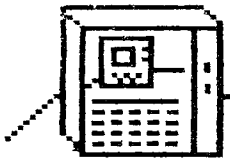
101 $7.10
QB-99



QB MONITOR

QB-99'ERS U.G. NEWSLETTER

Fall 1987 Issue

The QB MONITOR is the Newsletter of the QB-99'ers User Group, is printed Sept. thru June and sent in exchange for other User Group Newsletters. Send Exchange Newsletter to Frank Cotty, Queensborough Community College, Bayside, NY 11364. Credit original sources.

The QB 99'ers meets the second Saturday of each month September through May, at Queensborough Community College, Bayside New York, room S225. See the calendar at right for the dates

November 1987

S	M	T	W	T	F	S
1	2	**3**	4	5	6	7
8	9	10	**11**	12	13	(14)
15	16	17	18	19	20	21
22	23	24	25	**26**	27	28
29	30					

December 1987

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	(12)
13	14	15	**16**	17	18	19
20	21	22	23	24	**25**	26
27	28	29	30	31		

Contents Page

Articles for the DEC issue must be in by DEC 12

WORD COUNTING WITH TI-WRITER

by Ed Machonis

Did you know that you can count the number of words in your essay using only TI-Writer? Neither did I until I made a mistake one day and interchanged LM and RM. Just proves that if you make enough mistakes, something is bound to turn out right. I may be re-inventing the wheel but I have never seen this documented.

To count the words in a DV 80 file, load the file into the Editor, Select SD and note the number of sectors used by the file. Multiply this number by 40 to get a rough estimate. This number will be the Page Length. Now just enter the following line as the very first line: .LM 0;RM 1;FI;PL nnn (nnn being the Page Length derived above) Next count the number of blank lines in your file.

Save to disk, QUIT and select the Formatter. At the first prompt enter the Filename just used to save the file. At the second prompt for Print Device enter DSKn.FILENAME2. You can use the same drive but use a different filename. Accept all the remaining defaults. The Formatter takes roughly one minute for each thousand words.

When the Formatter is done, select Editor and load FILENAME2. You will notice that each word is on a separate line - now you get the idea!

Delete those leading 3 blank lines. Page down using FUNCTION 4 until you reach the end of the file, deleting any lines inserted by the Formatter for page breaks if found. Subtract the number of blank lines previously counted from the line number of the last word and you have your word count.

If your original file was very long, you may have to load FILENAME2 in sections to find the end. Total the number of lines in each section loaded.

There are 313 words in this file.

In the April issue of the QB Monitor was a "Brain Teasers" article. The last two answers to these puzzles are given below (upside down on this page)

Our newsletter exchange is increasing monthly. We are now exchanging with 90 bona fide Newsletter publishing UGs; We are always looking for more. Speaking of newsletters, all newsletters we receive from other groups continue to be inserted in three ring binders by month they were received. If you haven't looked over the newsletters we have in the library you are missing out on the most valuable source of information for your computer. As a member you are entitled to take home a binder and pass it on to other members.

To show the quality of publication we are working for we have included two articles from other groups. The XB tutorial comes from Australia via the Grand Rapids Area 99'er Computer Users Group. The second from the West Jax 99'er News concerns TI-MULTIPLAN.

As usual I have relied on Ed Machonis to provide our end of the work. Ed contributed four of the remaining articles each a gem in brevity of programming.

I just sat down to do this bimonthly issue to make up for not publishing an Oct issue. What a luxury time is!

Note:
At the last meeting the group decided to have a monthly free disk available to each attendee. Each member who attends will receive a disk containing member selected freeware programs. Each month a different member will be in charge of that month's selection. Copying will be donated by Dennis Coyle and Helen Griffin. Disks will be purchased with group funds. If you want to get in on this just show up at meetings!

We received the latest version (4.0) of the Funnelweb loader by way of the Lima area Users Group, OHIO USA. It's even better than the previous version 3.4..
We will be sending a check to Jim Peterson for his tutorial/tips disks also available in our disk library.

STYLE A LINE

A TINYGRAM

by Ed Machonis

TINYGRAM: A short program which can be typed in its entirety on one screen without any program lines scrolling off the screen. (REM statements can scroll off.) Popularized, I believe, by Mike Stanfill of the Dallas TI Home Computer Group.

First of all let me make clear that this is not a novelty program. It is a work horse, provided you have the work for it. What kind of work? Do you ever have to print just a line or two, such as a page header, an article or picture title, a title for a data base printout, a credit line for a reprinted newsletter article, etc., etc. Further, would you like to print this in an Expanded Compressed Italicized Double Strike Underlined type style? Yes all the same time! If so, this program is for you.

What no printer? I will try to have something for you next month. (A TINY GRAM - NOT a printer!)

Many of you are familiar with my 10 Line basic programs, PRINTSTYLE and and PRINTALINE. (Both TINYGRAMS, written before I knew the name existed.) I often use both of them in titling data base printouts or copy for the Newsletter but it got to be a pain to change between the two every time I wanted to change a type style. Finally the light dawned! Why not marry the two?

STYLE A LINE is the result of that marriage. One major revision was to change an INPUT statement in PRINTALINE to a LINPUT. No more need to enclose in quotes any text lines containing commas or leading spaces

Using LINPUT required that the program run in extended basic. After some streamlining by deletion of unneeded features from PRINTALINE and the consolidation of statements into multi-statement lines, we wound up with 9 Lines of code. (After merging TWO TEN Line programs. The power of extended basic!)

Don't let its brevity fool you. You can select any of the 128 type styles available on the Epson RX-80 and many compatibles. With line spacing and margin variations, over 2000 different selections can be had. (Half line spacing and condensed superscript will let you tack on several lines of comment onto a photocopied article.)

Although there are better ways of doing it, you can even produce a right margin justified letter. (THIS is novelty!) Using Emphasized Pica, set Left Margin at 13, and enter text. Two screen lines will print text 54 characters wide (LINPUT uses two character spaces.) Justify text by inserting spaces between words so that second line ends at screen edge. But it will NEVER replace TI-Writer!

