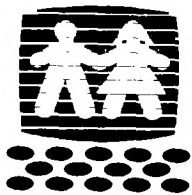


Woodward's
55052
0647 7
1 9.95

Creative Programming for Young Minds

... on the TI-99/4A



Volume III

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 Koloanis, 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 III

by Leonard Storm



© 1982, CREATIVE Programming, Inc., Charleston, IL 61920
A Subsidiary of R.V. Weatherford Co.

Congratulations! Volume III welcomes you, TI Level
II Programmer!

You are now well on your way to the wonderful world of
computer programming. So let's continue. More good
commands are on the way!

In Volume II, the following TI BASIC commands and
symbols were discussed:

FOR-NEXT STEP	=	,
PRINT variable		
CALL HCHAR	+	;
CALL VCHAR	*	:
CALL COLOR		

The materials in this new volume build upon the concepts
of Volume II. So make sure that you have mastered that
material before continuing on with this manual.

CREATIVE PROGRAMMING FOR YOUNG MINDS

... ON THE TI/99-4A

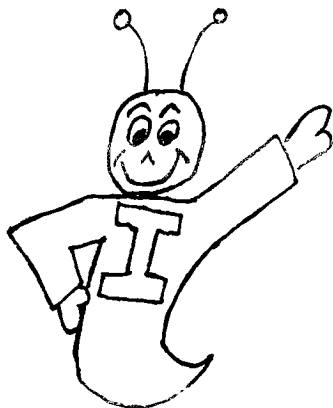
VOLUME III

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

LESSON #9	RANDOM NUMBERS	99
	RND	99
	RANDOMIZE	100
	INT(X)	102
LESSON #10	IF--THEN	108
	INPUT	110
	< (less than)	113
	> (greater than)	113
	<= (less than or equal to)	117
	>= (greater than or equal to)	117
	<> (not equal to)	117
LESSON #11	SOME MATH	122
	+ (addition)	122
	- (subtraction)	122
	* (multiplication)	122
	/ (division)	122
	Immediate Mode	122
	^ (raising a number to a power)	125
LESSON #12	CHAR	130
	Dot Code	131
	Character String	133
	CALL CHAR	133
	REM	137
VOLUME III	REVIEW QUIZ	141

YELLOW PROJECTS

LESSON #9: RANDOM NUMBERS



WELCOME BACK, PARTNERS! REMEMBER ME?
 I THOUGHT SO! SAY, DO YOU KNOW WHAT
 RANDOM NUMBERS ARE?
 RANDOM NUMBERS ARE NUMBERS WHICH
 FOLLOW NO PARTICULAR PATTERN.

The sequence of numbers: 1, 4, 7, 10, 13, 16, . . . and so on, is NOT a random sequence because the numbers follow predictably. Each number is 3 more than the one before it.

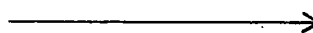
The sequence of numbers: 198, 24, 76, 76, 19, 44, 87, 2, 93, . . . and so on, IS a RANDOM sequence of numbers because there is no rule which would tell us which number should come next.

A lot of computer games use random numbers. This allows the computer to play differently each time the game is played.

Let's tell the computer to produce some random numbers.

Type in the following program:

```
100 CALL CLEAR
110 PRINT "RND= ";RND
120 PRINT "RND= ";RND
130 PRINT "RND= ";RND
```

Keep going. 

```
140 PRINT "RND= ";RND
150 GOTO 150
```

Now, RUN the program. Notice that the computer usually gives RND a different value each time it is used in the program. RND always has a value between 0 and 1 (0 included).

Record the values of RND on the lines below.

Now RUN the program several times. Notice that every time you RUN the program, the same sequence of numbers is generated.

