$1.50

THE NATIONAL NINETY-NINER

VOL III - NO. 6 - JUNE, 1986

THE 99ER'S ASSOCIATION
3535 SO.H ST., #26
BAKERSFIELD, CALIF. 93304
(805) 397-4361
DON VEITH - EDITOR

CREATED FOR TI 99/4A HOME COMPUTER OWNERS

COMPUSERVE ID #: 72257,3671

TABLE OF CONTENTS

THE NATIONAL NINETY-NINER

VOL III - NO. 6 - JUNE, 1986

THE 99ER'S ASSOCIATION
DON VEITH - EDITOR
COMPUSERVE ID #: 72257,3671

ANNOUNCEMENTS

TI EQUIPMENT PARTS

Call (806) 762-7457 to obtain parts for any equipment related to the 99/4A. The number will place you in contact with the Dealer Parts Department of Texas Instruments. If you care to write instead of calling, their address is Dealer Parts Department; P.O. Box 53; Lubbock, TX 79408.

ANOTHER SOURCE FOR CORCOMP CARD REPAIRS

Repairs for CorComp cards is being offered by Don Scofield. The repair costs are $35.00 per card. All repaired cards have a 30 day warranty period. Don Scofield is one of the original members of CorComp and may be contacted at:

CLELAND CONTROLS CORP 2212 DUPONT, SUITE 6 IRVINE, CA 92715

CorComp has stated the firm above is not an authorized repair center for their products.

CORCOMP TRIPLE-TECH CARD BATTERY PROBLEM

Mark Keeler, a subscriber from Dayton, Ohio, forwarded the information contained in the article below from The Source. Tery Atkinson was responsible for placing the notice there. The notice is shown below as it was posted.

Subject: EXPLODING TRIPLETECH FIX
From : TI6450
Posted : 4MAY 1986 1:47 am

The following message was downloaded from the Canadian Database Timeline. It was uploaded by GIL.A15C5 to Timeline. Begins:

I HAVE DEVELOPED A FIX FOR THE EXPLODING BATTERY PROBLEM ON THE CLOCK CARD. FOR THOSE OF YOU WITH THE CARD AND CAN, OR KNOW OF SOMEONE WHO IS ELECTRONICALLY INCLINED, THIS IS THE FIX TO PREVENT BATTERY CHARGE DURING P-BOX OPERATION:

1) LOCATE RESISTOR #7 (TO LOWER RIGHT OF THE SPEECH SYNTHESIZER SLOT)

2) ALSO NEAR THE CLOCK IC 5832

3) WITH A 15 WATT PENCIL SOLDERING IRON OR DESOLDERING TOOL, REMOVE THE #7 RESISTOR.

4) REPLACE IT WITH A IN914 DIODE WITH THE ANODE END TOWARDS THE BATTERY.

5) SOLDER THE DIODE IN PLACE AND THEN DO A METER TEST ON THE THE BATTERY CONTACT LEADS WITH THE BATTERY REMOVED AND THE CARD IN PLACE WITH THE P-BOX RUNNING. PRESTO! NO VOLTAGE AT THE LEADS! PLACE THE BATTERY IN, SET THE CLOCK, TURN OFF THE SYSTEM FOR 10 MINUTES AND THEN REBOOT THE CLOCK PROGRAM. VIOLA! YOU HAVE JUST PREVENTED THE BATTERY FROM BEING CHARGED! THIS FIX HAS BEEN CONFIRMED BY CORCOMP TO WORK WITH NO LOSS OF FUNCTION TO THE CARD!

DO NOT ATTEMPT TO MAKE THIS MODIFICATION IF YOU ARE NOT QUALIFIED TO DO IT! THE INDIVIDUAL CARD OWNER ASSUMES ALL RESPONSIBILITY ANY MODIFICATIONS MADE TO THEIR CARD.

EDITOR'S NOTE: If you own a Triple-Tech Card, I would advise you to contact CorComp about getting this problem corrected if your expertise is insufficient to perform the recommended fix above.

FROM THE MAILBOX

JOY PAINT '99

By Great Lakes Software

The firm's letter and enclosed flyer announced this new Graphics program. The program is written completely in Assembly Language with all functions controlled by your joystick. The software is currently compatible with Star Micronics, Epson, TI Impact, TI 855 and other Epson compatible printers. The firm plans to have it compatible with the Prowriter, Axiom, and Okidata printer in the near future. You may contact the firm at:

Great Lakes Software Attn: Ernest Chandler P.O. Box 241 Howell, MI 48843

NO LABEL SYSTEM

By Weber And Sons

This firm has developed a No Label System to aid in identifying your disk collection. The system consists of a plastic sleeve that you attach to the diskette jacket and an insert which comes in several colors upon which you write the disk contents. The firm has added continuous form feed inserts (tractor feed on you printer) which are available in red or white at this time.

The firm also offers disk mailers manufactured from 200# corrugated board for $.65 each with a minimum order quantity of 20 mailers. The mailer holds up to three (3) 5 1/4" diskettes. The firm's address is:

Weber And Sons P.O. Box 104 Adelphia, NJ 07710 1-800-225-0044

ALBUM COLLECTION MANAGER

By Thomas J. Strang

This software is designed for the music enthusiast who wishes to maintain a list of records in their album collection. The program maintains records on the album title, album side, artist, purchase date, and purchase price. Inquiry options include artist, album title, group code (rock, jazz, country/western, pop, etc.), or all file information.

The program can store and maintain a maximum or 350 records per file. The program requires Extended Basic, 32k Memory Expansion, and Disk Drive. A printer is optional. The purchase price is $19.95. Contact the program's author at:

Thomas J. Strang 90-92 Bertholdi Ave. Jersey City, NJ 07305

99-CALC ELECTRONIC SPREADSHEET

By Phil Barnes

99-Calc is now available as freeware. This program is an electronic spreadsheet. You create your own spreadsheets using row and column titles, numerial values, and formulas. The program runs in Extended Basic. It is available on disk or cassette. Instructions are included on the storage media of your choice. A copy may be printed on your printer for easier reference. The spreadsheet is organized into 33 rows and 14 columns. The rows are identified by numbers and columns by letters. 99-Calc comes with a sample spreadsheet to help you get started.

Send a blank disk or cassette to the address below. Please include sufficient postage for the return of your disk or cassette. You may simply choose to forward $5.00 to cover the cost and a disk or cassette, mailer, and the necessary postage for its return to you will be covered. DO NOT forget to include your address and specify disk or cassette.

Phil Barnes 24631 Vis San Fernando Mission Viejo, CA 92692

SPELLBOUND

By Roberts Information Systems

SPELLBOUND is a new educational software that utilizes progressive instruction rather than arcade-style action to effectively and firmly teach correct spelling. An adjustable "challenge level" allows the program to be tailored to the indiviual student's abilities. The program accepts any word list and saves the list to disk or cassette for repeated use. Any words misspelled by the student are recalled after the list has been completed, and are repeated until the word is spelled correctly or repeatedly misspelled for ten times. The program maintains records on the successful attempts by each student and reports the information at the end of each session. When the student successfully completes the assigned list, SPELLBOUND engages the student in a "Learning Reinforcement" activity by scrambling the letters of each word in the list challenging the student to unscramble the words as they are displayed in random order. Correct answers are praised and errors are responded by displaying the correct spelling of the scrambled word. No score is maintained during the "Learning Reinforcement" activity. SPELLBOUND is designed for both classroom and home use requiring a minimum level of supervision.