Using the program is very easy. When RUN, a menu is displayed for programming the printer. It is always best to select "1" to clear the printer. If your printer doesn't support a master reset code, turn it off then on to clear it. Combine styles by successive selections. Select Option 10 to input text.

If you wish to change the type style, or do repeated printings of the same text, typing "ZZZ" or "zzz" will return you to the menu. Option 9 will do repeat printing of the same text and styles can be changed as required. To input new text, select Option 10 again. When in text mode, pressing ENTER with no text input will print a blank line.

Watch those commas in Line 10. The next to last data item is a lower case "L", not the figure 1.

BRAIN TEASER: Where is the data to set the left margin at column 13?

```
1 ! *** STYLE A LINE   ***
  a TINYGRAM by Ed Machonis
    QB-99ers, Bayside, NY

2 DIM P$(15):: FOR I=1 TO 15
 :: READ P$(I):: NEXT I

3 OPEN #1:"PIO",VARIABLE 132

4 CALL CLEAR :: PRINT "1 PIC
A/RESET","9  PRINT TEXT","2
ELITE","10 INPUT TEXT","3 EX
PANDED","11 SUPERSCRIPT","4
COMPRESSED","12 SUBSCRIPT"

5 INPUT "5 EMPHASIZED  13 1/
2 LINE SP6 ITALIC      14 L
MARGIN 137 D'BLE STRIK 15 R
MARGIN 678 UNDERLINE    ?":I

6 P$(9)=" "&TEX$ :: PRINT #1
:CHR$(27)&P$(I):: IF I=4 THE
N PRINT #1:CHR$(27)&CHR$(15)

7 IF I<>10 THEN 4

8 PRINT :"INPUT TEXT OR 'ZZZ
' FOR MENU" :: LINPUT TRY$

9 IF TRY$="ZZZ" OR TRY$="zzz
" THEN 4 ELSE TEX$=TRY$ :: P
RINT #1:TEX$ :: GOTO 8

10 DATA @,M,W1,,E,4,G,-1,,,S
0,S1,1,1,QC
```

The 2nd word in the 2nd line originally read "listed". Using STYLE A LINE, the printed article was corrected to read "typed".

DISK LABEL II

Print Utility BU
A TINYGRAM From The
Library of Ed Machonis

The original DISK LABEL was a 10 Line Basic program and is on the TIMARC 4/86 disk in our library. It was written to solve the problem of the missing disk labels which were not included with packages of bulk diskettes.

I have been using mailing labels as disk labels for over two years without any problems. They are the preferred label for my disks; the "store boughten" kind are only used as temporary labels until a permanent one can be printed. I find it much easier to locate a disk in a storage case when the name is printed with an expanded type style. Colored ribbons add a nice touch.

The label used as a title for this article is an example of the labels generated by the program. The disk name appears on the first line in expanded emphasized underlined double strike type and is limited to 17 characters. The second line is available for those disks with longer titles or where two titles are appropriate (great for flippies); the same type style is used. Centering of titles is done by the program. If not required, the second line is left blank to enhance appearance and locatability.

The 3rd, 4th and last lines are limited to 28 characters, are printed expanded compressed double strike and, except for the last line, are underlined. The last line is also italicized.

The 3rd line is used for describing the disk contents, such as GAMES, UTILITIES, MP DATA, etc. The end of the line can be used to identify back ups or disk format such as BU, DSDD, etc.

The 4th line is for remarks and can be used for language, loading info, program names, etc. When required, the 3rd and even the 5th line can also be used for remarks.

The last line is used to identify the owner; handy for those round robin copy sessions, ensuring you go home at least with the disks you arrived with. Centering is automatic. It is also useful for identifying a User Group's library copies.

Soon after DISK LABEL was published, a fellow group member suggested a modification to enable text typed for a particular line to be used for the next label if desired. This was done in console basic and the original 5 sector program grew to 10 sectors.

In cleaning up and consolidating the code for this article, it was apparent that Extended Basic's "Accept At" statement would make the program a lot more user friendly. The program was rewritten and a 4 sector Tinygram is the result.

Using the program is very simple, just respond to the prompts. The program automatically limits the number of characters for the various lines of the label so that you cannot type in too long a line. If you notice a typing error after pressing enter, not to worry. Just continue with the other entries and for "How Many?" enter zero. You will be returned to the first line and need only to accept the defaults until the error is displayed. No need to retype, just correct the error.

I always enter 1 for a quantity at first and look over the label to see if it's as intended and then print the number of copies required. I often print a few extra copies for later use and either place them in the back up's jacket or in a label box. Saves reloading the blank labels at some future time just to print a label or two. If you trade many disks, the last line of the extra copies can be left blank.

Usage is not limited to disk labels. It has been used to identify binders of our User Group's newsletter library, name tags, place cards, bookplates, etc.

Epson compressed mode is 137 columns wide. Printers with other widths may change length of underlining. If so just change the TAB setting of the null strings for the respective lines. Epson's Emphasized mode takes precedence over Compressed and cancels it upon return to line 6. Your printer may require print code cancellation at the end of line 7.