RANDOMIZE

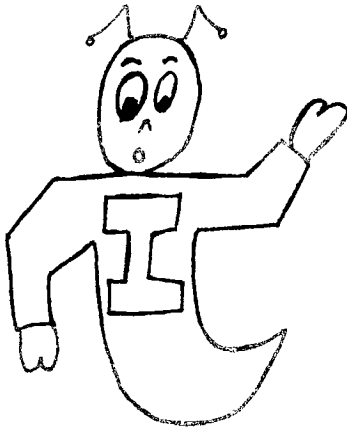
To get a different sequence of numbers, you need to use a special command in addition to RND. This special command is RANDOMIZE.

Put the following statement in your last program and RUN it again:

```
105 RANDOMIZE
```

Record the first four numbers below and then RUN the program again.

So statement 105 causes a different set of numbers to be produced than were produced before. Each time the RANDOMIZE statement is executed, it produces a different sequence of numbers.



EVERY TIME RANDOMIZE IS USED, A DIFFERENT SET OF RND NUMBERS WILL OCCUR.

EVERY TIME RND IS USED WITHOUT RANDOMIZE THE SAME SET OF RND NUMBERS WILL OCCUR.

Suppose you want the computer to produce RANDOM numbers between 0 and 10. No problem! Just multiply RND by 10. RUN the program below to try it out.

```

100 CALL CLEAR
110 RANDOMIZE
120 FOR I=1 TO 14
130 PRINT 10*RND
140 NEXT I
150 GOTO 150
  
```


Every time you RUN the program from the last page you will get a different set of numbers because of the RANDOMIZE statement.

Now if it's whole numbers (or integers) that you want, change statement 130 to:

```
130 PRINT INT(10*RND)
```

RUN the program a few times. Does a 10 ever occur?

INT(X)

The INT command changes a decimal number to a whole or integer number. For example:

```
INT(1.04) would equal 1
```

```
INT(1.94) would equal 1
```

```
INT(7.18) would equal 7
```

The INT command may also be used with negative numbers:

```
INT(-3.26) would equal -4
```

```
INT(-6.08) would equal -7
```

```
INT(-5) would equal -5
```

Notice that INT(X) always takes the next lower whole number. (-4 is lower than -3.26.)

To produce the integer numbers from 1 to 10 in the last program, what change could you make?

RUN your altered program again until you observe a 10.

Notice that the computer first finds the number value inside the parenthesis. Then the integer value of the number is taken.

Check your understanding by evaluating the following expressions, then RUN them on the computer to see if the computer agrees.

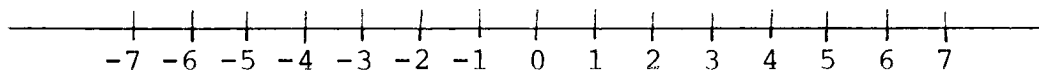
INT(3) = _____
 INT(4.69) = _____
 INT(0.99) = _____
 INT(-2.65) = _____
 INT(100*2.444) = _____
 INT(10*1.89) + 5 = _____

Several ways you might try to get the computer to print the numbers from 1 to 10 are shown below:

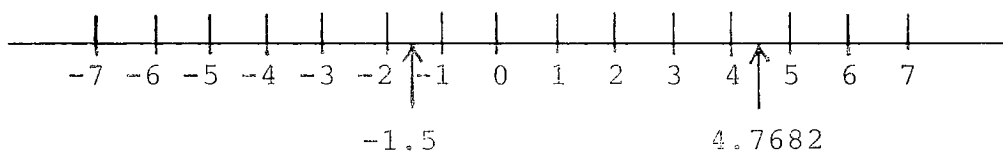
100 CALL CLEAR	100 CALL CLEAR
110 RANDOMIZE	110 RANDOMIZE
120 FOR T=1 TO 14	120 FOR T=1 TO 14
130 X=RND*10	130 PRINT INT(10*RND)+1
131 Y=INT(X)	140 NEXT T
132 Z=Y+1	
133 PRINT Z	100 CALL CLEAR
134 NEXT T	110 RANDOMIZE
	120 FOR T=1 TO 14
	130 PRINT INT(11*RND)
	140 NEXT T

Type the above statements into the computer and RUN the program again. Will they all produce numbers from 1 to 10?

