Creative Programming for Young Minds

. . . on the TI-99/4A ™

Volume V

CREATIVE
PROGRAMMING INCORPORATED
A SUBSIDIARY OF R.V. WEATHERFORD CO.

CREATIVE Programming for Young Minds

CREATIVE Programming for Young Minds didn't just happen. It represents the harvested fruit of an idea planted several years ago by Dr. Henry A. Taitt. He saw the pressing need for an enrichment program for young children that would help prepare them for the future they would be instrumental in shaping.

It was cultivated by Marilyn Buxton, whose deep interest in early childhood learning enabled her to find ways to teach primary children to program microcomputers.

It was fertilized by Devin Brown, with his lively wit and creative writing style. He gave it the nutrients it needed to appear in printed form to be shared. His shadow is cast over most of the later authors who patterned their style and examples after his original writings.

It was cared for by Howard Smith, Charles Miller, George Kolopanis, Alverta Darding, Lea Ann Hummel, Robin Koch and others, who worked with it in the lab helping to remove the bugs that would stunt its growth.

It was harvested by Nancy Taitt, Marilyn Hoots, Wayne Owens, Diane ZuHone, and others who typed and phoned and talked with people to spread the word and create a market for the final fruit.

And most important of all were the CHILDREN who tried and tested the materials that were produced. They shared their likes and dislikes, and made certain that everything that was included could be done by young minds.

These books were not created by a publisher to be sold to schools, where they would be used on children. They were instead, created from the successes of children, edited by the concerns of parents, and then offered to anyone that wishes to enrich the minds of young children.

If you elect to use these materials, then you assume the responsibility to encourage independent thought, reward creativity, enhance reasoning and logic, and above all, be forever open to alternate ways to solve problems.

If you do this, your own rewards will be found in the faces of the children you serve.

CREATIVE
PROGRAMMING INCORPORATED
A SUBSIDIARY OF R.V. WEATHERFORD CO.

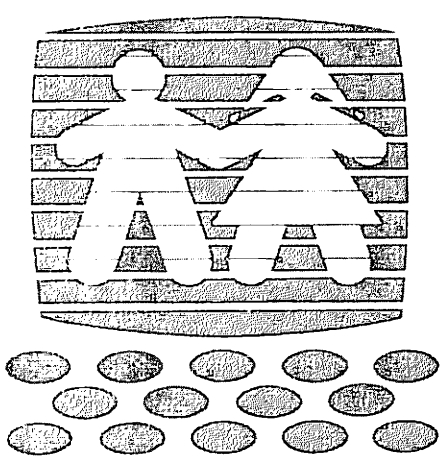
(217) 348-1451

Creative Programming for Young Minds

. . . on the TI-99/4A ™

Volume V

by Leonard Storm



® A registered trade mark of Texas Instruments, Inc.

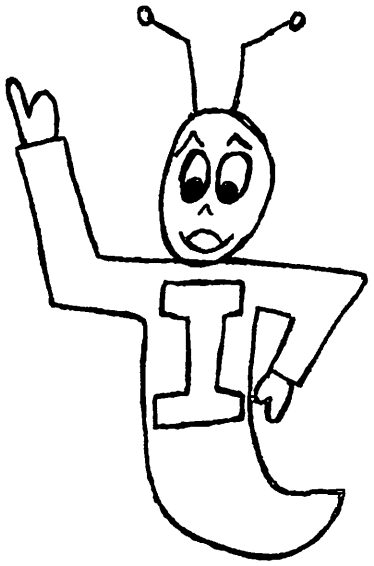
CREATIVE PROGRAMMING FOR YOUNG MINDS

... ON THE TI/99-4A

VOLUME V

T A B L E O F C O N T E N T S

LESSON #17: DEBUGGING



GREETINGS, TI PROGRAMMERS! WELCOME TO VOLUME V, LESSON 17. IN THIS LESSON, YOU WILL LEARN HOW TO GET RID OF PROGRAM BUGS. PROGRAM BUGS ARE ERRORS IN PROGRAM LOGIC OR SYNTAX WHICH CAUSE A PROGRAM TO MISBEHAVE. TRACE AND BREAK COMMANDS CAN BE USED TO ELIMINATE THESE NASTY BUGS.

The TRACE command lists program statement numbers on the screen as they are executed. This allows you to see the order in which statements are being executed and may help you to find an unwanted error.

Type in the following program and observe the effect of the TRACE command. Let the NUM command number the statements for you.

```
10 FOR I=1 TO 10
20 PRINT I;
30 NEXT I
40 PRINT "END OF LOOP"
50 STOP
```

Type in TRACE.

Then RUN the program.

Notice that the number of the statement executed is printed on the screen just before the statement itself is executed. Also, notice that PRINT statements still cause data to be printed on the screen.

Try RUNning the program again. Notice that the TRACE command is still in effect.

To get rid of the TRACE command just type in the UNTRACE command. Do it, then RUN the program again.

Now change statement 20 to:

```
20 PRINT I:
```

Next, type in TRACE and then RUN the program again. Notice the effect of the PRINT separator on the screen listing.

TRACE and UNTRACE commands may also be used as statements in a program. To illustrate this, type in the following additional statements:

```
1 TRACE
31 UNTRACE
```

Next, type in the command UNTRACE and then RUN the program again. Notice that statement 1 is not listed because the TRACE command is not yet in effect until after statement 1 is executed. Statements 40 and 50 are not listed because statement 31 undoes the TRACE command.

Now delete statement 31 and add the following statement to your program:

```
25 UNTRACE
```

Before you RUN the program again, try to predict what statement numbers will be printed out on the screen. Write your predictions on the line below and then RUN the program. _____

Next, type in NEW and press ENTER. The NEW command causes the TRACE command to be erased.

The BREAK command is another useful feature of TI BASIC which allows you to debug programs. Let's see how this command can be used to help find program logic errors.

Type in the following program:

```
10 CALL CLEAR
20 INPUT "INPUT A NUMBER THAT IS A MULTIPLE OF 3":N
30 K=N
40 N=N/3
50 C=C+1
60 IF N=1 THEN 80
70 GOTO 40
80 PRINT "3 TO THE POWER";C;"=";K
90 FOR D=1 TO 2000
100 NEXT D
110 GOTO 10
```

Now RUN the program. When it asks for a number, type in 9.

Did you get a message from the computer saying:

3 TO THE POWER 2 = 9

That is, 3 x 3 = 9.

Now type in the number 9 again. What response do you get this time? _____

The last response is obviously wrong. The value of C in statement 80 somehow got messed up. Here's what we'll do. Let's keep a close eye on the value of C as the program runs to see where the value of C goes buggy.

Type in the following statements:

35 BREAK

55 BREAK

Now RUN the program again. Again, type in 9 when the program asks for a number.

As the program continues, it encounters statement 35 which tells the computer to stop. The following message is printed on the screen:

*BREAKPOINT AT 35

Now we can take a peek at the value of C. Type in the following command:

PRINT C,N (ENTER)

The computer should print a 0 and a 9. That is, COUNT = 0 and N = 9.

Everything is O.K. so far since statement 20 makes N equal 9 and the value of C equals zero since it has not yet been given a value. (Remember, the computer initially sets all variables to zero when you type RUN.)