The print codes are for the Epson RX-80 printer. If your printer requires different codes, the cast of characters, in order of appearance, are as follows:

[ESC=E$=CHR$(27)]

ESC&"E"	Emphasized
ESC&"6"	Double Strike
ESC&"-1"	Underline
ESC&"W1"	Expanded
ESC&"F"	Cancel Emphasized
CHR$(15)	Compressed
ESC&"-0"	Cancel Underline
ESC&"4"	Italics
ESC&"5"	Cancel Italics

```
1 ! *** DISK LABEL II ***
  A Tinygram by Ed Machonis
    QB-99ers, Bayside, NY

2 OPEN #1:"PIO"

3 DISPLAY AT(3,1)ERASE ALL:"
DISK NAME?":D$ :: ACCEPT AT(
4,1)SIZE(-17):D$ :: DISPLAY
AT(7,1):"Continued?":C$ :: A
CCEPT AT(8,1)SIZE(-17):C$

4 DISPLAY AT(11,1):"TYPE?":T
$ :: ACCEPT AT(12,1)SIZE(-28
):T$ :: DISPLAY AT(15,1):"RE
MARKS?":R$ :: ACCEPT AT(16,1
)SIZE(-28):R$ :: E$=CHR$(27)

5 DISPLAY AT(19,1):"YOUR NAM
E?":N$ :: ACCEPT AT(20,1)SIZ
E(-28):N$ :: INPUT "HOW MANY
 COPIES? ":Q :: FOR J=1 TO Q

6 PRINT #1:E$&"E";E$&"6";E$&
"-1";E$&"W1";TAB((18-LEN(D$)
)/2);D$;TAB(18);"";TAB((18-L
EN(C$))/2);C$;TAB(18);"";E$&
"F";CHR$(15);TAB(2);T$;

7 PRINT #1:TAB(30);"";TAB(2)
;R$;TAB(30);"";E$&"-0";E$&"4
";TAB((30-LEN(N$))/2);N$;E$&
"5" :: NEXT J :: GOTO 3
```

QB MONITOR ~ QB-99'er NEWSLETTER

The following tutoral comes from Funnelweb Farm and are excellent information on Extended Basic programing. These were downloaded from GEnie. And will take the next 3 issues of Call Say. There is a lot of material on subprograming technic. These came from a BBS called the EASY CHAIR 1-414-384-2720, 300/1200 baud, 24 hours, 8N1. It has RLE viewing and downloading with Omegaterm and also Masstransfer MXT (muliple xmodem transfers) for downloading after you are verified. The sysop told me that he got the Funnelweb tutorials with the BBS software so that's all I know of their history.

Edited 08/22/87

EXTENDED BASIC TUTORIALS FUNNELWEB FARM

I. INTRODUCTION

In this series of notes on TI Extended Basic for the TI-99/4a we will concen- trate on those features which have not received due attention in User-group newsletters or commercial magazines. In fact most of the programs published in these sources make little use of that most powerful feature of XB, the user defined sub-program, or of some other features of XB. Worse still is to find commercially available game programs which are object lessons in how to write tangled and obscure code. The trigger for this set of tutorial notes was a totally erroneous comment in the TI- S.H.U.G Newsdigest in Jun 1983. Some of the books I have seen on TI Basic don't even treat that simpler language correctly, and I don't know of any systematic attempts to explore the workings of XB. The best helper is TI's Extended Basic Tutorial tape or disk. The programs in this collection are unprotected and so open for inspection and it's worth looking at their listings to see an example of how sub-programs can give an easily understood overall structure to a program.

Well, what are we going to talk about then ? Intentions at the moment are to look at:

(1) User-defined sub-programs (2) Prescan switch commands (3) Coding for faster running (4) Bugs in Extended Basic (5) Crunching program length (6) XB and the Peripheral Box (7) Linking in Assembler routines

Initially the discussion will be restricted to things which can be done with the console and XB only. Actually, for most game programming the presence of the memory expansion doesn't speed up XB all that much as speed still seems to be limited by the built-in sub-programs (CALL COINC,etc) which are executed from GROM through the GPL interpreter. The real virtue of the expansion system for game programming, apart from allowing longer programs, is that GPL can be shoved aside for machine code routines in the speed critical parts of the game, which are usually only a very small part of the code for a game. Even so careful attention to XB programming can often provide the necessary speed. As an example, the speed of the puck in TEX-BOUNCE is a factor of 10 faster in the finally released version than it was in the first pass at coding the game.

Other topics will depend mainly on suggestions from the people following this tutorial series. Otherwise it will be whatever catches our fancy here at Funnelweb Farm.

II. SUB-PROGRAMS in OVERVIEW

Every dialect of Basic, Extended Basic being no exception, allows the use of subroutines. Each of these is a section of code with the end marked by a RETURN statement, which is entered by a GOSUB statement elsewhere in the program. When RETURN is reached control passes back to the statement following the GOSUB. Look at the code segments.

```
290 .... 300 GOSUB 2000 310 ....
2000 CALL KEY(Q,X,Y):: IF Y=1 THEN RETURN ELSE 2000
```

This simple example waits for and returns the ASCII code for a fresh key-stroke, and might be called from a number places in the program. Very useful, but there are problems. If the line number of the subroutine is changed, other than by RESequencing of the whole program (and many dialects of Basic for microcomputers aren't even that helpful) then the GOSUBs will go astray. Another trouble, which you usually find when you resume work on a program after a lapse of time, is that the statement GOSUB 2000 doesn't carry the slightest clue as to what is at 2000 unless you go and look there or use REM statements. Even more confusingly the 2000 will usually change on RESequencing, hiding even that aid to memory. There is an even more subtle prob-

you don't really care what the variable "Y" in the subroutine was called as it was only a passing detail in the sub - routine. However, if "Y" is used as a variable anywhere else in the program its value will be affected. The internal workings of the subroutine are not separated from the rest of the program, but XB does provide four ways of isolating parts of a program.

(1) Built-in sub-programs (2) DEF of functions (3) CALL LINK to machine code routines (4) User defined BASIC sub-programs

The first of these, built-in sub-programs, are already well known from console Basic. The important thing is that they have recognizable names in CALL statements, and that information passes to and from the sub-programs through a well defined list of parameters and return variables. No obscure Peeks and Pokes are needed. The price paid for the power and express iveness of TI Basic/XB is the slowness of the GROM/GPL implementation.

DEF function is a primitive form of user defined sub-program found in almost all BASICs. Often its use is restricted to a special set of variable names, FNA,FNB, but TI Basic allows complete freedom in naming DEFed functions (as long as they don't clash with variable names). The "dummy" variable "X" is used as in a mathematical function, not as an array index

```
100 DEF CUBE(X)=X*X*X
```

doesn't clash with or affect a variable of the same name "X" elsewhere in the program. "CUBE" can't then be a variable whose value is assigned any other way, but "X" may be. Though DEF does help program clarity it executes very slowly in TI Basic, and more slowly than user defined sub-program CALLs in XB.