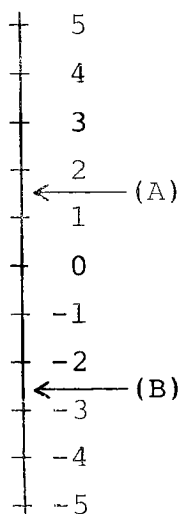
Have you ever seen a number line? One such line is shown below.



Notice that all the whole numbers are shown in order. The number 0 occurs at the center of the number line. As you move to the right along the number line, the numbers become steadily larger. However, moving to the left, the numbers become larger negative integers. Any number can be represented on the number line. For instance, the number 4.7682 would be a point between the points 4 and 5 on the number line. The number -1.5 would be located between -2 and -1.



Since most of us think of negative numbers as being in debt or "in the hole" you may also draw a number line like this:



With this view, the $\text{INT}(X)$ command always takes the lower whole number.

$$(A) \text{INT}(1.5) = 1$$

$$(B) \text{INT}(-2.5) = -3$$

EXERCISE 9-2

Write a program that will print 200 random whole numbers on the TV. The whole numbers should be in the range from 0 to 9 only.

When you get the above program to work properly, change the program so that it writes 100 whole numbers on the screen. This time the numbers should run from 50 to 99. Can you figure out how to do it?

EXERCISE 9-3

Now let's practice using the RND and RANDOMIZE commands. In Volume II, you learned how to make color squares on the screen. For example,

```
5 CALL CLEAR
10 CALL COLOR(2,7,7)
20 CALL HCHAR(16,10,40)
```

cause a red square to be printed at location row 16 and column 10.

Remember the first number in the color command is the character set number. The next number tells the foreground or character color. The last number specifies the background color. The first number in the HCHAR command specifies the row position of the character to be printed. The second number specifies column position. The last number tells which character to print.

Create a program that will print red squares at random positions on the screen.

What happens if you ask the computer to create a red square that would be off the screen?

Change your program so that the colors are selected randomly as well as their locations.

LESSON #10: IF-THEN

So far you have learned that a program moves steadily from lower numbered statements to higher numbered statements unless directed otherwise. FOR-NEXT loops or GOTO statements can be used to change the order of program execution.

GOTO statements are one example of branch statements.

GOTO statements cause a program to branch to a different location in the program. However, the GOTO command causes unconditional branching. This means that when the GOTO statement is executed, it always causes the program to jump to the other location.

In this lesson, you will learn about conditional branching, that is, branching which only occurs if some condition is met.

Suppose your mother says: "If it snows, then you may go sledding." Then your going out to sled depends upon a condition. The condition is that snow falls. If snow falls, you may sled. If snow doesn't fall, you may not sled.

The program on the next page shows an example of a conditional program statement (IF-THEN). Type the program into your computer and then RUN it.

```
10 CALL CLEAR
20 N=1
30 N=N+1
40 IF N=100 THEN 70
50 PRINT N;
60 GOTO 30
70 PRINT "END"
80 GOTO 80
```

This is how the program works:

The equal sign means "has the value of". Statement 20 sets 1 as the value of N. Statement 30 tells the computer to add 1 to the value of N. So N now equals 2. Statement 40 is the conditional statement. If N equals 100, statement 40 would cause a branch to statement 70. N equals 2 so no branch occurs. The program continues with the normal order. Statement 50 is executed next which prints the present value of N. Then statement 60 causes the program to branch back to statement 30. Eventually, N will be equal to 100. Then, statement 40 causes the computer to branch to statement 70.

Now change statement 40 to:

```
40 IF N=10 THEN 70
```

What will the program do this time?

RUN to check your answer.

Can you figure out what will happen if you eliminate 40?
RUN the program without line 40 to check.

Without the conditional statement, the program gets caught in an endless loop. Statements 70 and 80 never get executed. NOTE: DON'T ERASE THIS PROGRAM! We'll use it in this next section.

