Finally, take the disk that has given you so much trouble and drop it in the nearest trash receptacle. If a disk fails to hold data once, you will probably continue to encounter problems with it in the future. With disks "dirt cheap" a suspect disk simply isn't worth the risk!

Problems with sector >00 and sector >01 will account for the vast majority of "disk gremlins" and the above procedure will cure them quite satisfactorily. However, if the disk directory itself is blown, the fix is difficult at best. The individual directory entries can be repaired but a better solution would be to find another Users' Group member that has the files and ask to copy them from him.