CALL LINK to machine code routines goe under various names in other dialects of Basic if it is provided (eg USR( ) in some). It is only available *in XB* when the memory expansion is attached, as the TI-99/4a console has only 256 bytes of CPU RAM for the TMS9900 lurking in there. We will take up this topic later.

You should have your TI Extended Basic Manual handy and look through the section on SUB-programs. The discussion given is essentially correct but far too brief, and leaves too many things unsaid. From experiment and experience I have found that things work just the way one would reasonably expect them to do (this is not always so in other parts of XB). The main thing is to get into the right frame of mind for your expect -ations. This process is helped by figuring out, in general terms at least, just how the computer does what it does. Unfortunately most TI-99/4a manuals avoid explanations in depth presumably in the spirit of "Home Computing". TI's approach can fall short of the mark, so we are now going to try to do what TI chickened out of.

The user defined sub-program feature of XB allows you to write your own sub- programs in Basic which may be CALLed up from the main program by name in the same way that the built-in ones are. Unlike the routines accessed by GOSUBs the internal workings of a sub-program do not affect the main program except as allowed by the parameter list attached to the sub-program CALL. Unlike the built-in sub-programs which pass infor- mation in only one direction, either in or out for each parameter in the list, a user sub-program may use any one variable in the list to pass information in either direction. These sub-programs provide the programming concept known as "procedures" in other computer languages, for instance Pascal, Logo, Fortran. The lack of proper "procedures" has always been the major limitations of BASIC as a computer language. TI XB is one of the BASICs that does provide this facility. Not all BASICs, even those of very recent vintage are so civilised. For example the magazine Australian Personal Computer in a recent issue (Mar 84) carried a review of the IBM PCjr computer just released in the US of A. The Cartridge Basic for this machine apparently does not support procedures. Perhaps IBM don't really want or expect anyone to program their own machine seriously in Basic. You will find that with true sub-programs available, that you can't even conceive any more of how one could bear writing substantial programs without them (even within the 14 Kbyte limit of the unexpanded TI-99/4a let alone on a machine with more memory).

The details of how procedures or sub- programs work vary from one language to another. The common feature is that the variables within a procedure are localised within that procedure.

How they communicate with the rest of the program, and what happens to them when the sub-program has run its course varies from language to language. XB goes its own well defined way, but is not at all flexible in how it does it.

Now let's look at how Extended Basic handles sub-programs. The RUNning of any XB program goes in two steps. The first is the prescan, that interval of time after you type RUN and press ENTER, and before anything happens. During this time the XB interpreter scans through the program, checking a few things for correctness that it couldn't possibly check as the lines were entered one by one, such as there being a NEXT for each FOR. The TI BASICs do only the most rudimentary syntax checking as each line is entered, and leave detailed checking until each line is executed. This is not the best way to do things but we are stuck with it and it does have one use. At the same time XB extracts the names of all variables, sets aside space f r them, and sets up the procedure by which it associates variable names with storage locations during the running of a program. Just how XB does this is not immediately clear, but it must involve a search through the variable names every time one is encountered, and appears to trade off speed for economy of storage.

XB also recognizes which built-in sub-programs are actually CALLed. How can it tell the difference between a sub-program name and a variable name? That's easy since built-in sub-program names are always preceded by CALL. This is why sub-program names are not reserved words and can also be used as variable names. This process means that the slow search through the GROM library tables is only done at pre-scan, and Basic then has its own list for each program of where to go in GROM for the GPL routine without having to conduct the GROM search every time it encounters a sub-program name while executing a program. In Command Mode the computer has no way provided to find user defined sub-program names in an XB program in memory even in BREAK status. XB also establishes the process for looking up the DATA and IMAGE statements in the program.

Well then, what does XB do with user sub-programs? First of all XB locates the sub-program names that aren't built into the language. It can do this by finding each name after a CALL or SUB statement, and then looking it up in the GROM library index of built-in sub- program names. You can run a quick check on this process by entering the one line program

```
100 CALL NOTHING
```

TI Basic will go out of its tiny 26K brain and halt execution with a BAD NAME IN 100 error message, while XB, being somewhat smarter, will try to execute line 100, but halts with a SUBPROGRAM NOT FOUND IN 100 message. The XB manual insists that all sub-program code comes at the end of the program, with nothing but sub-programs after the first SUB statement (apart from REMarks which are ignored anyway). XB then scans and establishes new variable storage areas, starting with the variable names in the SUB xxx(parameter list), for each sub-program from SUB to SUBEND, as if it were a separate program. It seems that XB keeps only a single master list for sub-program names no matter where found, and consulted whenever the interpreter encounters a CALL during program execution. Any DATA statements are also thrown into the common data pool. Try the following little program to convince yourself.

```
100 DATA 1 110 READ X :: PRINT X :: READ X :: PRINT X 120 SUB NOTHING 130 DATA 2 140 SUBEND
```

When you RUN this program it makes no difference that the second data item is apparently located in a sub-program. IMAGEs behave likewise. On the other hand DEFed functions, if you care to use them, are strictly confined to the particular part of the program in which they are defined, be it main or sub. During the pre-scan DEFed names are kept within the allocation process separately for each subprogram or the main program. Once again try a little programming experiment to illustrate the point.

```
100 DEF X=1 :: PRINT X;Y :: CALL SP(Y) :: PRINT X;Y 110 SUB SP(Z) :: DEF X=2 :: Z=X :: DEF Y=3 120 SUBEND
```