The best news is that the firm is only requesting $14.95 for a copy of this software. No shipping and handling charges are listed in the mailer. SPELLBOUND is available on disk for the TI-99/4A, Apple II (+, e, c), Commodore 64, Kaypro (CP/M models), and the TRS-80 Model A.
The program is available on cassette for the TI-99/4A and Commodore 64. Order you copy for the company at:

Robins Info. Systems Inc. PO Box 666 Prineville, OR 97754 (503) 447-6275

GEMINI 10X UPGRADE CHIP AVAILABLE

A replacement upgrade chip is available for the Gemini 10X. The purchase of this chip will allow you to possess the Near Letter Quality print available on the newer Model SG-10. The price is $57.50 for the necessary chip. You will have to open your printer and look at the board to determine which of the two replacement chips are required. If you find a chip labeled D78016176, you need the 610M chip. If you have D78006, you need NLQ chip number 610. The new type NLQ font replaces the Italic font and is invoked by the same commands. You may purchase the chip from:

E.S.P. CORP. 7900 N. TAMIAMI TRAIL SARASOTA, FL 34243 (813) 355-6797

SUPER CARTRIDGE CHIP

The chip for making the Supercartridge is available from Texas Instruments by calling (806) 762-7457. The GROM number is CD 2204, the TI part number is 1015960-1204. The chip is the Editor-Assembler chip with a purchase price of $3.60 plus shipping and handling.

ODDS 'N ENDS

DISK COPYING ETHICS

By John Pearce - Front Range 99'ers - Colorado Springs, CO

Recently there has been some discussion about certain track copying programs which have appeared both locally and on Compuserve. It seems that certainpeople who are well known in the TI-99 world are upset about these track copying programs because it is now possible to copy disks like Millers Graphics Explorer and Advanced Diagnostics, Graphx, and DataBioTic's 4A/Talk. Seems to me I heard similar complaints about the disk copying program MASSCOPY when it first appeared. Likewise for NIBBLER, TURBO, and CLONE.

Is TRTRACK or TRACK-HACK so much different from the other copy programs? Copy programs **MUST BE USED** in a legitimate fashion! Have you ever purchased a copy-protected piece of software and wondered what would happen if the disk became defective? Essentially, you are at the mercy of the vendor unless you have one of these disk programs. It has been known for some time that a particular model Radio Shack computer with the proper software could copy most any TI-99 disk regadless of its protection. In fact, some of the club members have had their disks copied that way.

TRACK, et. al., now give us the ability to do that on the TI-99 itself. But should that give us or anyone the right to copy and handout copyrighted softwear? Certainly not! To repeat what I stated above, copy programs **MUST** be used only in legitmate ways. Copyright laws give us the right to have one backup copy. The copy programs make those copies.

Let us not have a rash of illegally copied software because it can easily be done. I have said many times before that if the software authors do not receive fair value for their products, they will stop creating software. For our orphan TI-99, that would certainly spell the end for the computer.

ARTICLES

The TI Forth Dimension

Jeff Stanford

"Decisions, Decisions"

In my last article, which appeared several issues ago, I put forth several examples of Forth's graphic capabilities in the multicolour mode and the bit map mode. I hope at least some of you found my examples interesting and hopefully useful but, how am I to know how well (or poorly) I am doing if no one writes to express their wants and needs, plus their praises and criticisms. So what I would like to see happen is for YOU, the readers of this article to become writers for a change. Write down your ideas on what you like and dislike or what you would like to see covered in future articles on Forth. Send them to me in care of THE NATIONAL NINTY NINER; 3535 So. H St. #26; Bakersfield, CA 93304. This is not to say that we of the NNN do not want your ideas on topics other than Forth but, to say I am primarily interested in what you think I can do to cover TI Forth better. Remember, any idea is useful and ideas are the basis of publications such as this newsletter. It is always nice to hear from our readers.

Now that I have spent my 60 seconds on the soap box, I can proceed with this article. As the sub-title implies, I plan to outline the decision making structures provided by TI Forth. Every language written for any computer has the ability to make a decision based on some event. For example, Basic uses GOTO, IF_THEN_ELSE, FOR_NEXT, ON_GOTO and ON_GOSUB statements to control what happens in a program. TI Forth is no exception, it also possesses a collection of decision making structures. These structures fall into three main groups: (1) if statements, (2) looping statements and (3) the case statement.

IF_ELSE_ENDIF statements work in a manner similar to the IF_THEN_ELSE in Basic. The general syntax looks like: "boolean_flag(1) IF true_part ELSE false_part ENDIF". The boolean_flag is simply a constant, a variable or an expression which has a value of either TRUE or FALSE. In Forth terms, a boolean_flag is an integer number or the result of some comparison or expression for which a zero value denotes FALSE and a non zero value denotes TRUE. In most examples found in books on Forth that I have read, the authors limit the values of boolean flags to just 1 (TRUE) and 0 (FALSE) to simplify matters. I will follow their example and do the same. The Forth word 'IF' tests this flag and determines if the true_part or the false_part will be executed. For example, we could define a new word called 'TEST' which takes a flag and if the flag is TRUE and another message if the flag is FALSE. Try typing in:

```
: IFDEMO IF CR ." Roses are red" ELSE CR ." Violets are blue" ENDIF ;
```

After getting the OK from the Forth compiler, use it by typing: 1 TEST 0 TEST. When the word TEST sees a TRUE flag, it will type out "Roses are red" and type out "Violets are blue" when the flag is FALSE. The Forth IF_ELSE_ENDIF is also similar to it's Basic cousin. The ELSE part is optional and the syntax could be condensed to: 'flag IF true part ENDIF' when no ELSE part is required. **NOTE:** Some Forth systems use THEN instead of ENDIF. TI Forth will accept either one as a valid ending for an IF statement. One final point concerning the IF statement. If you look back to my example, I had to define a new word (colon definition) to show how a IF_ELSE_ENDIF works. This is required by Forth because some of the internal branches have to be calculated. This is normally done while compiling a new word. Typing in a IF statement that is not enclosed in a definition will result in an error.

As you can see, the IF statement allows control of what is executed, but provides no way to repeat a section of the program more than one time. Forth is a structured language and does not have a GOTO statement defined. To get around this apparent deficiency, structured programming languages use defined loops to allow a section of program to be repeated any number of times. Loops can be divided into two main subgroups: (1) definite loops and (2)indefinite loops. The only real difference between these groups is the number of times they executed.

Definite loops are repeated a fixed number of times. The number of loops is determined by the command when the loop is started. The Basic FOR_NEXT loop is an example of this type statement where an index is set up to vary over a fixed range and step size. Forth has two similar types of definite loops which are:

DO_LOOP and DO_+LOOP

The only difference between these two is the DO_LOOP assumes a step size of one and DO_+LOOP allows a variable step size. The syntax for these loops are 'n2 n1 DO loop_contents LOOP' and 'n2 n1 DO loop_contents n3 +LOOP' where n1 is the first value of the loop and n2 is the last value of the loop plus one. In the DO_+LOOP, the value n3 can be a constant, a variable, or an expression and is the step size of the DO_+LOOP loop. Therefore the following two loop definitions would be identical:

```
: LOOPDEMO1
     11 1 DO loop_contents... LOOP
     11 1 DO loop_contents... 1 +LOOP;
```

Both of these loops will repeat ten times and then continue on with the rest of the program. Now, you may be wondering, "Where is the index variable because one is not defined in the loop?" Forth uses an interesting approach to this need. Rather than having explicit index variables which are defined when the loop is defined, Forth uses implicit indexes which reference the loop index relativly. The two loop indexer words TI Forth has predefined are 'I' and 'J'(2). 'I' is a function which returns in current index of the loop where it is used, 'J' does the same fuction except it gets the index of the loop that is one more nested level outward.