What is the value of K at this breakpoint (statement 35)? Check your answer by typing:

```
PRINT K
```

Now let's allow the program to continue where it left off. Type:

```
CONTINUE
```

and press ENTER. The BREAKPOINT at 55 causes the program to stop once again. At this time, N has been divided by 3 in statement 40 so that N = 3. And C has been incremented in statement 50 so that C = 1. The variable, C, keeps track of how many times the original number has been divided by 3.

Check that the above values are correct by typing in the following command:

```
PRINT C;N;K                    (ENTER)
```

Now type:

```
CONTINUE                       (ENTER)
```

The program should execute statements 60, 70, 40, 50, and 55 before it again stops. N should have been divided by 3 again so that N now equals 1. C should equal 2. Verify this by commanding the computer to print the values of C, N, and K.

Type CONTINUE again. The computer should then print out the correct message.

Enter the number 9 again when the computer asks for an input.

At the next breakpoint, check the values of C, N, and K.

Notice that C = 2. It has not been reset to zero as it should be. This is why the program works only the first time through.

Now CONTINUE the program and check the values of C, N, and K at each breakpoint. Fill in the following blanks:

BREAKPOINT	C	N	K
35	_____	_____	_____
55	_____	_____	_____

3 TO THE POWER _____ = _____

Now let's fix the program. See if you can make the necessary changes to make the program RUN correctly. This is called debugging!

Also get rid of statements 35 and 55.

Now RUN the program again. Use the values given below for the input numbers. Fill in the blanks with the computer's response.

INPUT	3 TO THE POWER ?	= ?
9	_____	_____
9	_____	_____
3	_____	_____
27	_____	_____
5	_____	_____

Notice that the first 4 inputs get correct responses from the computer. But when the input equals 5, the computer gets hung up. Let's do some sleuthing and find out why.

Enter the FCTN 4 command to stop the program. Then set the TRACE command. Finally, RUN the program. Input 5 again. After the screen fills with numbers, type FCTN 4.

Notice that the same statement numbers are executed over and over in the same order:

40 50 60 70 40 50 60 70 ETC.

Evidently the condition that N=1 must never be met in statement 60.

Next, put a BREAK command at line number 65 so that we can keep track of N.

RUN the program again. Again, input the number 5.

At each breakpoint, find out the value of N.

FIRST BREAK N = _____

SECOND BREAK N = _____

THIRD BREAK N = _____

Can you see why N never equals 1 when N starts at 5?

To correct this program error, we may include the following program statements:

```
65 IF N < 1 THEN 120
120 PRINT K; "IS NOT AN INTEGER MULTIPLE OF 3."
130 GOTO 90
```

Type in the above statements, get rid of TRACE, and eliminate the breakpoint. Then RUN the program again.

Input the number 26 and then write down the computer's response on the line below.

_____

Is the computer's response correct? . . . . What a clever little computer!! (And an even smarter programmer!)

In the example below, you will learn about variations of the BREAK command. Enter the following program:

```
10 A=1
20 PRINT A
30 PRINT A;A+A
40 PRINT A;A+A;3*A
50 FOR I=1 TO 1000
60 NEXT I
70 PRINT :::
80 GOTO 10
```

Now type in the following command. (Notice no line number.)

```
BREAK 10,30,70
```

Now RUN the program. Each time a breakpoint is encountered, type in CONTINUE. Observe what happens.

The program should BREAK at statement 10 first, then at 30, and finally at statement 70. But notice that the breakpoints are automatically removed as the computer loops through the program additional times.

Stop the program using FCTN 4 and then RUN it again. This time do not enter the BREAK command.

Observe that the breakpoints at 10, 30, and 70 have been removed forever.

Stop the program again and add the following program line:

```
5 BREAK 10, 30, 70
```

Now RUN the program again. Again, type in CONTINUE whenever a program break occurs.

Observe that the breakpoints at 10, 30, and 70 again are set only once. This is because breakpoints are removed after having been executed.

Stop the program and start it again. Notice that the breakpoints have been reset because statement 5 (which sets the breakpoints) has been executed.

Stop the program again and change statement 80 to:

```
80 GOTO 5
```

Before RUNning the program again, what effect do you think this will have on the program?

Now RUN the program again.
Observe that the program stops at lines 10, 30, and 70 each time through the loop because statement 5 is encountered every time through the program.

Now type in CONTINUE several times as needed until the following message occurs on the screen:

```
*BREAKPOINT AT 10
```

After you get this message, type in:

```
UNBREAK
CONTINUE
```

The UNBREAK command clears out the rest of the breakpoints set in statement 5 so that no stops occur this time until statement 5 resets the breakpoints.

Now at the message

*BREAKPOINT AT 10

type in the following additional statement:

```
15 UNBREAK 10,30,70
```

Then type CONTINUE.

This time the program won't continue to execute because you have edited the program. So you must RUN the program again. Do it. Type CONTINUE at any breakpoint.

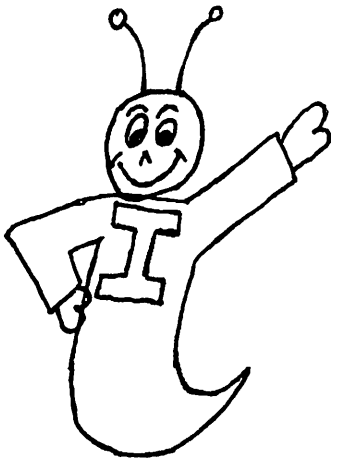
Notice that statement 15 has eliminated the breakpoints at 30 and 70, but it comes too late to affect the breakpoint set at line 10.

What happens if you input the following command? Write the computer's response on the line below.

```
BREAK 300
```

For your own benefit, you should practice using the TRACE, UNTRACE, BREAK, and UNBREAK commands until you have mastered them. They will be of great benefit when you start debugging large programs.

LESSON #18: ARRAYS



IN THIS LESSON, YOU WILL LEARN HOW TO USE ARRAYS IN YOUR PROGRAMMING. AN ARRAY IS A COLLECTION OR LIST OF VARIABLES. EACH VARIABLE IN THE ARRAY IS CALLED AN ELEMENT OF THE ARRAY. AN EXAMPLE OF A SIMPLE ARRAY IS SHOWN BELOW.

```
M(1) = 31
M(2) = 28
M(3) = 31
M(4) = 30
M(5) = 31
M(6) = 30
M(7) = 31
M(8) = 31
M(9) = 30
M(10)=31
M(11)=30
M(12)=31
```

M is the name of the above array. Notice that each element in the array is given a number. This number tells the position of the element in the array. Thus M(1) is the first element in array M. M(6) is the sixth element in array M, etc.

In the example given, the elements of the array have been assigned values representing the number of days in each month of the year. M(1) = 31 represents the number of days in the first month, January. M(2) = 28 is the number of days in February, and so on.

Arrays can help you simplify your programs. For example, if you had made the following assignments,

```
JAN = 31
FEB = 28
MAR = 31
APR = 30
  ETC.
```

then to print out all this information on the screen requires either twelve PRINT statements or one long PRINT statement. However, if we had this information stored in an array, we could put the information on the screen using the following statements:

```
80 FOR I=1 TO 12
90 PRINT M(I)
100 NEXT I
```

Let's see how that can be done. Type in the following program:

```
10 DIM N$(12), M(12)
20 CALL CLEAR
```

Here we will establish two arrays, N$(12) for the names of the months and M(12) for the number of days. Notice N$(12) is a string array while M(12) will contain only numbers.