This point is not explicitly made in the XB manual and has been the subject of misleading or incorrect comment in magazines and newsletters. A little reflection on how XB handles the details will usually clear up difficulties. TI BASICs assign nomina. values to all variables mentioned in the program as part of the prescan, zero f r numeric and null for strings, unl e some languages (some Basics even) w ch

will issue an error message if an unassigned variable is presumed upon. This means that XB can't work like TI LOGO which has a rule that if it finds an undefined variable within a procedure it checks the chain of CALLing procedures until it finds a value. However, unlike Pascal which erases all the information left within a procedure when it is finished with it, XB retains from CALL to CALL the values of variables entirely contained in the sub-program. The values of variables transferred into the sub-program through the SUB parameter list will of course take on their newly passed values each time the sub-program is CALLed. A little program will show the difference.

```
100 FOR I=1 TO 9 :: CALL SBPR(0)::
NEXT I 110 SUB SBPR(A):: A=A+1 :: B=B+1
:: PRINT A;B 120 SUBEND
```

The first variable printed is reset to 0 each time SBPR is called, while the second, B, is incremented from its previous value each time. Array variables are stored as a whole in one place in a program, within the main program or sub-program in which the DIMension statement for the array occurs. XB doesn't tolerate attempts to re-dimension arrays, so information on arrays can only be passed down the chain of sub-programs in one direction. Any attempt by a XB sub-program to CALL itself, either directly or indirectly from any sub-program CALLed from the first, no matter how many times removed, will result in an error. Recursive procedures, an essential part of TI LOGO, are NOT possible with XB sub-programs, since CALLing a sub-program does not set up a new private library of values.

All of this discussion of the behaviour of TI Extended Basic comes from programming experience with Version 110 of XB on a TI-99/4a with 1981 title screen. Earlier Versions and consoles are not common in Australia, but TI generally seems to take a lot of trouble to keep new versions of programs compatible with the old. On the other hand TI has also been very reticent about the details of how XB works. The Editor/Assembler manual has very little to say about it, less by far even than it tells about console Basic. I am not presently aware of any discussion of the syntax of the Graphics Programming Language (GPL), let alone of the source code for the GPL interpreter which resides in the console ROM of every 99/4a.

Another simple programming experiment will demonstrate what we mean by saying that XB sets up a separate Basic program for each sub-program. RUN the following

```
100 X=1 :: CALL SBPR :: BREAK 110
SUB SBPR :: X=2 :: BREAK :: SUBEND
```

When the program BREAKs examine the value of variable X by entering the command PRINT X, and then CONtinue to the next program BREAK, which this time will be in the main program, where you can once again examine variable values.

(a) XB treats each sub-program as a separate program, building a distinct table of named (REFed) and DEFed variables for each.

(b) All DATA statements are treated as being in a common pool equally accessible from all sub-programs or the main program as are also IMAGE statements,CHARacters,SPRITEs,COLORs, and File specifications.

(c) All other information is passed from the CALLing main or sub- program by the parameter lists in CALL and SUB statements. XB does not provide for declaration of common variables available on a global basis to all sub-programs as can be done in some languages.

(d) Variable values confined within a sub-program are static, and preserved for the next time the sub-program is CALLed. Some languages such as Pascal delete all traces of a procedure after it has been used.

(e) XB sub-programs may not CALL themselves directly or indirectly in a closed chain. Subject to this restriction a sub-program may be CALLed from any other sub-program.

(f) The MERGE command available in XB with a disk system (32K memory expansion optional) allows a library of XB sub- programs to be stored on disk and incorporated as needed in other programs.

III. SUBPROGRAM PARAMETER LISTS

In the last chapter we saw how subprograms fitted into the overall workings of Extended Basic. In this chapter we are going to go into the details of writing subprograms. Most of the fiddly detail here concerns the

construction of the parameter list attached to CALL and SUB statements, and some of the little traps you can fall into.

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

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

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

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

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

```
230 CALL DELAYSEC(2) 990 CALL DELAYSEC(T) 4000 SUB DELAYSEC(A):: A=A0
4010 FOR I= 1 TO A :: NEXT I :: SUBEND
```

Now after DELAYSEC has been executed with the CALL from 990, T will have value 250 times its value before the CALL. This won't be a bother if you don't use T again for its previous value. If the CALLing program specifies a numeric constant as in line 230, or a numeric expression, the change in A in the subprogram has no effect on the main program. Suppose you can't tolerate T being changed in line 990 (and this kind of thing can be a source of program bugs). You will find that XB allows for forcing T to be treated as though it were an expression, thus isolating T from alteration by the subprogram. if T is enclosed in brackets in the CALL (not SUB) list. Suppose DELAYSEC is also called from line

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

If this CALL in line 970 is followed by the CALL from line 990, T not having been altered in the meanwhile, the same delay will be obtained, but if the order of CALLs were reversed the second delay would be 250 times the first. In the language of XB this is known as "passing by value" as distinct from "passing by reference". This can only be done for single variables or particular array elements, which behave like simple variables in CALL lists. Whole arrays cannot be passed by value, but only by reference. Expressions and constants can only be passed by value, and its hard to see what else could be done with them. In the example as written, a different variable name was used in the SUB, but if you remember the little experiment in the last chapter you'll see that it wouldn't make any difference if T had been used in the SUB list instead of A.

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

```
200 CALL MESSAGE(300," YOUR TURN NOW") 270 CALL MESSAGE(T,A$) 3000 SUB MESSAGE(A,A$):: DISPLAY AT(24,1):A$ 3010 FOR I=1 TO A :: NEXT I :: DISPLAY AT(24,1):"" 3020 SUBEND
```

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

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

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

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

```
100 DIM SC(2,5) :: .... 450 CALL BOARD(P,A$(),SC(,)) 4000 SUB BOARD(P,A$(),S(,)):: DIM AY(5):: .... 4080 SUBEND 5000 SUB REF(V,A(),B(,)):: .... :: SUBEND
```

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

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

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

For completeness note that , although the XB manual has nothing to say about it, IMAGE statements for

formatting PRINT output are accessible from any part of a program in the same way as DATA statements and not confined to the subprograms in which they occur as are DEF entries.