This may be a little confusing at first, but with practice it becomes easy. The main reasons I can see for this departure from 'normal' concepts may lie in Forth's stack usage nature and that it also eliminates the need to define a variable just to be an index. The main problem novice Forth programmers may run into is that since the indexes are referenced relatively, the values returned by 'I' and 'J' depend on where you use them. Look at the following example where I show two nested loops:

```
( a)  : LOOPDEMO2 11 1 DO
( first loop )
( b)                    ( do_something)
( c)                    I .
( d)                    11 1 DO
( second loop )
( e)
( do_something_else )
( f)                         I .  J .
( g)                         LOOP
( h)                    I .
( i)                    LOOP ;
```

Observe, the word 'I' is used in three places and 'J' is used in one place. On the first occurance of 'I' on line (c), only the first loop is active and, at that time, is the innermost loop. 'I' would return the value of the first loop's index. On the second occurance of 'I' and the occurance of 'J' on line (f), the second loop is now active and 'I' now returns the value of the second loop's index and 'J' returns the index of the first loop. 'I' of line (c) and 'J' would yield the same value. After the second loop is finished, the last ocurance of 'I' on line (h) would now again reference the index of the first loop.

NOTE: I also included this demo within a definition (LOOPDEMO1), this needs to be done for the same reason it was required by the IF_ELSE_ENDIF statement.

Another important word to use with DO LOOPs is LEAVE. It allows you to leave a loop before it finishes. In most cases it is found within an IF_ELSE_ENDIF statement which tests for some exit or error condition in which it would be beneficial to get out of the loop.

Indefinite loops require a slightly different perspective. Indefinite loops are defined as loops in which the number of times they will be repeated is not known. These types of loops consist of two parts: (1) the section of program to be in the loop and (2) a test to determine when the loop will terminate. TI Forth has two types of indefinite loops. Their forms are:

BEGIN_WHILE_REPEAT and BEGIN_UNTIL.

Both of these loops work in a similarly manner. They only differ in the placement of the exit test. BEGIN_WHILE_REPEAT does the test at the beginning of the loop while BEGIN_UNTIL does the test at the end of the loop. This means the BEGIN_UNTIL loop will always be executed at least one time whereas the BEGIN_WHILE_REPEAT loop may be skipped without being executed at all. The general syntax for these two loops is:

```
BEGIN boolean_flag WHILE loop_contents
REPEAT
BEGIN loop_contents ...  boolean_flag
UNTIL
```

The boolean_flag is defined the same way for the IF_ELSE_ENDIF statement. It can be the result of a constant, or an

expression which leaves a TRUE or FALSE flag. In the first indefinite loop shown above, the word which performs the test on the boolean flag is WHILE. As long as the flag is TRUE, the loop contents will be executed and then the word REPEAT loops back to the beginning of the loop designated by BEGIN. When the flag becomes FALSE, the program continues with the next statement following the word REPEAT. So, if the flag turns out to be FALSE initially, this loop will not be executed. In the other loop, the flag is tested by the word UNTIL which is done after the loop is executed the first time. UNTIL will loop back to the BEGIN statement until the flag is FALSE. When the flag changes to TRUE, the program continues on with the next statement following the word UNTIL. The following examples will demonstrate some of the ways these loops could be used:

```
0 CONSTANT Hell_Freezes_Over ( define
some constants )
1 VARIABLE Still_Hungry
--------------------------------------

: LOOPDEMO2

BEGIN Still_Hungry @ WHILE

                ( eat some FOOD! )

                ( if full, set
Still_Hungry to FALSE )

                REPEAT
--------------------------------------

BEGIN

     ( do something forever )

Hell_Freezes_Over UNTIL
;
```

As you can see, there is nothing complex about using loops. There is one more type of indefinite loop or the infinite loop. The BEGIN_UNTIL loop shown above is one way to write an infinite loop. Forth does have an indefinite loop predefined. It's syntax looks like this: 'BEGIN (do something forever) REPEAT'. Every time the program gets to the REPEAT it returns to the BEGIN and starts over.

The last control structure Forth has to offer is the CASE statement. The CASE allows you to select one action from a list of actions based on a single input. The syntax of the CASE looks like this:

```
n CASE
  case_1 OF ( action #1 ) ENDOF
  case_2 OF ( action #2 ) ENDOF
  case_3 OF ( action #3 ) ENDOF
  ...
  case_n OF ( action #n ) ENDOF
  ENDCASE
```

The input 'n' is typically an integer. When CASE is invoked, it is compared with each of the cases found before the OF's. When a match is found, the action for that case is then executed and the program continues with the next statement following the ENDCASE. If a match is not found among all of the choices, the program falls out of the bottom of the case statement and continues with the next statement following the ENDCASE (in some languages (like Pascal) this would result in an error). The CASE statement offers the power to choose while not being overly complex. For example, try doing the following example with only IF_THEN_ELSE's:

```
0 VARIABLE CMD  ( define a variable )
( accept a command and set CMD )
: CASEDEMO
        CMD @ CASE
              1 OF HONK ENDOF
              2 OF BEEP ENDOF
              3 OF CLS  ENDOF
              ENDCASE
;
```

When CMD is equal to one, the sound for a bad input is made. When CMD is equal to two, a beep is sounded. And when CMD is equal to three, the screen is cleared. The only problem I have with the Forth's CASE statement is that it does not have a provision to allow the matching part of the CASE to be a range of numbers. There is no otherwise clause to be a catch-all for cases which do not match any of the cases (this is useful for error trapping). I have seen some proposed enhancements for Forth's CASE statement which would include these features, but I have yet to see one which is claimed to be the standard.

In closing, all I have to say is with all of the power these control structures give you, who needs a GOTO! Until next time.

==========
footnotes :
==========

(1): A boolean flag, as I said above, is simply a constant, a variable or an expression which can only take on one of two

values. The two values TRUE and FALSE. TI Forth uses non-zero integers and zero to represent these two states. Boolean constants and variables work just the same as constants and variables do in decimal mathematics. In Forth, we can define them just like we do for their decimal counterpart. For example, we could define :

```
1 CONSTANT TRUE
0 CONSTANT FALSE

TRUE  VARIABLE FLAG_1
FALSE VARIABLE FLAG_2
```

What these four lines do is define two constants named TRUE and FALSE, and two variables, FLAG_1 and FLAG_2, which have values of TRUE and FALSE respectively. Boolean expressions are similar to their decimal cousins in the fact that they too can do calculations with constants and variables. They are also different in the type of operations you can do with these inputs. In decimal mathematics you use operators such as negation, %, /, +, and -. For boolean expressions you have a whole new set of operations. they are: NOT, AND, OR AND XOR (Forth does not have NOT predefined, type in ' 0= ; ' to add NOT to your system). The NOT operator negates an expression. For example, using the constants defined above, 'TRUE NOT' would be equal to FALSE (remember to use RPN for Forth) and conversely 'FALSE NOT' would be TRUE. NOT can also be used with variables like this: ' FLAG_1 @ NOT ' would equate to FALSE because the value of FLAG_1 is TRUE and ' TRUE NOT ' is FALSE. AND and OR are boolean operators which take two boolean inputs and returns single boolean result. The following table shows the relationship between inputs and output for these two operators:

INPUTS		AND	OR	XOR
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

As you can see, AND is only TRUE if both of the inputs are also TRUE. OR is TRUE if either or both of the inputs are TRUE. XOR is TRUE only if one input is TRUE. Another useful set of boolean operations are the logical comparisons which simply compare two numbers to each other and return a boolean flag. TI Forth has the following logical comparisons predefined: < (lesser than), = (equal), > (greater than), 0< (less than zero), 0= (equal to zero), and U< (unsigned lesser than). Some examples of their usage with answers are:

```
15 VARIABLE X ( DEFINE SOME VARIABLES
)
 23 VARIABLE Y
-------------------------------------
37  99  <       (TRUE)
33  32  =       (FALSE)
X @ Y @ >       (FALSE)
  0     0=      (TRUE)
 -3     0<      (TRUE)
```

These comparisons can be combined with the boolean operators to make complex expressions with equate to a single boolean value which can be used for some test. For example, if we wanted to test a variable Y and if it was greater than zero and less than or equal to 100 type out some message, you could write it as such using an IF_ENDIF statement:

```
Y @ 0 >    Y @ 100 > NOT    AND IF ."some_message" ENDIF
```

Y @ 0 > tests to see if Y is greater than zero, Y @ 100 > NOT tests to see if Y is less than or equal to 100 (not greater than is equal to less than or equal) and AND tests the two results to see if they are both TRUE and when this occurs the message will be typed out by the IF_ENDIF statement. There is no limit to the complexity of boolean, so feel free to experiment with them and learn.

(2): Forth only defines index words for the first two levels but it is possible to define other index words which retrieve further nested indexes. The general form of the CODE definition is :

```
HEX ( change input base to hex )
CODE ( some_name ) 0649 , C66E , N ,
045F ,
```

Where N is four times the level you want to retrieve less one. So if we wanted define a word to read the third level index it would look like this:

```
HEX
CODE K 0649 , C66E , 0008 , 045F ,
```

In this case N equals 4*(3-1) or 8. The assembly language mnuemonics for this definition are :

```
DECT SP          move stack pointer to make room for the new number to be placed there

MOV  @8(RP),*SP  move the selected index to the stack

B    *NEXT       return from the routine
```

SP, RP and NEXT are synonyms for registers R9, R14, and R15 respectively as defined by TI Forth manual (chapter 9 pg 3)

PASCAL NOTES

By Edgar Dohmann - JSC Users Group (JUG)

This month we will continue our study of the TI p-System Compiler by looking at some of the Compile-time Options. These options allow you to direct some of the Compiler's actions by placing directives in appropriate places in your source code.

The directives which control the Compile-time options are placed in your source code with "pseudo-comment" statements. Pseudo-comments are similar to ordinary comments in your source code but the first character following the opening comment indicator must be a dollar sign ($). Comments can be delimited by braces { } or by parentheses and asterisks (* *). A pseudo-comment compile-time directive then will have one of the following forms:

```
{$  } or (*$  *)
```

There are two types of compile-time options: switch and string. The switch options are a letter followed by a plus, minus, or caret. The string options are a letter followed by a text string. A pseudo-comment can contain any number of switch options separated by commas and one string option. If a string option is used, it must be the last (or the only) option in the pseudo-comment. The reason for this is that the comment delimiter is used to indicate the end of the string.

The following chart indicates the types of compile-time options:

OPTION	TYPE	DEFAULT	DESCRIPTION
B	STRING		BEGINS A CONDITIONAL COMPILE SECTION
C	STRING		INSERT COPYRIGHT NOTICE
D	STRING		DECLARE OR CHANGE VALUE OF A CONDITIONAL COMPILE FLAG
E	STRING		END A CONDITIONAL COMPILE
I	SWITCH	+	INPUT/OUTPUT CHECK CONTROL
I	STRING		INCLUDE A FILE
L	SWITCH	-	LISTING CONTROL
L	STRING		SPECIFY LIST FILE
P	SWITCH	+	PAGINATION CONTROL
Q	SWITCH	-	CONTROLS COMPILER OUTPUT TO SCREEN
R	SWITCH	+	RANGE CHECKING CONTROL
T	STRING		TITLE INSERTION
U	SWITCH	+	USER OR LIBRARY INDICATOR
U	STRING		USE SPECIFIED LIBRARY

The chart above contains two entries each for the I, L, and U options because these can be used either as string or as switch options. If the letter is followed by a plus, minus, or caret it will be treated as a switch option otherwise it will be treated as a string option.

These options are all described in detail in the Compiler manual for the TI p-System. Rather than repeat all of that information here, our discussion will be limited to a few of the more commonly used options.

The L options controls whether or not a list file will be generated. The default is L- which means a listing will not be generated unless you enter a pseudo-comment of L+. If a listing is generated, the default file is SYSTEM.LST.TEXT. If you want the listing to go to a different file, you will need to enter a different file name. The following example will turn on the list switch and specify a list file:

```
(*$L+,L DEMO.TEXT*)
```

The P option controls pagination when a listing is generated. P+ which is the default turns on pagination. P- will turn off pagination. A P without a plus, minus, or caret will start a new page in the listing.

The R option provides some range checking in the run-time program. R+ which is the default turns range checking on while R- turns it off. Programs compiled with R- run faster and require less memory space. However, R+ provides some valuable run-time error checking. Until a program is completely debugged, the use of the R+ option is highly recommended. If maximum speed and memory efficiency are desired, the program can always be recompiled with the R- option later.

The remaining options have valuable uses under some circumstances. Usually you will not need to change the default conditions of these other options until your programs become more sophisticated and larger. If you think you need to use these other options, be sure to read the Compiler manual for information.

TI-WRITER HELP - PART II

By Tom Kennedy
Compuserve ID# 74176,774

Now, I want to cover the Text Formatter which prints out the document. Most importantly, the special symbols, called Format Commands, that the formatter uses to alter the print-out of the document, which are installed in the Text Editor.

In other words, you put these commands into the text when you write it and as the formatter comes across them it changes the text accordingly but doesn't actually print the symbols.

There are six groups of formatter commands that are all applied in a similar manner. All commands must be in caps and must be on a line that starts with a period.

##

Text Dimension commands, as the name implies, move or shape the words in the document (margins, linespacing, right justify, etc.)

```
.FI   : FILL     : PUTS AS MANY WORDS ON A LINE AS WILL FIT.
.NF   : NO FILL  : CANCELS FILL.
.AD   : ADJUST   : ALIGNS THE TEXT TO THE LEFT AND RIGHT MARGINS (RT JUSTIFY).
.NA   : NO ADJUST: CANCELS ADJUST.
.LM n : LF MARGIN: SETS LEFT MARGIN TO "n".
.RM n : RT MARGIN: SETS RIGHT MARGIN TO "n".
.IN n : INDENT   : CREATES AN AUTO-INDENT FROM LEFT MARGIN.
.LS n : LINE SP  : SETS LINE SPACING TO "n" LINES.
.PL n : PG LENGTH: DEFINES NUMBER OF LINES TO A PAGE.
.BP   : BEGIN PG : DEFINES FIRST LINE OF NEW PAGE.
```

Internal Format commands control the spacing of characters on a line.

```
.SP n : SPACE    : SIMILAR TO THE TAB FUNCTION.
.CE n : CENTER   : CENTERS NEXT "n" LINES BETWEEN MARGINS.
```