Keep going. ——————→

```
30 INPUT "WHAT'S YOUR NAME: ":NAM$
40 PRINT
50 FOR I=1 TO 12
60 PRINT "WHAT IS THE NAME OF MONTH #";I
70 INPUT N$(I)
80 PRINT "HOW MANY DAYS IN ";N$(I):
90 INPUT M(I)
100 NEXT I
110 CALL CLEAR
120 PRINT "THANKS FOR THE DATA, ";NAM$::
130 PRINT "I HAVE PERFECT RECALL!  WATCH!"
140 FOR X=1 TO 1000
150 NEXT X
160 CALL CLEAR
170 FOR K=1 TO 12
180 FOR Y=1 TO 250
190 NEXT Y
200 PRINT "THE NUMBER OF DAYS IN ";N$(K);" IS ";M(K)
210 NEXT K
220 GOTO 220
```

The DIM command in line 10 tells the computer how many elements there are to be in an array. The DIM statement must always appear before the array is used in the program.

Try to figure out how the program works before you RUN it. Then RUN the program.

The DIM statement in line 10 actually provides for 13 elements in both array N$ and array M. The other element is the zeroth element. That is, N$(0) and M(0) are valid elements.

EXERCISE 18-1

Create a program that uses arrays to enter ten names and ten phone numbers. When you have this, add to the program the ability to enter a name and have the computer find and print the phone number.

So far, we have limited our dimensional arrays, or lists to one dimension. TI BASIC however allows for 2 or 3 dimensional arrays as well. Let's first talk about the 2-D (two dimensional) array. An example of a 2-D array is shown below.

	FIRST COLUMN	SECOND COLUMN	THIRD COLUMN	FOURTH COLUMN
FIRST ROW	1	2	3	4
SECOND ROW	2	4	6	8
THIRD ROW	3	6	9	12
FOURTH ROW	4	8	12	16

Let's call this array, TABLE. There are 16 elements in TABLE. To specify a particular element in TABLE, one must give two positions, the element's ROW and COLUMN.

For example, there is a 12 in ROW 4, COLUMN 3. So one could write:

```
TABLE(4,3)=12
```

Notice that the ROW position comes first:

```
TABLE(ROW,COLUMN)=ELEMENT VALUE
```

Similarly:

```
TABLE(2,4)=8
TABLE(1,3)=3
```

To see if you've gotten the general idea, fill in the following blanks:

TABLE(1,1)=_____ TABLE(2,3)=_____ TABLE(___,2)=8

Now let's use a 2-D array in a program. Type in the following program:

```
5 CALL SCREEN(15)
10 DIM A(24,32)
20 CALL COLOR(1,2,15)
30 CALL CHAR(33,"E7818100008181E7")
40 CALL CHAR(34,"E7F3F97C3E9FCFE7")
50 CALL CHAR(36,"E7CF9F3E7CF9F3E7")
60 RANDOMIZE
70 CALL CLEAR
80 INPUT "INPUT A # BETWEEN 0 AND 1 ":N
90 IF N>1 THEN 80
100 IF N<0 THEN 80
110 FOR ROW=1 TO 24
120 FOR COL=1 TO 32
130 X=RND
140 IF X>N THEN 170
150 A(ROW,COL)=33
160 GOTO 220
170 X=RND
180 IF X>.5 THEN 210
190 A(ROW,COL)=36
200 GOTO 220
210 A(ROW,COL)=36
220 NEXT COL
230 NEXT ROW
```

Keep going. ——————→

```
240 PRINT "THE ARRAY IS FILLED"
250 INPUT "TYPE C TO CONTINUE ":A$
260 IF A$<>"C" THEN 250
270 CALL CLEAR
280 FOR I=1 TO 24
290 FOR J=1 TO 32
300 CALL HCHAR(I,J,A(I,J))
310 NEXT J
320 NEXT I
330 GOTO 330
```

This is how the program works:

STATEMENT:

5	turns the screen gray.
10	defines an array called A which is to have 24 rows and 32 columns. Array A will be used to store screen characters.
20	Set 1 will have a black foreground and a gray background.
30,40,50	define 3 characters in set 1.
60	sets up the random number generator to give a different sequence of numbers every time the program is run.
80,90,100	These statements ask for a number between 0 and 1. If the number input is outside this range, statement 90 (or 100) causes the input statement to be repeated.
110-230	These statements cause the elements of array A to be given a value of 33, 34, or 36 in a random fashion.

STATEMENT:

110,120 The elements are filled in a row at a time:

ROW=1,COL=1
ROW=1,COL=2
ROW=1,COL=3

up to

ROW=1,COL=32

then

ROW=2,COL=1
ROW=2,COL=2

up to

ROW=2,COL=32

and so on.

130,140,150 160,220 Statement 130 generates a random number between 0 and 1. If the random number is less than or equal to the number you have input in statement 80, then statement 150 is executed which sets A(ROW,COL) equal to number 33 (also a character number). Then statement 160 causes a jump to statement 220 which increments the COL number by 1.

130,140, 170-220 If the random number generated in statement 130 is greater than the number you have input in 80, then statement 140 causes a jump to statement 70.

Statement 170 generates another random number between 0 and 1. If this random number is greater than ½, then the element of the array will be assigned the number 36. If the random number is less than or equal to ½, then the element of the array is given a value of 34.

STATEMENT:

240	This statement lets you know when all the elements of the array have been given a value.
250,260	These statements cause the computer to pause so that the message in statement 240 can be read before statement 270 clears the screen.
280-320	These statements cause all of the elements of the array to be printed on the screen in the same ROW and COL as the element appears in the array. Statement 300 causes the elements to be printed on the screen as the characters defined in statements 30-50.
330	Statement 330 holds the pattern on the screen.

RUN the program several times. Each time input a different number at statement 80. (Suggestions: Input zero, one, .5)

Try to figure out what would happen if statement 170 were changed to:

```
170 X=X
```

Then test your answer by RUNning the program. On the lines below, explain why X=RND is needed in line 170.

_____

_____

_____

_____

Next, put statement 170 back the way it was. Now change the program where necessary to create an array A containing 6 rows and 10 columns.

Let the program fill up this array in the same way as it filled the one before. Also, have the program print the elements of the array on the screen as it did before.

Show the changes to the program on the lines below.

RUN the program and de-bug it until it is working properly.

Now change the way the array is printed on the screen. Instead of printing row by row, change the program so that it prints the array column by column but still with the array elements in the same location on the screen.

Show the program changes below:

As a final test of understanding, change the printing technique once more. This time, have the program put the elements of the array on the screen using the RND command to determine which element gets printed first, which second, and so on. Each element should be printed in the same screen position as it was in the previous programming examples.

Test run your program to see that it is working properly. Then record the program changes below.