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

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

IV. USEFUL SUBPROGRAM EXAMPLES

In the previous chapter we used as an example a DELAY subprogram which could, with a little refinement, be used to substitute for the WAIT command available in some other languages. You can extend this idea to build up for yourself a library of handy-dandy subprograms which you can use in programs to provide your own extension of the collection of subprograms that XB offers.

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

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

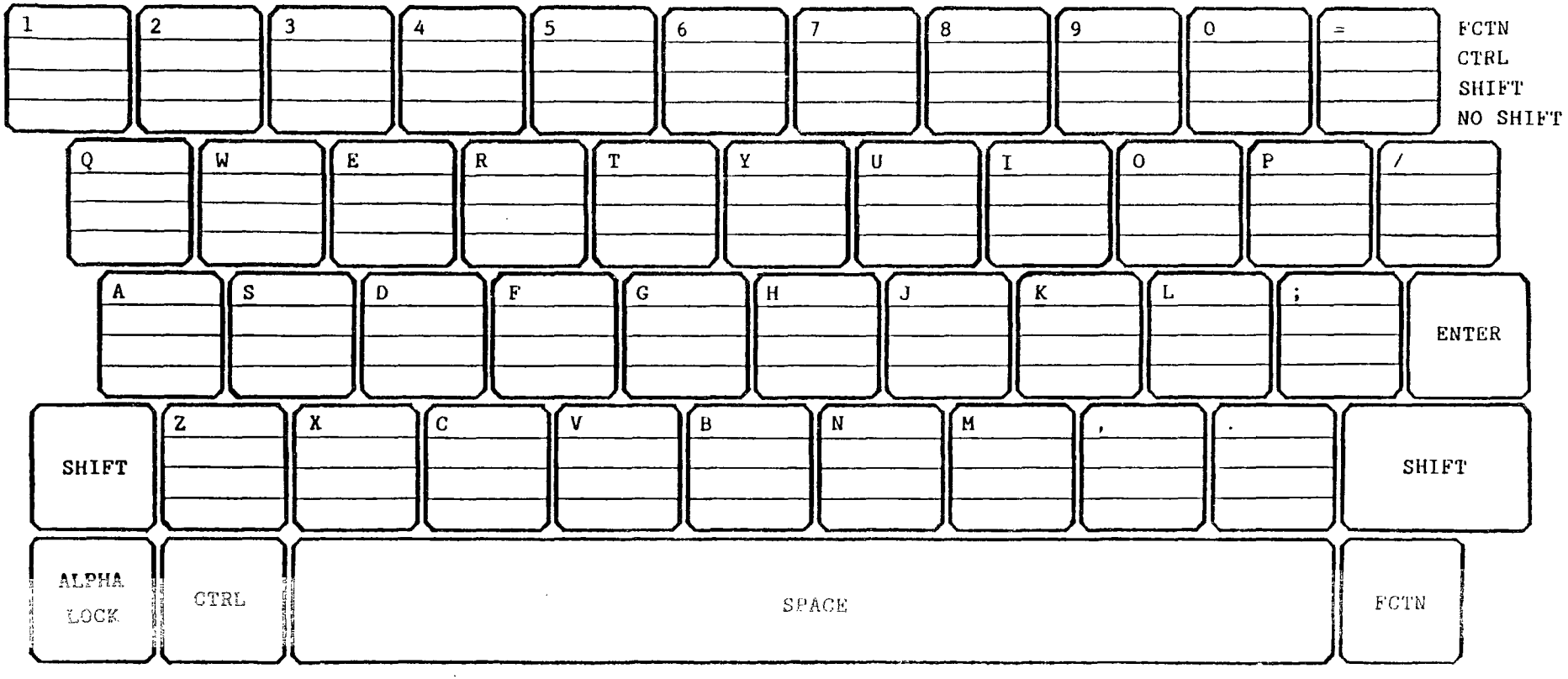
The first step in telling whether keys or joysticks are being used is to check the keys, and if none have been pressed then to check the joysticks. If a key has been pressed then its return, K, has to be processed so that the direction pads embedded in the keyboard split-scan return the corresponding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

```
900 SUB JOY(PL,X,Y):: CALL KEY(PL,K,ST):: IF ST=0 THEN CALL JOYST(PL,X,Y):: SUBEXIT 910 X=4*((K =4 OR K=2 OR K=15)-(K=6 OR K=3 OR K=14)) 920 Y=4*((K=15 OR K=14 OR K=0)-(K=4 OR K=5 OR K=6)) 930 SUBEND
```

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

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

TI-99/4A KEYBOARD LAYOUT



PROGRAM NAME: _____

Notes : _____

ORIGINAL SOURCE UNKNOWN.

TI-MULTIPLAN

By
Rick Felzien
West Jax 99ERS

I think that most of us have at some time acquired a Multiplan package and because we don't understand the package and it's many powers, have stored it away with other items that we seldom if ever have occasion to use.

The Multiplan package is a very powerful and useful tool which has a multitude of uses. I have designed a template or model which allows me to use Multiplan at least once a month. I decided that as long as I had it I might as well use it.

In this article I will do a walkthrough on a template that I use to reconcile my checkbook and do the statement balancing, then save the file for later recall. This allows me to store the actual statements and checks away in a box in the attic or garage, or wherever a person stores things that are seldom used. If I need information concerning my checking I just go to that Multiplan file and obtain the data. You can also do a printout of any given month. The months are also linked.

Before I talk about my template, there was an article in the April 1985 issue of MICROpendium page 35, which was written by Earl Hall of Chicago Il, that tells how to change disk drive and printer defaults so that you don't have to change them every time you want to use Multiplan. At this point I will go through a walkthrough for the changes using Advanced diagnostics. The first thing to do to make the job easier is to initialize a disk and copy the MPINTR file to it. The reason for this is that it makes finding the right data blocks easier. Another tip which will speed up the use of multiplan is to arrange the program files in a certain order on the disk, which effects access time. This should be done in the following order. OVERLAY, MPHLP, MPCHAR, MPDATA, MPINTR, and MPBASE.