Highlighting commands control functions such as underline or bold and allow you to redefine characters to use them to send CTRL codes to the printer.

```
 ^    : REQUIRED : JOINS WORDS TOGETHER WHEN REQUIRED TO PREVENT SPLITTING IN
      :   SPACE  :      REFORMATING, UNDERLINE, ETC.
 &    : UNDERLINE: (UNDERSCORE) UNDERLINES ALL TEXT FOLLOWING UNTIL NEXT PAGE.
 @    : BOLD     : (OVERSTRIKE) RETYPES FOLLOWING TEXT FOUR TIMES.
.TL xx: TRANS-   : ALLOWS REASSIGNMENT OF ONE CHARACTER TO REPRESENT A NUMBER.
      : LITERATE : OF CHARACTER VALUES TO SEND CODES TO THE PRINTER.
.CO t : COMMENT  : SIMILAR TO REM IN BASIC--ALLOWS NOTES THAT DO NOT PRINT.
```

Page identification commands print notes in the upper or lower corner of each page, either headers or footers.

```
.HE t : HEADER   : PRINTS TEXT (t) AND PAGE NUMBER AT TOP OF EACH PAGE.
.FO t : FOOTER   : PRINTS TEXT (t) AND PAGE NUMBER AT BOTTOM OF EACH PAGE.
.PA   : PAGE .   : RESETS PAGE NUMBER IN .HE AND .FO
```

File management commands include:

```
.IF f : INCLUDE  : MERGES A FILE TO PRINT A DOCUMENT TOO LARGE FOR ONE FILE.
      :  FILE    :
```

Mail Merge option commands are used to supply values to the variables in a letter that has been set up for the mail merge option

```
.ML f  :MAIL LIST: IDENTIFIES VALUE FILE (f) FOR MAIL LIST.
&n&    :VARIABLE : INSERTED IN TEXT AS VARIABLE FOR ASSIGNMENT FROM VALUE FILE
.DP n:t:DISPLAY  : PROMPTS YOU USING TEXT "t" TO ASSIGN TO VARIABLE (&n&).
       :  PROMPT :
```

##

The use of these commands in your text is what separates the word processor from a typewriter. They allow you to get the most out of your printer.

So, now you've written your document, and inserted all the format commands, now how do you print it out? First, save the document and exit the Text Editor. At the title menu, select Text formatter, (make sure the program disk is in the drive) and the screen will blank with the prompt "ENTER INPUT FILENAME". Enter the name of the file you just saved, (ex. DSK1.MYFILE) and hit enter.

Next, the prompt "ENTER PRINT DEVICENAME" appears after the file is loaded. If you use a serial printer, the device name would be RS232.BA=xxx with xxx being the baud rate. If you're using a parallel printer, the device name is PIO. Also, you must add either .CR or .LF to the end of the device name. This tells TI-Writer wether you're printer will handle the carriage return or the line feed. Check you're printer manual and the TI-Writer manual in detail to find out which you use.

The next prompt is "USE MAILING LIST". If you aren't printing "form letters" just hit enter to accept the default of N (NO).
Next is "WHAT PAGE(S)? (ALL). If you want to print the whole document, accept the default for all pages. Otherwise, you can print any of the pages or groups of pages.

The prompt "NUMBER OF COPIES: 1" tells how many copies of each page are to be printed.

The last prompt is "PAUSE AT END OF PAGE? N". The main purpose of this function is if you are using separate sheets of paper it will stop and wait for you to align the next sheet. Another use is to save a little paper. TI-Writer has an annoying habit of scrolling one whole blank page up before starting to print, which is not that big of a deal since what's one piece of paper worth considering how much you go through normally. But if you're just running test samples of type styles, or the like, you end up with a lot of white paper at you're feet. To prevent this, type "Y" and turn off you're printer. Now hit enter and turn the printer on, you should see "PRESS ENTER TO CONTINUE" (the software thinks one

page has been printed). If not, turn the printer on and off again. Now you align the paper to the top of the page and hit enter and the printing begins. But if it's a long letter, you'll have to sit there and hit enter after each page so usually it's better to select the default when using continuous feed paper.

Now, about the Mailing List Option. Let's say you've written a form letter to send out to various individuals, maybe a resume'. You write the letter like normal, but when you come to a name or address or something that will change with each letter, you put in it's place a variable in the form of #n#, where n is a number to identify the order. So instead of starting off with:"Dear Mr. Smith" you would have "Dear Mr. 1#" and so on. When you're all through with your letter, save it and purge the memory. Now you must create what is called a Value File, which is your mailing list where TI-Writer will draw the variables from. A value file consists of a list of values to be inserted into the letter, listed one to a line, preceded by the number of the variable and ending with a carriage return symbol. Groups of values must be separated by a line with just an asterisk and a carriage return. For example:

```
1 John Smith
2 123 STREET
3 Seattle, WA
*
1 Jane Doe
2 456 STREET
3 Seattle, WA
```

At the top of your letter you insert the .ML f command where f equals the filename of your value file. After selecting the mailing list option the computer will use this command to fill in the variables. If there is no .ML command in the letter then when you are prompted for "MAILING LIST NAME:" you supply the filename. This allows you to call on a number of files for different groups.

Another way to insert values is to use the Define Prompt command. With this command you do not insert a .ML command calling a value file. Instead you insert lines containing the format: .DP n:t - where n is the number of the variable and t is the prompt text. Now, when you come to the prompt "USE MAILING LIST?" you select "N" for NO. As the document is printed, a variable is encountered and the printing stops. The text you chose appears on the screen requesting that you input the appropriate value. If you don't include a ".DP n:t" command in your text, the computer responds with "ENTER DATA FOR VARIABLE #n#". It can get confusing trying to remember which item you're on. This method is handy for letters where you only desire to print one copy at different times to different people.

Let me tell you, this is why I bought a computer. I'm sure we all went through that period of time before buying a computer when we would ask: "what am I going to use a computer for, anyway?". Well I decided there were two things I wanted to do: 1) Store files of data (recipes, albums, Etc.) and 2) Use my computer as a typewriter. I didn't know about TI-WRITER when I bought the 99/4A, but now I know that I made the best choice possible. I hope you will all find TI-WRITER as easy to use and as powerful as I have.

REVIEWS

SUPERBUG II - Version 2.0

By Edgar L. Dohmann

Thanks to a lot of help from Ewell Brigham of the Houston User's Group, version 2.0 of SUPERBUG II will soon be available. I am hoping for a May 1, 1986 release date if I can get the manual updated and printed by then.

Two major new features of ver 2.0 are the ability to load and save program files. The load feature will operate in a similar fashion to option 5 of the Editor/Assembler. However, if you want to load the program into a different area from its default location, this can be done with the new version of SUPERBUG II.

The save program file is similar to the SAVE utility on the disk supplied with Editor/Assembler. However, with SUPERBUG II, the SFIRST, SLAST, and SLOAD labels do not have to be DEFined in the object file. You supply the starting and ending address when you activate the save function and it saves the program in the same way the SAVE utility does.

Other changes from version 1.0 are mainly cosmetic but a few remaining bugs are also fixed. The remaining changes include:

1) The J command is changed so the border colors are also changed when screen colors are changed.

2) A bug in the M and D commands is fixed. This bug only showed up when memory dumps crossed address >8000.

3) A bug in the D command to an external device is fixed. This bug only showed up when VDP or GROM memory was dumped to an external device.

4) The Q and E commands have been improved, especially for the SUPER SPACE version of the program.

5) The small character sets are automatically loaded when the program is started up from Console BASIC or from SUPER SPACE.

6) The Console BASIC startup now works like the Extended BASIC startup. In version 1.0, the initial prompts were not visible from Console BASIC.

7) The leading zero is removed from registers R0 through R9 in the disassembler. This allows the code produced to be easily reassembled.

8) A bug in disasseably of JMP instructions is fixed so the operand value will reasseable properly.

9) A aodification for writing to GRAM is added for compatibility with GRAM KRACKER.