Another example of a 2-D array is given below. Type this program into the computer.

```
10 CALL CLEAR
20 DIM N(50,2)
30 INPUT A,B
40 IF A=0 THEN 90
50 I=I+1
```

Keep going. ——————→

```
60 N(I,1)=A
70 N(I,2)=B
80 GOTO 30
90 FOR J=1 TO I
100 CALL SOUND(N(J,1),N(J,2),0)
110 NEXT J
120 INPUT "R=REPLAY ":C$
130 IF C$="R" THEN 90
140 STOP
```

RUN the program. Input the numbers given below:

A	B	A	B
500	330	500	262
250	294	500	294
500	262	500	330
500	294	500	330
500	330	750	330
500	330	500	330
1000	330	500	294
500	294	500	294
500	294	500	330
1000	294	500	294
500	330	1000	262
500	392	0	0
1000	392		
500	330		
500	294		

Array N is a two column array. The first column represents the duration of a note. The second column represents the tone of the note. Thus, N(7,1) is the duration of the seventh note and N(7,2) is the tone of the seventh note. Play with this program a little. Make some changes.

If you are ready to go on, change the program so that two 1-D arrays are used instead of the one 2-D array. Let D(I) be the array representing the duration of the Ith note. Let T(I) be the array representing the tone of the Ith note. When you get your program working properly, record the program on the lines below.

Now let's take a look at three dimensional arrays. Elements in a 3-D array are specified by giving three numbers (or positions). An example of a 3-D array is shown below.

C O L U M N ARRAY M

	1	2	3	4	
ROW 1	1	19	8	3	PAGE 1
ROW 2	14	21	17	11	

ROW 1	6	7	16	13	PAGE 2
ROW 2	12	10	23	22	

ROW 1	15	2	100	18	PAGE 3
ROW 2	5	9	20	4	

The number 16 has been assigned to one of the positions in the array M. Note that the 16 is on page 2 in the first row and in the third column. Therefore, one could write: M(2,1,3)=16. In a similar way, one could write: M(1,1,3)=8. Notice that the numbers represent page, then row, and then column.

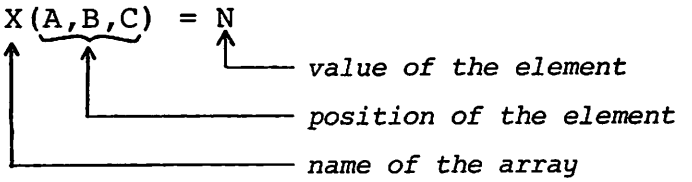
Now you fill in the following blanks:

M(_____,_____,_____)=5

M(2,2,2) = _____ M(1,1,1)+M(3,1,2)=1 + 2 = 3

M(1,1,1) = _____ M(2,2,4)+M(2,2,1)= _____

The general form for all arrays is as follows:



Besides numerical arrays, one may also have string arrays. For example, M$(1,1)="HELLO". In this example, the array M$ is a 2-D string array. The value at location, ROW=1 and COL=1 is the string "HELLO".

Type in the following program which contains a 3-D array.

```
10 CALL CLEAR
20 DIM A$(3,3,3)
30 PRINT "3-D TIC TAC TOE":::
40 FOR P=1 TO 3
50 FOR R=1 TO 3
60 FOR C=1 TO 3
70 INPUT T$(P,R,C)
80 NEXT C
90 NEXT R
100 NEXT P
110 PRINT :::::
120 FOR P=1 TO 3
130 FOR R=1 TO 3
140 FOR C=1 TO 3
150 PRINT T$(P,R,C);"  ";
160 NEXT C
```

Keep going. ——————→

```
170 PRINT
180 NEXT R
190 PRINT :::
200 NEXT P
210 GOTO 210
```

When you RUN the program, input the following data (one at a time):

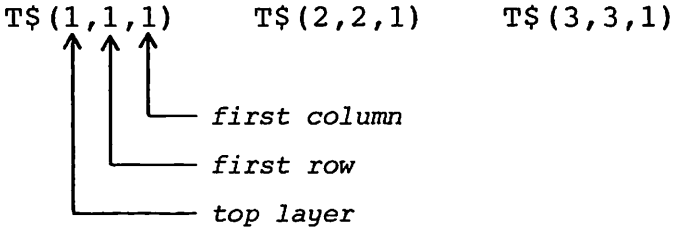
X, O, __, O, X, __, __, O, O, __, __, __, X, X,

O, __, __, __, __, X, __, __, O, __, X, O, X

The program displays your data in the form of a 3-D TIC-TAC-TOE board. The 3 by 3 square at the top of your screen represents the top layer of the 3-D TIC-TAC-TOE board. The square in the middle of your screen represents the middle layer of the 3-D TIC-TAC-TOE board, and so on.

Can you tell that "X" won the TIC-TAC-TOE game in 2 different ways?

One way was:



What was the other way?

_____

EXERCISE 18-2

Write a program that INPUTs numbers from the keyboard and stores them in a 1-D array. Design the program so that when you enter the number 999, the computer stops taking new numbers and begins printing the elements of the array back onto the screen. Design the program so that up to 1000 numbers can be put into the array.

When you get the above program to work properly, add some program statements that will allow the computer to keep track of the number of elements that have been input into the array. The computer should then print this information on the screen:

YOU HAVE INPUT __?__ NUMBERS

EXERCISE 18-3

In this lesson, you are going to design a "coloring" program which will allow you to paint pictures on your TV screen. These pictures will be made of colored squares.

Include a statement that will allow the user to choose the "background" screen color upon which the pictures will be painted. For example:

```
INPUT "SCREEN COLOR ":SC$
```

An INPUT statement should also be used to input the color of the square and the square's position to the computer. This information will be stored in a 2-D array.

The INPUT statement may have the following form:

```
INPUT "ROW,COLUMN,COLOR ":R,C,A$
```

This information may be stored in an array using a statement similar to the one below:

```
TV$(R,C)=A$
```

R and C are the row and column numbers of a screen location. And A$ is the color of the square to be placed at this position.

A$ could be any of the following string constants:

B, MG, LG, DB, LB, DR, C, MR, LR, DY, LY, DG, M, G, W

These represent the possible colors available in TI BASIC: B=black, MG=medium green, LG=light green, and so on.

For example, when the computer prints out:

```
ROW,COLUMN,COLOR
```

the user might respond:

```
8,22,LY
```

The computer should then put the string constant, LY, at row 8, column 22 of the array TV$. All of this would mean that the user wants a light yellow square to be printed on the TV screen at row 8, column 22.

The program should allow you to input as many position colors as you want. When you are through entering colors, enter:

```
0,0,0
```

to tell the computer you are through.

The DECODING of color codes can be done by using IF-THEN statements. An example is shown below.

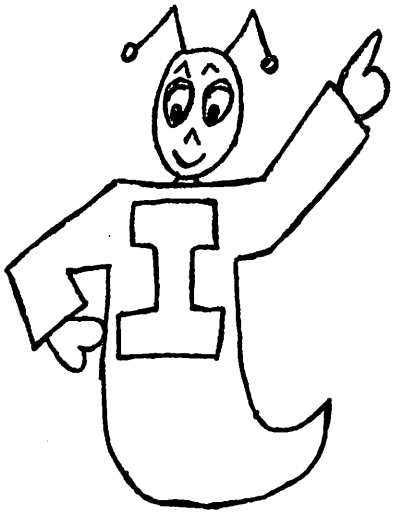
```
1000 IF TV$(I,J)="B" THEN 2000
1010 IF TV$(I,J)="MG" THEN 2020
                .
                .
                .
              Etc.
2000 CALL HCHAR(I,J,40)
2010 GOTO 3000
2020 CALL HCHAR(I,J,48)
                .
                .
                .
              Etc.
3000 NEXT J
3101 NEXT I
```

Statement 2000 contains the character code 40 which is to represent a black square. The character code 48 in statement 2020 represents a medium green square.