After you have saved the MPINTR file to a new disk, Load up Diags. and Edit Sector >22(34), which will be the first data sector. Shift into ASCII screen mode and find the words DSK1 and RS232 and it's defaults. Move the cursor to the appropriate spots to change DSK1 to DSK2 and RS232 to PIO. The RS232 data after that should be blanked out by using the space bar, if you use delete character, The end of the data block will be messed up. Now Write Sector back to the disk and you are all set.

The first thing to do is to insert the Multiplan Cartridge and place the Program disk in drive #1. For those with two or more drives, te data disk should be placed in drive #2. I have written this article with the assumption that you have at least two drives. After loading the program set, select 'O' for OPTIONS and enter 'N' for no auto-recalc. If you don't make this selection the program will go into Recalc mode after each entry.

The first thing we want to do after loading the Multiplan files, is to set up the various workspaces by naming the work areas and afterword we will enter our formulas for doing the necessary calculations.

First use FCTN[1](HOME), this places the cursor at row1, column1 (R1C1). Starting at R1C1, select ALPHA and type in 'CHECKS', then press FCTN X which will enter the name and advance the cursor down one row. Next enter '_____' (6 underline characters) and press FCTN D which will enter the underline characters and advance the cursor one column to the right. Now use FCTN E to put the cursor at the beginning of the second column. Now type in 'Date' and FCTN X and then enter '_____'(4 underline characters) and press FCTN D then FCTN E. Repeat the procedure for the next three columns entering the following names 'PAID_TO', 'AMOUNT', and 'DEPOSIT. For the next column we don't want to enter the underline characters, the reason will be explained later. However at R1C6 we want to enter 'BALANCE'. Now we will name your areas. First HOME the cursor and hit N for Name. The first field should now display the name that you typed in, Hit CTRL A to advance to the next field and then FCTN 4 and CTRL 4 twice. Then hit SHIFT colon and enter 55. The field should now read R1:55CX(where X equals the current column). Do this for all six of the columns with the names which you have typed in.

Now we want to format our columns, I used 55 rows as that was two more than the largest amount of checks that I had written in a given month. First HOME the cursor to R1C1 and then check to see if the help lines are at the lower area of the screen, If not CTRL A will bring them up. Now enter F for FORMAT and then select columns and using CTRL A to advance to each field, select the following. Before we start formatting the columns starting at line three, keep in mind that for the first field we can hit FCTN 4 which will place the cursor at the R in R3CX (with X being the current column). Now hit CTRL 4 twice and observe the cursor. Hit SHIFT colon and then 55. The field should now read R3:55CX, which meand from R3CX to R55CX. Enter D for decimal and hit CTRL A then C for center then CTRL A and SHIFT dollar sign for Dollar format. Then use CTRL A again and enter 2 for the default of two culumns after decimal point. Now for column two we want to enter G for Gen, CTRL A, G for Gen, CTRL A, and leave the rest of the defaults. Do the same with column 3. In columns 4, 5, and 6 we want to enter D for Decimal, dollar sign for dollars, and 2 for decimal places. However in column 6 we want to start in row 2 instead of 3. Now we must format the width of our columns as follows. Here we want to start at row 1 and select F for format and use the same procedure as before in the first field. Columns 1 and 2 can be left at te TIMP default width. Three should be 15, 4 and 5 should be 9 and column 6 should be 10.

Now lets enter a formula for subtracting the checks and adding the deposits. Place your cursor in the cell at R3C6 and hit the equal sign. Note that the command line Asks for a value. Using the up arrow, move the cursor up one row to R2C6 and then hit SHIFT minus. You will notice that the cursor returned to R3C6, but a formula started to build on the command line. Now move the cursor to R3C4 and hit SHIFT plus and the cursor will again return to R3C6. Now move the cursor to R3C5 and hit enter this time. This will enter your formula to take the balance at R2C6 subtract the check value at R3C4 and then add (if any) the value of the deposit at R3C5, then place the new balance at R3C6. The formula should read thus R[-1]C-RC[-2]+RC[-1]. Next use CTRL A to get the help lines and enter C for Copy. Then select D for down and for the number of rows as 52 since we are starting at row 3. This will copy the formula down to the desired 55 rows.

Now we want to set up an area for reconciling the checklog against the monthly bank statement. Advance the cursor to R61C1 and type in OSTD_CK and this will be where we will enter the checks that were written after the statement date. At R61C2 enter PND_DEP for pending deposits after the statement date. Name the cells R62:77CX and then select Format Cells and use R62:77CX then use D for decimal, Shift dollar sign for dollar format, and 2 decimal places. Move the cursor to R77C1 and hit the equal sign and type in SUM(OSTD_CK) and hit enter. At R77C2 use SUM(PND_DEP). this will cause the total of what is entered in go to R61C3 and type in 'S_BAL' and these cells R61:62C3 as such, this is where we will enter the ending balance from our bank statement. then at R64:65C3 we want to assign 'CK_BAL' to carry our balance forward and use it in our next formula. Now assign R67:68C3 'CORR', this will be the final value to see whether we need to correct our checkbook or not.

Now we can enter the final mathematics formulas. First at R65C3 enter the following formula by hitting the equal sign and typing R[-10]C[+3]. This will carry the total from column 6 to this space. Next, at R68C3 hit equal again and enter the following formula to comput the correction if any. R[-6]C-R[+9]C[-2]+R[+9]C[-1]-R[-3]C, then hit enter. If you follow this formula step by step you will see that it takes the S_BAL(statement balance) and subtracts the SUM(OSTD_CK) or total of outstanding checks, then adds SUM(PND_DEP) or the total pending deposits, and then subtracts the CK_BAL to find out if the statement balance is equal to our checkbook balance.

If I get a CORR value other than zero, I first check to see whether I may have entered a value wrong, and if so enter the correct amount. If all is well with my entries, I know that I need to change my checkbook as there is a mistake somewhere in my figures that are usually done mentally anyway.