INPUT

There will be times when you would like to enter number values into a program while it is running. This saves time. You don't have to keep rewriting the program!

Now let's change lines 15 and 40 of that last program.

Type these statements into the computer:

```
15 INPUT X
40 IF N=X THEN 70
```

Statement 15 tells the computer to wait while you INPUT a number from the keyboard. It sets X equal to the number you type in.

Now RUN the program. When you see the ? printed on the screen, type a number into the computer, such as 28. Then press ENTER.

When statement 40 sees that N=X, (In this case N=28.) it will jump to statement 70.

Try RUNNING the program several times. Each time input a different number. When you are satisfied that you know how the program works, go on to the next page.

Now let's use the INPUT and RANDOM commands to write a simple computer game. The computer will pick a random number from one to ten. It will then ask you to guess the number. If you guess wrong, the computer will ask you to guess again. If you pick the right number, it will tell you so.

Type in the following program statements:

```
10 CALL CLEAR
20 RANDOMIZE
30 N=INT(10*RND)+1
40 PRINT "I AM THINKING OF A NUMBER"
50 PRINT "BETWEEN 1 AND 10"::::::
60 PRINT "GUESS WHAT IT IS."
70 INPUT X
80 IF X=N THEN 120
90 CALL CLEAR
100 PRINT "NO, TRY AGAIN.":
110 GOTO 70
120 PRINT "CONGRATULATIONS!"
130 PRINT "YOU GUESSED THE NUMBER!"
140 FOR I=1 TO 1000
150 NEXT I
160 CALL CLEAR
170 GOTO 30
```

Try to figure out how the program works before you RUN it. Then RUN the program and play the game a few times.

Fill in the following blanks.

If statement 30 has just been executed and $RND=0.659$, what number will the computer set N equal to? _____

Now when statement 70 is executed, suppose the player inputs the number 1. What will be the next 7 statement lines executed?

_____' _____' _____' _____' _____' _____' _____'

Suppose that the number 8 had been entered instead of 1 when statement 70 was executed. What would have been the next 5 statements to be executed?

_____' _____' _____' _____' _____'

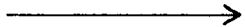
What is the purpose of statements 140 and 150? _____

Now let's spruce up the program a little. To put a lot of flash into the program, type in the following additional statements:

```

80 IF X=N THEN 118
118 FOR J=1 TO 10
119 CALL CLEAR
120 FOR K=1 TO 50
121 NEXT K
122 PRINT "CONGRATULATIONS!"
123 FOR L=1 TO 50

```

Keep going. 

```
124 NEXT L
```

```
125 NEXT J
```

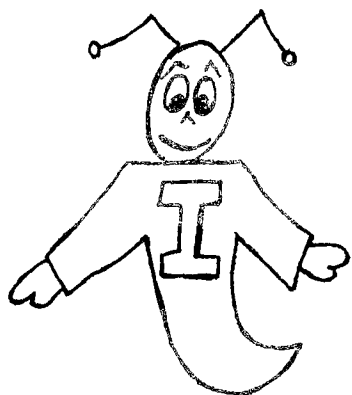
Now RUN the program and watch the flash.

NOTE: Save this program for a while . . . don't type NEW.)

There are other ways in which the program could be made better. For instance, we could have the computer tell us whether a guess is too high or too low. To do this, we need to learn about two other conditional statements.

The following examples will illustrate these other conditional statements.

Instead of using the words LESS THAN or GREATER THAN to tell the relation between two numbers, we may use the symbols $<$ and $>$.



$>$ STANDS FOR GREATER THAN.

$8 > 4$

$129 > 15$

$<$ STANDS FOR LESS THAN.

$4 < 8$

$15 < 129$

Now you try it. Put a $>$ or $<$ in the following blanks to express the correct relation between numbers.