After the program fills up the screen, it should wait a while and then loop back to the beginning so that the user may make changes or additions.

When you get the program to work properly, call someone over and share your success!

LESSON #19: MORE IF-THEN



IN THIS LESSON, YOU WILL LEARN HOW TO ADD PIZZAZ TO THOSE CONDITIONAL STATEMENTS. YOU WILL LEARN HOW TO INCORPORATE MORE THAN ONE CONDITION IN AN IF-THEN STATEMENT AND HOW TO CAUSE ADDITIONAL BRANCHING BY USING IF-THEN-ELSE.

Begin this lesson by typing in the following program:

```
10 CALL CLEAR
20 INPUT "INPUT A NUMBER ":N
30 IF N=16 THEN 40 ELSE 10
40 PRINT "YOU GUESSED THE NUMBER"
50 FOR I=1 TO 500
60 NEXT I
70 GOTO 10
```

RUN the program and input some numbers. Notice how statement 30 works. When the number input is 16, statement 30 causes the program to jump to statement 40. If the condition that N=16 is not met, the ELSE part of statement 30 causes the computer to jump to statement 10.

In English, one could interpret statement 30 as follows:

> If the number N is a 16, then execute statement 40, otherwise execute statement 10.

Sometimes one would like to include more than one condition in the same statement. This is possible in TI BASIC. The following program illustrates this possibility. Type in the program and RUN it.

```
10 CALL CLEAR
20 INPUT "INPUT A NUMBER: ":N$
30 IF (N$="10")+(N$="TEN") THEN 50 ELSE 90
40 STOP
50 PRINT "CONDITION MET"
60 FOR I=1 TO 500
70 NEXT I
80 GOTO 10
90 PRINT "CONDITION NOT MET"
100 FOR I=1 TO 500
110 NEXT I
120 GOTO 10
```

Notice the special use of this symbol to mean OR.

Try entering the following string constants at statement 20:

1, ONE, 10, TEN, 10.0

The + in statement 30 should be read as OR. Thus statement 30 reads:

> If N$ equals "10" OR if N$ equals "TEN", then go to statement 50; otherwise, go to statement 90.

When you are satisfied that you know how the program works, you may continue.

Now make the following changes to the previous program:

```
20 INPUT "INPUT 2 NUMBERS: ":A$,B$
30 IF (A$="10")+(A$="TEN")+(B$="7") THEN 50 ELSE 90
```

Try the following inputs. Check (✓) the appropriate response:

INPUT	NEITHER CONDITIONS MET	AT LEAST ONE CONDITION MET
1,6	_____	_____
1,7	_____	_____
2,7	_____	_____
10,5	_____	_____
TEN,5	_____	_____
10,7	_____	_____
TEN,7	_____	_____

Next, make the following changes to the above program:

```
30 IF(A$="5")*(B$="6") THEN 50 ELSE 90
50 PRINT "BOTH CONDITIONS MET"
90 PRINT "AT LEAST ONE CONDITION NOT MET"
```

*Notice the special use of * to mean AND.*

RUN the program again.

Now turn the page and fill in the table.

INPUT	BOTH CONDITIONS MET	AT LEAST ONE CONDITION NOT MET
7,7	_____	_____
7,8	_____	_____
5,7	_____	_____
7,6	_____	_____
5,6	_____	_____
6,5	_____	_____

The * in statement 30 should be read as AND. Thus statement 30 reads:

> If A$ equals "5" AND if B$ equals "6", then go to statement 50, otherwise, go to statement 90.

Only when all the conditions are true, does the AND IF-THEN statement cause a jump to statement 50. For the OR IF-THEN statement, only one condition had to be true for a jump to statement 50 to take place.

Now type in the following program:

```
10 CALL CLEAR
20 CALL COLOR(2,2,15)
30 CALL CHAR(40,"FFFFFFFFFFFFFFFF")
40 CALL SCREEN(15)
50 CALL VCHAR(1,1,40,48)
60 CALL VCHAR(1,31,40,48)
70 D=1
80 I=3
```

Keep going. ——————→

```
90 CALL HCHAR(10,I,61)
100 CALL GCHAR(10,I-D,X)
110 IF X=40 THEN 130
120 CALL HCHAR(10,I-D,32)
130 I=I+D
140 IF (I=2)+(I=31) THEN 150 ELSE 90
150 D= -1*D
160 I=I+D
170 GOTO 90
```

RUN the program and observe the moving pattern which results.

Statements 10 through 60 define the background colors and the border color which appear on the screen. Statements 70 through 170 produce and control the moving equal sign.

Let's first examine statement 100 which contains the CALL GHAR command. GCHAR stands for Get CHARacter. This command is used to determine what character is currently printed at some screen location.

The form of the command is as follows:

CALL GCHAR(ROW,COLUMN,CODE)

When this command is used, the variable, CODE, is set equal to the character code of the symbol printed at position ROW,COLUMN on the screen.

For example, if the letter A is printed at location ROW=5, COLUMN=22 on the screen, then CALL GCHAR(5,22,CODE) will set the numeric variable CODE equal to 65, since 65 is the character code for A.

Now back to the program . . .

Statement 90 prints an equal sign at position ROW = 10, COLUMN = I on the screen. Statement 100 then finds out what character is printed one column behind this equal sign. If the character behind is a border character, then statement 120 is by-passed. If the character behind the equal sign is another equal sign, then this previously printed equal sign is erased by statement 120.

Statement 130 changes the column variable, I, by one. Statement 140 checks to see that I doesn't equal a border column number. If it doesn't, the program loops back to statement 90 and prints the new equal sign. If I does equal a border number, then the program jumps to statement 150, where the direction variable, D changes sign. The column variable is then changed in the opposite way. If it was increasing before, it now decreases, or if it was decreasing, it now increases.

EXERCISE 19-1

Given the following IF-THEN statement,

```
100 IF (A<>7)*(B=5)+(C=2) THEN 1000 ELSE 5000
```

tell which statement the program would transfer to for the following values of A, B, and C:

A	B	C	TO STATEMENT:
7	5	3	_____
6	5	3	_____
7	5	2	_____
10	4	7	_____

Now check your answers by writing a simple program which uses statement 100 above.

EXERCISE 19-2

In this exercise, you will be learning about a convenient way of writing large numbers. For example, you'll agree that it takes an insufferable amount of time to write down the following number:

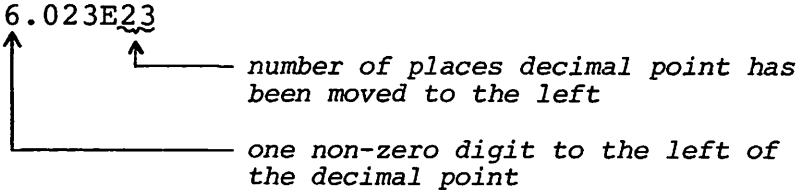
602300000000000000000000

But this is about equal to the number of atoms in a cubic centimeter of a metal.