Now hit FCTN 8 for Recalc and when it is finished, the areas that have formulas related to them should contain $0.00. if not, recheck the formulas and try again. If all goes well we can now lock opur formulas by selecting L for Lock and then F for formulas and the formulas will be permenantly locked in. Before saving your template to disk moce the cursor to R2C6 and then when you load it you will start at that point every time you load it.

When you go to use the template the first time, just enter the balance where you wish to start in the checkbook. When saving each month's file I name it for the statement month such as JAN87 which contains the last of December and the first of January checks contained in the statement for January. After the first month has been saved, from then on, since you will start in R2C6 you can go to External copy immediately after loading the template, which I named CHECKLOG. To do this hit X for external, then C for copy then enter the previous month's file name and then CTRL A to advance to the next field. Now enter R65C3 and the template will load the previous month's CK_BAL and continue from there in the calculations.

I hope you find this as useful as I have and I hope to do some more Multiplan articles in the near future.

COLISTER

A TINYGRAM

by Ed Machonis

Another 28 column lister? Why not? This one happens to be my favorite and not just because I wrote it. I like it because it does the job the way I want it done, but then I wrote it that way.

At the time I wrote COLISTER, I had no access to any program that could do what I wanted done, which was to be able to list a program to disk or printer in 28 column format, the way it appears on the screen.

A 28 column listing makes it easier for the reader to type in the program with less chance for error. It also makes it simpler to check for errors should any creep in. One only has to check the end of each line as it appears on the screen against the printed listing to see if any characters were omitted or added. (Home Computer magazine never did learn this lesson.)

But the biggest reason is that it not only saves the work of typing in a program in 28 column format, but it eliminates the chance for typing errors. By letting the computer do the work, nothing can go wrong. (If you believe this, I have a fantastic deal on a Bridge I'd like to tell you about!)

Why not just LIST to Printer or Disk? It's not that simple. The computer will list the program in 80 column format. Why not set the printer's right margin at 28? It will work up to a point. The point being a program line of more than 80 characters. The computer will send a carriage return after the 80th character and start printing the rest of the code on a new line. Listing to disk will also give you an 80 column listing.

Since I originally wrote this program several years ago, two programs that do the same work have been brought to my attention. One is 28 Column Converter by Jim Peterson, published in Tigercub Tips #18, and the other is COLIST, a Fairware program by the McGoverns. Both are very nice programs and you may well find them more useful to you than the one presented here. (I had originally named my program COLIST but have since renamed it COLISTER to avoid confusion.)

COLISTER has a couple of features not available in the other programs. First, it will print a blank line between program lines. I feel this makes it easier to "read" the program, especially the spaghetti code I am prone to. It facilitates picking out a line number in the middle of the program when following those GOTOs and orELSEs.

Second, it TABs the output 6 spaces. This centers the listing when merged into 40 column text in TI-Writer's Editor, and provides a margin so hard copies can be loose leaf bound.

COLISTER does not require that a program's line numbers be resequenced in order to list it. A lot of my program lines are numbered from 1 to 10. Default resequencing (100,10) would sometimes destroy their Tinygram status. (COLISTER is a good example. One Tinygram "trick" is to use single digit line numbers to gain a few extra character spaces for your code.)

COLISTER will print to either disk or printer. Listings printed to disk can the be merged with text in TI-Writer's Editor. Do not print the listing through the Formatter unless you have modified your Formatter file to ignore the special format command characters that are also often found in programs.

This Tinygram uses only 4 sectors of disk space, which can be reduced to 3 sectors by deleting Line 1. It earns its keep on my SSSD utility disk. (Small is Beautiful)

Using COLISTER is very simple. First, load into memory the program you want to list. Next make a DV 80 listing by typing LIST "DSKn.FILENAME". Don't use the same filename as the program or the listing can overwrite the program.

Then load and RUN COLISTER. At the first prompt, enter the DSK number and the filename used above. For the second prompt, enter the print device name. This can be either PIO, RS232, or DSKn.FILENAME2. Again, use a different filename if reading from and writing to the same drive.

If you don't want the blank line between program lines, just change the FOR statement in Line 8 to read: FOR I=0 TO L-1. The TAB setting in this line can also be changed or eliminated, as desired. If for some reason you want a listing with a different width, say 40 columns for those "other" owners, just change the value of C in Line 5. (The reason it's in Line 5, and being constantly updated, is because that's where the room was. Another Tinygram "trick".)

If you prepare program listings for newsletters, I think you'll find this program useful. The algorithm used to detect a new line number is relatively unsophisticated. It hasn't failed me yet, but I'm sure that someone, someday will write code that will trip it up. For that reason it is well to always look over the output to be sure that lines have not been split or joined when they should not have been.

```
1 !   *** COLISTER ***
 A Tinygram by Ed Machonis
   QB-99ers, Bayside, NY

2 PRINT :"1st LIST your prog
ram to diskThen RUN COLISTER"

3 PRINT :;:"INPUT FILENAME?
ex:DSKn.LIST" :: INPUT F$ ::
 INPUT "OUTPUT FILENAME? ex:
 PIO or DSKn.LIST28 :":P$

4 OPEN #1:F$,INPUT :: OPEN #
3:P$,OUTPUT :: ON ERROR 10

5 C=28 :: LINPUT #1:A$ :: IF
 LEN(A$)<80 THEN 8

6 LINPUT #1:B$ :: IF VAL(SEG
$(A$,1,POS(A$," ",2)))<VAL(S
EG$(B$,1,POS(B$," ",2)))THEN
 F=1 :: GOTO 8

7 A$=A$&B$ :: IF LEN(B$)>=80
 THEN 6

8 A=LEN(A$):: L=A/C+.99 :: F
OR I=0 TO L :: PRINT #3:TAB(
6);SEG$(A$,1+I*C,C):: NEXT I
 :: IF EOF(1)AND F=0 THEN CL
OSE #1 :: CLOSE #3 :: END

9 IF F=1 THEN F=0 :: A$="" :
: GOTO 7 ELSE 5

10 ON ERROR 10 :: RETURN 7
```