There are still a few additional features I would like to add to the prograa but it is a full 8K in size at the present time and that is all that will fit into SUPER SPACE. I aa trying to do a little aore code squeezing and if I can save enough space, there aay be a few aore goodies added. Any such additional features aay have to wait for version 3.0 however.

The distribution disk for version 2.0 will not have a complete aanual on the disk as with version 1.0. For one thing, with the additional information to describe new features, the complete aanual will not fit. In the second place, I feel that the aanual-on-a-disk feature was an experiaent that did not work out well.

When I distributed SUPERBUG II through public domain FAIRWARE, I requested a contribution of $3.00 to $5.00 froa anyone who got a copy in this aanner. I also proaised a printed copy of the aanual to anyone who sent at least $5.00. So far I have only received ONE contribution in this aanner. I know that a large aumber of User's Groups have a copy of ay prograa in their library and it's hard to believe that only one person has obtained a copy in this aanner. I assuae that because the disk was totally self-sufficient that aost people don't see the need to send a contribution and get a printed copy of the aanual.

On the other hand, I have received over 150 orders froa publicity generated by MICROpendiua. I have also sold about 20 copies through authorized distributors at various /4A fairs. Only because of this response have I had the initiative to coaplete version 2.0. Since those people who order a copy froa ae directly also get a printed copy of the aanual, there is no real need for the aanual on the disk.

The new disk will have a short help file that will explain how to load the prograas and will offer a printed copy of the aanual to anyone who sends ae a contribution of $5.00 or aore. People who order copies froa ae directly will continue to receive a printed copy of the aanual.

I will be continuing the saae ordering policy that was in effect for version 1.0. Anyone aay receive a copy of the prograa with a aanual if they send ae $10.00 or if they send ae a disk, aailer, return postage, and $5.00.

Anyone who already has version 1.0 and wants to upgrade to version 2.0 will have to follow the saae procedure used to obtain a copy the first tiae. Since the price I charge is so ainiaal, I cannot afford any other type of upgrade service.

I sincerely appreciate the orders I have received and the ONE public doaain contribution. Many people have followed up their initial correspondence with aore letters. I really enjoy the opportunity to get to know other users around the country better and appreciate all the suggestions and coaaents that were sent. I hope that version 2.0 of SUPERBUG II will also be well accepted.

HINTS_'N_TIPS

QUIET, AT LAST!!!!

By L. R) Livergood

After reading about a quiet replaceaent fan available for the TI PE box, I decided to invest in one ayself. I purchased the E6 & 6 ROTRON SU2J7 Sprite Fan supplied by STATCO Inc. (P.O. Box 145; Townsend, MA 01469-0145). The fan was delivered within a week and had all the instructions required for this specific installation. Within one hour after I started, I had the fan installed and tested.

The results are very iapressive. At first, I could not even tell it was working. My iaaediate thought was that it aight not be aoving enough air, however, a quick look at the statistics supplied with the fan indicate that it possesses a higher efficiency design than the one TI used (115v - 6.9 Watts - 27 CFM vs. 115v - 11.0 Watts - 22 CFM).

The fan sells for $15.50 plus $2.50 for shipping. Using soae coaaon sense and care, I found the installation to be fairly siaple. Assuaing you have reaoved the internal internal cards and disk drive. All that is required is to reaove 20 or so screws that hold the outer shell of the PE Box in place. The nuts holding the existing fan can be reaoved with a pair of pliers (they were too saall for ay 1/4" rachet set). By tracing the wires froa the fan, you can easily locate the quick disconnects.

There are two items I aust infora each of you about. I purchased a package of solderless quick disconnects (Radio Shack No 64-3049) which were recoaaended by soaeone who also perforaed this aodification. As it turned out, they were partly coapatible. The plastic jacket had to be cut back because it would not fit around the one TI supplied. After attaching both leads, I wrapped thea electrical tape.

The other item to watch is that the lower left-hand nut is hard to reach. It required reaoving two aore screws which hold a circuit board to the chassis of the PE Box. This provides a bit aore room to reach the nut.
Could this aodification possibly affect the operation of the PE Box? I aa not an electronic expert, but ay feeling is that it will not affect its operation. The aain reason is that the fan is connected to the line side of the power supply. Unless you happen to cross up soae wiring during installation, there should not be any change to the DC output.

TIPS FROM THE TIGERCUB

#35

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213

Distributed by Tigercub Software to TI-99/4A Users Groups for promotional purposes and in exchange for their newsletters. May be reprinted by non-profit users groups, with credit to Tigercub Software.

Over 130 original programs in Basic and Extended Basic, available on casette or disk, only $3.00 each plus $1.50 per order for PPH.

Entertainment, education, programmer's utilities.
Descriptive catalog $1.00, deductable from your first order.
Tips from The Tigercub, a full disk containing the complete contents of this newsletter Nos. 1 through 14, 50 original programs and files, just $15 postpaid.
Tips from the Tigercub Vol. 2, another diskfull, complete contents of Nos. 15 through 24, over 60 files and programs, also just $15 postpaid. Or, both for $27 postpaid.
Nuts & Bolts (No. 1), a full disk of 100 Extended Basic utility subprograms in merge format, ready to merge into your own programs. Plus the Tigercub Menuloader, a tutorial on using subprograms, and 5 pages of documentation with an example of the use of each subprogram. All for just $19.95 postpaid.
Nuts & Bolts No. 2, another full disk of 100 utility subprograms in merge format, all new and fully compatible with the last, and with 10 pages of documentation and examples. Also $19.95 postpaid, or both Nuts Bolts disks for $37 postpaid.
Tigercub Full Disk Collections, just $12 postpaid! Each of these contains either 5 or 6 of my regular $3 catalog programs, and the remaining disk space has been filled with some of the best public domain programs of the same category. I am NOT selling public domain programs - my own programs on these disks are greatly discounted from their usual price, and the public domain is a FREE bonus!
TIGERCUB'S BEST
PROGRAMMING TUTOR
PROGRAMMER'S UTILITIES
BRAIN GAMES
BRAIN TEASERS
BRAIN BUSTERS!
MANEUVERING GAMES

ACTION GAMES
REFLEX AND CONCENTRATION
TWO-PLAYER GAMES
KID'S GAMES
MORE GAMES
WORD GAMES
ELEMENTARY MATH
MIDDLE/HIGH SCHOOL MATH
VOCABULARY AND READING
MUSICAL EDUCATION
KALEIDOSCOPES AND DISPLAYS

For descriptions of these send a dollar for my catalog!