What we need is a short-cut for writing down this number. We will call this short-cut method, SCIENTIFIC NOTATION. How's that for an important sounding name!

To convert the above number to scientific notation, all we have to do is move the decimal point so that there is only one non-zero digit to the left of the decimal point, and then write down the number of places the decimal point was moved.

The above number becomes 6.023E23 when written in scientific notation.



If one wanted to convert this number back to its old form, this would be quite easy. Just move the decimal point 23 places to the right and fill in the missing zeroes which serve as place holders: 6.023 *fill with zeroes*

EXERCISE 19-2 (CONT.)

Let's look at a few more examples:

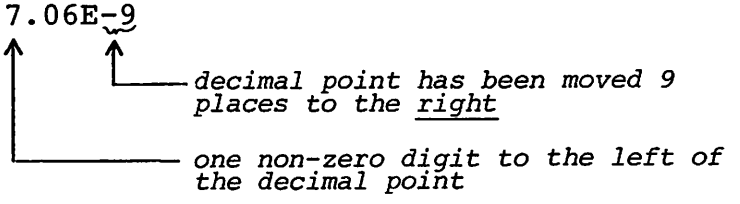
86000 = 8.6E4

2004000000 = 2.004E9

2004 = 2.004E3

-2004 = -2.004E3

Scientific notation can also be used to write down numbers which are incredibly small. For example, 0.00000000706 becomes:



Other examples:

.1 = 1E-1

.01 = 1E-2

-.051 = -5.1E-2

1 = 1E0

500E-2 = 5E0

.05E-2 = 5E-4

700E3 = 7E5

Study the above examples carefully and then fill in the following blanks.

A	B(IN SCIENTIFIC NOTATION)
2000	_____
20100	_____
300.54	_____

EXERCISE 19-2 (CONT.)

A	B(IN SCIENTIFIC NOTATION)
.00720	_____
.00720000	_____
_____	7.0605E3
_____	-7.00E-2
_____	6.54E7
_____	8.09E-0
9.076	_____
90.3E2	_____
90.3E-2	_____

To check your answers, type in and RUN the following program. Use columns A and B above as the inputs to the program.

```
10 CALL CLEAR
20 PRINT TAB(5);"CHECK YOUR ANSWERS"
30 PRINT :::
40 PRINT "INPUT A AND B: ":
50 INPUT A,B
60 IF A=B THEN 70 ELSE 100
70 PRINT :A;TAB(11);"EQUALS";TAB(18);B
80 PRINT :::
90 GOTO 40
100 PRINT :A;TAB(11);"DOES NOT =";TAB(18);B
110 PRINT :::
120 GOTO 40
```

Notice the TAB function which has been introduced in lines 70 and 100. The TAB function tells the PRINT command where to print the next print item. For example, TAB(6) tells the computer to print the next item in column 6 from the left. The following program illustrates this property. Type it into the computer and RUN it.

```
10 CALL CLEAR
20 N= -5
30 PRINT "COLUMN";N
40 PRINT TAB(N);"*"
50 N=N+1
60 FOR I=1 TO 1000
70 NEXT I
80 GOTO 30
```

Notice that when N is less than or equal to zero, the TAB(N) function causes the next print to occur in column 1. If N is between 1 and 28, including 1 and 28, then the TAB(N) function causes the next print item to occur in that column number.

However, if the value of N is greater than 28, then the computer keeps reducing the value of N by 28 until N finally equals a number between 1 and 28. For example, TAB(30) would cause the next print to occur in column: (30-28) = 2. TAB(60) would allow the next print item to be printed in column: 60 - 28 - 28 = 4.

One last note. The TAB(N) function will not allow you to print backwards on a line. If you attempt to do so, the TAB(N) function will cause the next item to be printed on the next line in column N. The next program illustrates this property. Type it in and RUN it.

```
10 CALL CLEAR
20 PRINT "123456789";TAB(7);"*"
30 GOTO 30
```

The TAB(7) does not allow the "*" to be printed over the "7". So the "*" appears in column 7 of the next line.

The program below shows an easy way to generate a random star field. Type it in and RUN it.

```
5 CALL CLEAR
10 RANDOMIZE
20 CALL COLOR(2,16,1)
30 CALL SCREEN(2)
40 PRINT TAB(28*RND);"*"
50 GOTO 40
```

NOTE: THE TAB FUNCTION AUTOMATICALLY ROUNDS OFF ITS ARGUMENT, 28*RND, SO THAT YOU DON'T HAVE TO WORRY ABOUT THE FACT THAT 28*RND WILL GENERALLY NOT BE A WHOLE NUMBER.

The CALL COLOR command causes set 2 characters to have a white foreground on a transparent background. The transparent background means that the square behind the character will take on the color of the screen. To illustrate this, RUN the program twice more with the following changes. Once with:

```
30 CALL SCREEN(7)
```

and once with:

```
20 CALL COLOR(2,15,2)
30 CALL SCREEN(7)
```

Another TAB program is shown below. Type it in and RUN it.

```
10 CALL CLEAR
20 X=14*SIN(I)
30 Y=ABS(X)
40 PRINT TAB(Y);"*"
50 I=I+.25
60 GOTO 20
```

Statement 20 generates values of x from -14 to +14. (Don't worry about the SIN function. All you need to know is that it generates values between -1 and 1 depending on the value of I.) Statement 30 contains the ABSolute value function. If X is a positive number, statement 30 sets Y equal to X. If X is a negative number, then statement 30 sets Y equal to -X. That is, the ABS value function always returns a positive number. Examples are shown on the next page.

```
ABS(28.4)     = 28.4

ABS(-28.4)    = 28.4

ABS(2.8E-4)  = 2.8E-4 or .00028

ABS(-2.8E-4)= 2.8E-4 or .00028
```

LESSON #20: SPEECH

HEAR YE! HEAR YE! IN THIS LESSON, YOU ARE GOING TO LEARN HOW TO MAKE THE COMPUTER SPEAK. IT'S REALLY QUITE EASY TO DO!

To begin, make sure that the Speech Synthesizer is attached to the right side of the computer console. If it isn't, ask your teacher to plug it in for you.

Next turn your TV monitor and your computer console on. (Or if these are already on, press FCTN = to return to the TI title page.)

Find the Speech Editor Command Module and slide it into the slot on the top of your computer. The title page should disappear momentarily and then reappear.

Next press any key to obtain the master selection list.

Finally, select the SPEECH EDITOR option. You are now ready to program the computer to speak. Type in HELLO and then press ENTER.

The speech synthesizer should say the word, HELLO. Press ENTER again. Again, the word HELLO should be spoken by the computer.

Next type in the word, I. The screen should now have the following words:

HELLO I

Be sure to leave a space between and O and I. If you make a typing mistake, you can use all the editing features that you have learned about previously to correct the mistake.

Press ENTER to hear the words: HELLO I

Now finish the sentence in the following way:

HELLO I AM A #TEXAS INSTRUMENTS# HOME COMPUTER

Press ENTER to listen to the sentence.

The # symbols around TEXAS INSTRUMENTS must be included because the speech synthesizer doesn't know the word TEXAS or the word INSTRUMENTS, but it does understand #TEXAS INSTRUMENTS#. The # symbols let the computer know that these two words are stored in memory as one word.