The April Micropendium had a rather slow routine to count the number of words in a D/V text file. I think the following will be much faster. It ignores any lines beginning with a period (TI-Writer formatter commands), otherwise counts each cluster of characters followed by a space, plus the last cluster on the line.

```
10 !WORDCOUNT by Jim Peterso
n
100 DISPLAY AT(12,1)ERASE AL
L:"INPUT FILENAME? DSK" :: A
CCEPT AT(12,20):F$ :: OPEN #
1:"DSK"&F$,INPUT
110 A=1 :: LINPUT #1:M$ :: I
F ASC(M$)=46 THEN 130
120 X=POS(M$," ",A):: IF X=0
 THEN 130 :: IF X=A THEN A=X
+1 :: GOTO 120 ELSE F=1 :: C
=C+1 :: A=X+1 :: GOTO 120
130 C=C+F :: F=0 :: IF EOF(1
)<>1 THEN 110 :: CLOSE #1 ::
 DISPLAY AT(12,1)ERASE ALL:"
APPROXIMATELY "&STR$(C)&" WO
RDS"
```

Have you tried those black write-protect tabs, made of a material similar to electrical tape? They do not become dog-eared from bumping against the drive slot, and do not leave the disk sticky when you remove them.

```
100 !TIGERCUB GRAPHPRINT by
Jim Peterson
110 !Will output to printer
a line graph of 31 items of
data, as for instance the
temperature for each day of
a month
120 !Values must be positive
 integers within a range of
75 from minimum to maximum
130 M$=RPT$("!_",65):: DIM T
$(31),D$(75):: MN=10000
140 DISPLAY AT(12,1)ERASE AL
L:"Input data - maximum 31":
"items. Enter to finish"
150 FOR X=1 TO 31 :: DISPLAY
 AT(14,1):X;TAB(4);CHR$(1)::
 ACCEPT AT(14,4)VALIDATE(DIG
IT)SIZE(-5)BEEP:T$(X):: IF T
$(X)=CHR$(1)THEN X=X-1 :: GO
TO 170
160 T=VAL(T$(X)):: MX=MAX(MX
,T):: MN=MIN(MN,T):: NEXT X
170 RN=MX-MN :: IF RN>75 THE
N PRINT "EXCEEDS MAXIMUM RAN
GE OF 75" :: STOP
180 IF MX>75 THEN AD=MX-75
190 OPEN #1:"PIO",VARIABLE 1
32 :: PRINT #1:CHR$(15);CHR$
(27);CHR$(51);CHR$(12):: PRI
NT #1:RPT$("_",132)
200 DISPLAY AT(12,1)ERASE AL
L:"Wait, please...": :".....
.this takes time"
210 LM=LEN(STR$(MX)):: FOR J
=1 TO 75 :: J$=STR$(76+AD-J)
220 IF J>66+AD THEN J$=J$&"
"
230 IF J/2=INT(J/2)THEN D$(J
)=RPT$(" ",LM)&SEG$(M$,1,132
-LM)ELSE D$(J)=J$&SEG$(M$,1,
132-LM)
240 NEXT J :: PRINT #1:RPT$(
" ",LM)&SEG$(M$,1,132-LM)
250 J=1 :: T=VAL(T$(J))-AD ::
 T=76-T :: D$(T)=SEG$(D$(T)
,1,J#4+4)&CHR$(239)&SEG$(D$(
T),J#4+6,255):: J=J+1
260 T2=T :: T=VAL(T$(J))-AD
:: T=76-T :: FOR N=T2 TO T S
TEP (T2)T)+ABS(T)=T2):: D$(N
)=SEG$(D$(N),1,J#4+2)&CHR$(2
55+(T(T2))&SEG$(D$(N),J#4+4.
255):: NEXT N
270 J=J+1 :: D$(T)=SEG$(D$(T
),1,J#4)&CHR$(239)&SEG$(D$(T
),J#4+2,255):: IF J<=X THEN
260
280 FOR J=1 TO 75 :: PRINT #
1:D$(J):: NEXT J :: PRINT #1
290 T=8 :: FOR J=1 TO 31 ::
PRINT #1:TAB(T);STR$(J);:: T
=T+4 :: NEXT J
```

When you are analyzing an Extended Basic program, or modifying it, it is often easier to work with single-statement lines. This program will break all multi-statement lines into single-statement lines, except when they are followed by IF or ELSE. When you are finished modifying, a Compactor or Smash program can be used to compact it again.

```
100 !DECOMPACTER by Jim Pete
rson
110 DISPLAY AT(3,5)ERASE ALL
:"TIGERCUB DECOMPACTER": :"
Program must first be -": :"
RES 100,100": :"SAVE DSK(fil
ename),MERGE"
120 DISPLAY AT(12,1):"INPUT
FILENAME?":"DSK" :: ACCEPT A
T(13,4):IF$
130 DISPLAY AT(12,1)ERASE AL
L:"OUTPUT FILENAME?":"DSK" :
: ACCEPT AT(13,4):OF$
140 OPEN #1:"DSK"&IF$,INPUT
,VARIABLE 163 :: OPEN #2:"DS
K"&OF$,OUTPUT,VARIABLE 163 :
: LN=100
150 LINPUT #1:M$ :: P=POS(M$
,CHR$(130),3):: IF P=0 THEN
PRINT #2:M$ :: GOTO 270
160 A$=SEG$(M$,1,P-1):: IF P
OS(A$,CHR$(129),1)<>0 OR POS
(A$,CHR$(132),1)<>0 THEN PRI
```

```
NT #2:M$ :: GOTO 270
170 PRINT #2:A$&CHR$(0)
180 AN=LN+1 :: GOSUB 280
190 M$=SEG$(M$,P+1,255)
200 P=POS(M$,CHR$(130),1)
210 IF P=0 THEN PRINT #2:LN$
&M$ :: GOTO 270
220 A$=SEG$(M$,1,P-1)
230 IF POS(A$,CHR$(129),1)<>
0 OR POS(A$,CHR$(132),1)<>0
THEN PRINT #2:LN$&M$ :: GOTO
 270
240 PRINT #2:LN$&A$&CHR$(0)
250 AN=AN+1 :: GOSUB 280
260 GOTO 190
270 LN=LN+100 :: IF EOF(1)<>
1 THEN 150 ELSE CLOSE #1 ::
CLOSE #2 :: END
280 LN$=CHR$(INT(AN/256))&CH
R$(AN-256*INT(AN/256)):: RET
URN
```

I still think of the TI as a HOME computer, and I still think that the home computer is an invaluable educational tool - but I guess not many folks agree with me. I had thought of writing full disks of a progressive series of lessons on one subject, but my present two full disks of math education have sold a combined total of 7 copies in 7 months, so that would obviously be a waste of time.