Now press FCTN 4 to erase the text on the screen.

Now try this. Type in the following and press ENTER.

HELLOI

Instead of hearing the words HELLO and I, the letters, HELLOI, flash at the bottom of the screen, indicating that this "word" is not in the vocabulary.

One may change the pause between words by using word separators other than spaces. A list of word separators is given on the next page.

SEPARATOR	WAIT(IN SECONDS)
+	0
space	.1
-	.2
,	.3
;	.5
:	.8
.	1.0

Try the following examples. Use the edit features to insert the separators.

```
I AM SO SORRY          (ENTER)
I.AM;SO+SORRY          (ENTER)
I+AM++SO...SORRY       (ENTER)
```

Now use an arrow key to position the flashing cursor over either letter of the word SO. Then press FCTN 3. Notice that this deletes the word SO and all trailing separators.

Let's now incorporate the speech feature into a TI BASIC program. Press FCTN = . Then press 1 twice to get to the TI BASIC option. Then type in the following program and RUN it.

```
10 CALL CLEAR
20 PRINT "TYPE IN A NUMBER BETWEEN"
30 PRINT TAB(10);"0 AND 9"
40 CALL SAY("TYPE IN A NUMBER BETWEEN 0 AND 9")
50 INPUT ":":A$
```

Keep going. ——————→

```
60 CALL SAY(A$)
70 B$="DO+IT+AGAIN"
80 CALL SAY(B$)
90 GOTO 50
```

Notice the form of the command used to speak words. The words to be spoken are enclosed in quotes to form a string. This string is then used as the argument of the CALL SAY command.

In statement 40, the word string itself is the argument of the CALL SAY command. But in statements 60 and 80, a string variable, which has been set equal to the word string, is used as the argument of the CALL SAY command. This latter form can save a considerable amount of typing if a phrase needs to be spoken over and over.

Now type in the following example and RUN it.

```
10 DIM A$(9)
20 A$(0)="0"
30 A$(1)="1"
40 A$(2)="2"
50 A$(3)="3"
60 A$(4)="4"
70 A$(5)="5"
80 A$(6)="6"
90 A$(7)="7"
100 A$(8)="8"
```

Keep going. ——————→

```
110 A$(9)="9"
120 FOR I=0 TO 9
130 CALL SAY(A$(I))
140 NEXT I
```

When you are satisfied that you know how the program works, continue.

Now make the following change to the program and RUN it again.

```
110 A$(9)="XX"
```

Notice the sound which occurs when statement 130 attempts to say "XX". Because no such word exists in the synthesizer vocabulary, you hear an "UHOH" sound.

Now type in the following program and RUN it.

```
10 CALL CLEAR
20 PRINT "SOMEWHERE"
30 CALL SAY("SOMEWHERE")
40 PRINT "SOME+WHERE"
50 CALL SAY("SOME+WHERE")
60 A$="THEREFORE"
70 B$="THERE+FOUR"
80 PRINT A$
90 CALL SAY(A$)
100 PRINT B$
110 CALL SAY(B$)
120 PRINT "A$=THERE,B$=4"
```

Keep going. ——————→

```
130 A$="THERE"
140 B$="4"
150 CALL SAY(A$&B$)
160 A$="READ"
170 B$="READ1"
180 PRINT A$
190 CALL SAY(A$)
200 PRINT B$
210 CALL SAY(B$)
220 GOTO 220
```

This program illustrates how some words can be spoken even though they are not in the vocabulary list of the Speech Synthesizer. Notice the three different ways shown.

The program also illustrates that some words in the TI vocabulary have two different pronunciations. For example, READ is pronounced as reed, but READ1 is pronounced as red.

EXERCISE 20-1

Now it's your turn. Write a program which asks you to input two numbers. The program should then compute the sum of the two numbers that you input and then say:

"HERE IS THE SUM"

The program should then print out the sum. Write your completed program below.

NOTE: A COMPLETE LIST OF VOCABULARY WORDS FOR YOUR TI SPEECH SYNTHESIZER IS GIVEN ON THE FOLLOWING PAGES.

VOCABULARY LIST

-(NEGATIVE)	BLUE	DO	FOURTEEN
+(POSITIVE)	BOTH	DOES	FOURTH
.(POINT)	BOTTOM	DOING	FROM
0	BUT	DONE	FRONT
1	BUY	DOUBLE	G
2	BY	DOWN	GAMES
3	BYE	DRAW	GET
4	C	DRAWING	GETTING
5	CAN	E	GIVE
6	CASSETTE	EACH	GIVES
7	CENTER	EIGHT	GO
8	CHECK	EIGHTY	GOES
9	CHOICE	ELEVEN	GOING
A	CLEAR	ELSE	GOOD
A1	COLOR	END	GOOD WORK
ABOUT	COME	ENDS	GOODBYE
AFTER	COMES	ENTER	GOT
AGAIN	COMMA	ERROR	GRAY
ALL	COMMAND	EXACTLY	GREEN
AM	COMPLETE	EYE	GUESS
AN	COMPLETED	F	H
AND	COMPUTER	FIFTEEN	HAD
ANSWER	CONNECTED	FIFTY	HAND
ANY	CONSOLE	FIGURE	HANDHELD UNIT
ARE	CORRECT	FIND	HAS
AS	COURSE	FINE	HAVE
ASSUME	CYAN	FINISH	HEAD
AT	D	FINISHED	HEAR
B	DATA	FIRST	HELLO
BACK	DECIDE	FIT	HELP
BASE	DEVICE	FIVE	HERE
BE	DID	FOR	HIGHER
BETWEEN	DIFFERENT	FORTY	HIT
BLACK	DISKETTE	FOUR	HOME

HOW
HUNDRED
HURRY
I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT
J
JOYSTICK
JUST
K
KEY
KEYBOARD
KNOW
L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER
M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST
N
NAME
NEAR
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER
O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER
P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
POSITIVE
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING
Q
R
RANDOMLY
READ
READ1
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND
S
SAID
SAVE
SAY
SAYS
SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY
SPACE
SPACES
SPELL
SQUARE
START
STEP

STOP
SUM
SUPPOSED
SUPPOSED TO
SURE
T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS CORRECT
THAT IS RIGHT
THE
THE1
THEIR
THEN
THERE
THESE
THEY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRTY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE
U
UHOH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE
V
VARY
VERY
W
WAIT
WANT
WANTS
WAY
WE
WEIGH
WEIGHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WON
WORD
WORDS
WORK
WORKING
WRITE
X
Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR
Z
ZERO

THE COLORED PAGES

At the end of this manual, you will find several colored pages. These are projects that test your ability to use what you have learned. There are no right or wrong answers. If your program does what is asked, then it is quite acceptable. You are free to express your creativity. Be proud of what you do. Do not worry whether your solution is like anyone else's.

Some of these projects may seem easy. . .but do not be deceived into thinking that you can skip them. After all, if they are easy for you, then it will not take long to do them.

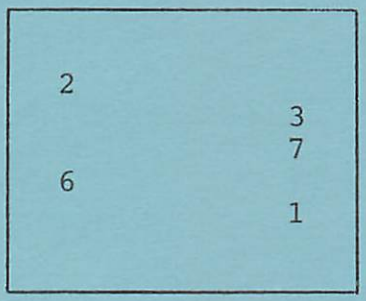
Good luck!