I had written this next program for that purpose and I guess it's no use wasting it, so -

```
100 CALL CLEAR :: CALL TITLE
(5,"TAKE AWAY")!by Jim Pete
son
110 DISPLAY AT(3,10):"COPYRI
GHT":TAB(10);"TIGERCUB SOFTW
ARE":TAB(10);"FOR FREE":TAB(
12);" DISTRIBUTION":TAB(11);
"SALE PROHIBITED"
120 CALL PEEK(-28672,A@):: I
F A@=0 THEN 150
130 DATA FINE,NO,GOOD,UHOH,R
IGHT,TRY AGAIN,YES,THAT IS N
OT RIGHT
140 FOR J=1 TO 4 :: READ RIG
HT$(J),WRONG$(J):: NEXT J
150 FOR D=1 TO 1000 :: NEXT
D :: CALL DELSPRITE(ALL)
160 CALL CLEAR :: CALL CHAR(
95,"FFFF"):: CALL MAGNIFY(2)
:: RANDOMIZE :: CALL SCREEN(
14):: FOR SET=5 TO 8 :: CALL
 COLOR(SET,16,1):: NEXT SET
170 CALL CHAR(120,"E70042001
8007E0000E700420099423CE7004
20099423C00E700421800 3C4200"
)
180 CALL CHAR(124,"0E0004010
00708007000208000E01000")
190 DISPLAY AT(3,10):"TAKE A
WAY" :: CALL CHAMELEON
200 CALL COLOR(14,2,2):: CAL
L HCHAR(4,4,143,2):: CALL HC
HAR(5,4,143,2):: CALL SPRITE
(#25,120,11,25,25)
210 T=T+1 :: N=1-(T>5)-(T>15
):: G=10-(T>5)*80-(T>15)*810
 :: H=0-(T>5)*10-(T>15)*90
220 X=INT(6*RND+H):: Y=INT(6
*RND+H):: IF Y>X THEN TT=X ::
 X=Y :: Y=TT
230 IF X=X2 OR Y=Y2 THEN 220
 :: X2=X :: Y2=Y :: Z=X-Y
240 GOSUB 250 :: GOTO 210
250 GOSUB 260 :: GOSUB 280 :
: GOSUB 310 :: FOR D=1 TO 20
0 :: NEXT D :: CALL DELSPRIT
E(ALL):: DISPLAY AT(18,1)::
CALL CHAMELEON :: CALL SPRIT
E(#25,120,11,25,25):: RETURN
260 FOR J=1 TO LEN(STR$(X)):
: :: A(J)=VAL(SEG$(STR$(X),J
,1)):: NEXT J :: FOR J=1 TO
LEN(STR$(Y)):: B(J)=VAL(SEG$
(STR$(Y),J,1)):: NEXT J
270 FOR J=1 TO LEN(STR$(Z)):
: C(J)=VAL(SEG$(STR$(Z),J,1)
):: NEXT J :: M=LEN(STR$(Z))
-LEN(STR$(X)):: RETURN
280 R=96 :: CC=96 :: FOR J=1
 TO N :: CALL SPRITE(#J,48+A
(J),11,R,CC):: CC=CC+16 :: N
EXT J
290 R=116 :: CC=96 :: FOR J=
1 TO N :: CALL SPRITE(#4+J,4
8+B(J),11,R,CC):: CC=CC+16 :
: NEXT J
300 CALL HCHAR(18,12,95,N*3)
:: CC=CC-16 :: RETURN
310 R=140 :: FOR J=LEN(STR$(
Z))TO 1 STEP -1 :: IF LEN(ST
R$(X))=1 THEN M=CC :: GOTO 3
30
320 FOR M=CC TO CC+8 :: CALL
 LOCATE(#J-N,96,M,#J+4-N,116
,M):: NEXT M
330 IF A(J-N)>B(J-N)THEN 36
0 :: CALL SPRITE(#20,49,16,9
6,M-9)
340 IF F3=1 THEN 360 :: F1=1
 :: A(J-N-1)=A(J-N-1)-1 :: I
F A(J-N-1)<0 THEN A(J-N-1)=9
 :: F2=1 :: A(J-N-2)=A(J-N-2
)-1
350 CALL SPRITE(#22,48+A(J-N
-1),16,80,M-24):: IF F2=1 TH
EN CALL SPRITE(#21,48+A(J-N-
2),16,80,M-40)
360 CALL SPRITE(#27,45,16,11
6,M-12)
370 CALL SPRITE(#20,63,11,K,
M)
380 CALL KEY(3,K,ST):: IF ST
<1 OR K<48 OR K>57 THEN CALL
 PATTERN(#20,32):: CALL PATT
ERN(#20,63):: GOTO 380
390 CALL DELSPRITE(#20,#28):
: CALL SPRITE(#12+J,K,11,R,M
)
400 IF K-48<>C(J)THEN GOSUB
450 :: CALL DELSPRITE(#12+J)
:: F3=1 :: GOTO 330
410 CALL DELSPRITE(#27):: IF
 F1=1 THEN 420 ELSE IF F2=1
THEN 430 ELSE 440
420 F1=0 :: CALL DELSPRITE(#
J-N-1):: FOR P=80 TO 96 :: C
ALL LOCATE(#22,P,M-24):: NEX
T P :: CALL SPRITE(#J-N-1,48
+A(J-N-1),16,96,M-24):: CALL
 DELSPRITE(#22):: GOTO 440
430 F2=0 :: CALL DELSPRITE(#
J-1-N):: FOR P=80 TO 96 :: C
ALL LOCATE(#21,P,M-24):: NEX
T P :: CALL SPRITE(#J-1-N,48
+A(J-1-N),16,96,M-24):: CALL
 DELSPRITE(#21)
440 CC=CC-16 :: NEXT J :: GO
SUB 480 :: F3=0 :: RETURN
450 DATA 123,124,125,123,124
,125,123,120
460 IF A@=0 THEN 470 :: CALL
 SAY(WRONG$(INT(RND*4+1)))
470 RESTORE 450 :: FOR JJ=1
TO 8 :: READ P :: CALL PATTE
RN(#25,P):: XX=2^250 :: NEXT
 JJ :: RETURN
480 DATA 121,122,121,122,121
,122
490 IF A@=0 THEN 500 :: CALL
 SAY(RIGHT$(INT(4*RND+1)))
500 RESTORE 480 :: FOR JJ=1
TO 6 :: READ P :: CALL PATTE
RN(#25,P):: XX=2^250 :: NEXT
 JJ :: RETURN
510 SUB CHAMELEON
520 M$="1800665AC342DB667E18
8100995AC3A5E78142BD24DB6600
8142992400 7E5AC3A53C241800FF
DB5AFF7EFF00991881006600 18"
530 RANDOMIZE :: CALL CHAR(1
20,SEG$(M$,INT(43*RND+1)*2-1
,16)):: X=INT(14*RND+3)
540 Y=INT(14*RND+3):: IF Y=X
 THEN 540 :: CALL COLOR(13,X
,Y)
550 CALL HCHAR(1,2,128,30)::
 CALL HCHAR(24,2,128,30):: C
ALL VCHAR(1,31,128,96):: SUB
END
560 SUB TITLE(S,T$)
570 CALL SCREEN(S):: L=LEN(T
$):: CALL MAGNIFY(2)
580 FOR J=1 TO L :: CALL SPR
ITE(#J,ASC(SEG$(T$,J,1)),J+1
-(J+1=S)+(J+1=S+13)+(J>14)*1
3,J*(170/L),10+J*(200/L))::
NEXT J
590 SUBEND
```

When you give your printer instructions, it remembers them until you turn it off. That is why you may find that your letter to Aunt Sally is being printed in double width underlined italics. The solution is found in another gobbledegook paragraph in the Gemini manual - "when (ESC "@") is sent to the printer, the conditions of the printer are initialized."

In plain English,

OPEN #1:"PIO" :: PRINT #1:CHR$(27);"@" in your program or CTRL U, FCTN R, CTRL U, SHIFT 2 at the beginning of your TI-Writer text will cancel out any special orders the printer is still remembering and return it to its default conditions.

Here's a bright idea by Scott King in the AVTI U6 newsletter. When you load a program in order to modify it, put a reminder of its filename in the first line, such as 1 ! SAVE DSK1.NAME . Then, when you are ready to save it, just list line 1, FCTN 8, use the space bar to erase the 1 !, and Enter.

MEMORY FULL!

Jim Peterson

SUBSCRIPTION FORM
FOR "THE NATIONAL NINTY-NINER"
PUBLISHED BY THE 99'ERS ASSOCIATION

NAME:_____ DATE:_____

ADDRESS:_____

CITY:_____STATE_____ZIP _____

SUBSCRIPTION TYPE	AMOUNT	CHOICE
THIRD CLASS - BULK RATE	$12.00	_____
FIRST CLASS - US AND CANADA	$17.00	_____
FIRST CLASS - OVERSEAS	$22.00	_____

PLEASE MAIL CK./M.O. FOR SUBSCRIPTION CHOICE SELECTED ABOVE TO:

THE 99'ERS ASSOCIATION
ATTN: LUCI VEITH
3535 SO. H ST., #26
BAKERSFIELD, CALIF. 93304

6/85

THE NATIONAL 99'ER
3535 SO. H ST., #26
BAKERSFIELD, CA 93304

BULK RATE
U. S. POSTAGE
PAID
Permit No. 787
Bakersfield, Calif.