Henry A. Taitt

Henry A. Taitt
Director

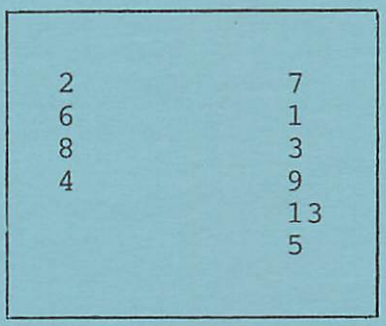
BLUE PROJECT 1

In YELLOW PROJECT 1, you were asked to input five numbers and then have the computer print the even numbers in one column and the odd numbers in a second column. Your screen may have looked like this:



Each number was printed as it was entered. You had nowhere to store them and arrange them. An array will let you store them and then arrange them for printing.

CREATE a program that will allow you to place ten different numbers into an array, and then print the even numbers in one column and the odd ones in a second column so that your screen looks like this:



BLUE PROJECT 2

CREATE a program that will randomly pick ten numbers, but will not pick the same number twice. Store them in an array and then display them on the screen. (See YELLOW PROJECT 8 . . . it should be much easier to do with arrays!!!)

BLUE PROJECT 3

Using the PRINT TAB(X) command, CREATE a program that will produce the following display:

N	N^2	N^3	N^4	N^5
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64		
5	25	125	etc.	

BLUE PROJECT 4

CREATE a special character of your own design.

Have it appear randomly on the screen ten times.

BLUE PROJECT 5

CREATE a program that will allow you to input two numbers. Have your program add the numbers and get the sum. Then have your program (using the speech synthesizer) pronounce the numbers and their sum. For example:

(on the screen) 3 + 4 = 7

(sound) Three plus four equals seven.

PROJECT BLUE 6

Using the speech synthesizer, CREATE a program that will recite a four line poem. Have a colored picture appear with each line.

BLUE PROJECT 7

The following is a code.

COLUMN I	COLUMN II	COLUMN I	COLUMN II
A	E	N	Y
B	F	O	X
C	G	P	W
D	A	Q	V
E	H	R	U
F	I	S	T
G	B	T	S
H	J	U	R
I	K	V	Q
J	L	W	P
K	M	X	O
L	C	Y	N
M	Z	Z	D

CREATE a program that will allow you to code a message (You put in letters in column I and get out letters in column II.) or decode a message. (You put in letters in column II and get out letters in column I.)

BLUE PROJECT 8

A deck of playing cards has four different suits: hearts, spades, diamonds, and clubs. Each suit has 13 cards: Ace, two, three, four, five, six, seven, eight, nine, ten, Jack, Queen, and King.

Randomly draw two hands of five cards each and store them in an array. Then print them on the screen in two columns. Remember that no card may be used more than once!

Send us a list of your working program, and we will send you your BLUE PROGRAMMER V card. This BLUE page must be included.

Send to:

Henry A. Taitt
CREATIVE Programming Inc.
604 Sixth Street
Charleston, IL 61920

TI-99/4A

Your name _____

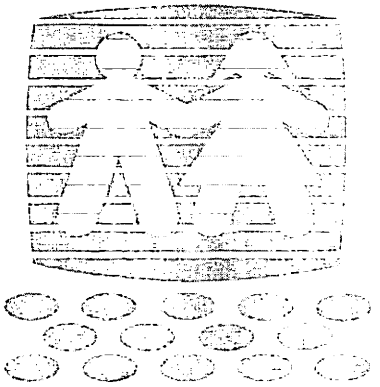
Phone # _____

Address _____

City, State _____

Zip_____Birthdate _____

Don't forget to enclose a self-addressed stamped envelope.



CREATIVE Creations

A FORUM FOR YOUNG MINDS

CREATIVE Programming, Inc., Charleston, IL 61920

A newsletter published 12 times a year. The articles are for young programmers, about young programmers and often written by young programmers.

Each month a graphics program created by a student is selected for the cover. It could be yours! Contests, mind bending challenges, computer game reviews, new creations, programs, even an X-rated column for parents and teachers who are running programs in their areas.

Name_____

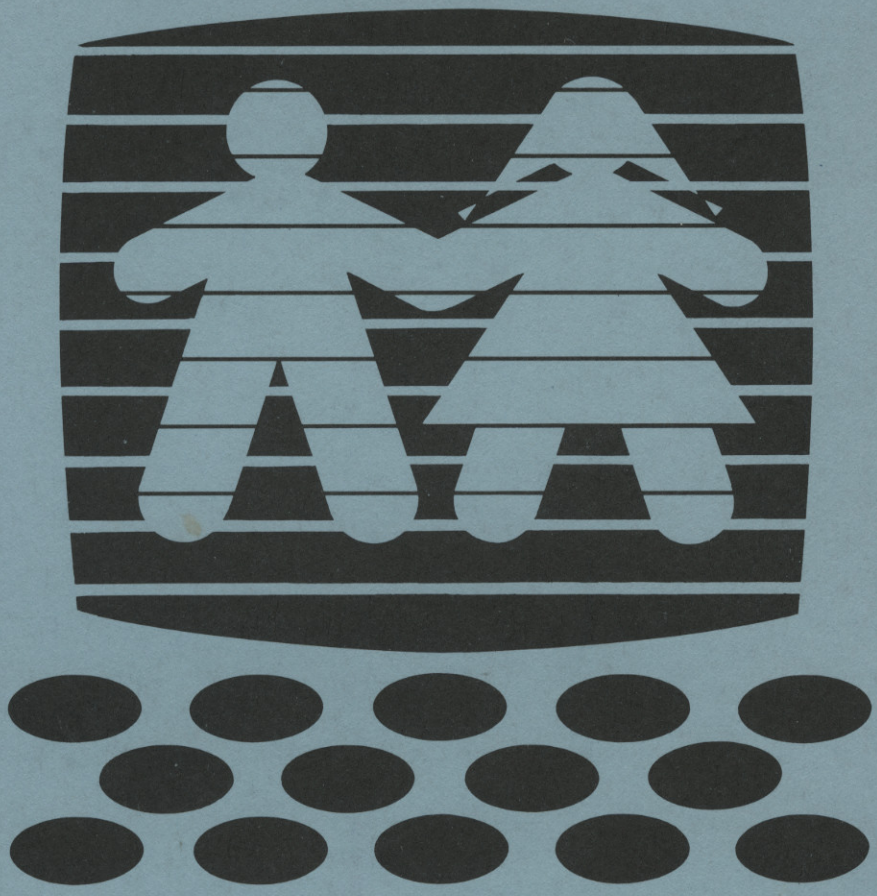
Address _____

City _____ State _____ Zip _____

Please make checks payable to: CREATIVE Creations
604 Sixth Street
Charleston, IL 61920

Only $18 a year ($32 for two years) brings all twelve issues to your door. Join us today in sharing in the excitement of CREATIVE Programming through CREATIVE Creations.

☐ one year ($18.00) ☐ two years ($32.00)



CREATIVE PROGRAMMING INCORPORATED
A SUBSIDIARY OF R.V. WEATHERFORD CO.

604 6th St., Charleston, IL 61920

T19-R-05