6 <u> </u> $>$ <u> </u> 2	100 <u> </u> 1000
4.3 <u> </u> $<$ <u> </u> 4.4	32 <u> </u> 6.4
16 <u> </u> 100	5 <u> </u> 5.1
2 <u> </u> 32	17.6 <u> </u> 12.15

Now let's use these new symbols in IF-THEN statements. Add the following program lines to your last program.

```

81 IF X > N THEN 90
82 CALL CLEAR
83 PRINT "YOUR GUESS IS TOO LOW!"
84 PRINT "TRY AGAIN."
85 GOTO 70
90 CALL CLEAR
95 PRINT "YOUR GUESS IS TOO HIGH!"
100 PRINT "TRY AGAIN."
110 GOTO 70

```

The program will work the same way as it did before for statements 10 through 80. Then if $X=N$, 80 will cause the program to jump down to the CONGRATULATIONS statement. If X doesn't equal N , then statement 81 will be executed next. If X is greater than N , statement 81 will cause the program to jump down to statement 90 to clear the screen and then print the TOO HIGH statement. And finally, if X doesn't equal N and if X isn't GREATER THAN N , then the program prints the TOO LOW message.

RUN the program to see that it works properly.

Next, we will add a few program lines so that the computer can keep a count of misses.

First type:

```
RES 10,10
```

Then LIST the program.

The program should be as follows:

```
10 CALL CLEAR
20 RANDOMIZE
30 N=INT(10*RND)+1
40 PRINT "I AM THINKING OF A NUMBER"
50 PRINT "BETWEEN 1 AND 10.":::
60 PRINT "GUESS WHAT IT IS."
70 INPUT X
80 IF X=N THEN 180
90 IF X > N THEN 140
100 CALL CLEAR
110 PRINT "YOUR GUESS IS TOO LOW!"
120 PRINT "TRY AGAIN."
130 GOTO 70
140 CALL CLEAR
150 PRINT "YOUR GUESS IS TOO HIGH!"
160 PRINT "TRY AGAIN."
170 GOTO 70
180 FOR J=1 TO 10
```

```
190 CALL CLEAR
200 FOR K=1 TO 50
210 NEXT K
220 PRINT "CONGRATULATIONS!"
230 FOR L=1 TO 50
240 NEXT L
250 NEXT J
260 PRINT "YOU GUESSED THE NUMBER!"
270 FOR I=1 TO 1000
280 NEXT I
290 CALL CLEAR
300 GOTO 30
```

RUN your program. If it doesn't work, check it to see that it looks just like the one above. Correct any errors.

Now to make the computer count the misses, type in the following line statements:

```
35 C=0
115 C=C+1
155 C=C+1
255 PRINT "YOU HAD";C;"MISSES."
```

Statement 35 resets the number of misses count before every new game. Statements 115 and 155 cause the count to be increased by 1 after every miss.

RUN the program to see that it works properly.

You have learned quite a lot about IF-THEN statements already, but there is more . . .

You have already learned that > means greater than, < means less than, and = means has the value of. We may combine the above symbols to express other relations.

For example:

<= means less than or equal to
 >= means greater than or equal to
 <> means less than or greater than
 OR not equal to

Type in the following program and then RUN it.

```
5 CALL CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT N
30 IF N<>135 THEN 10
40 PRINT "THAT'S IT!"
50 END
```

Notice that if the number you input doesn't equal 135, then statement 30 sends the program back to statement 10. Statement 30 is saying:

IF N DOESN'T EQUAL 135, THEN GOTO 10.

When you finally do input 135, the condition in statement 30 is not met. So the computer continues with statement 40.

EXERCISE 10-3

Write a program that will plot a red square on a black screen at a position specified by INPUT commands.

Use IF-THEN statements to keep the user from INPUTing positions which would be off the screen.

Use PRINT statements to tell the user how to operate the program.

Use the lines below to plan your program if you wish.

A sheet of graph paper is included on the next page.

LESSON #11: SOME MATH

YOUR TI COMPUTER CAN COME IN VERY HANDY WHEN YOU HAVE TO ADD, SUBTRACT, MULTIPLY, OR DIVIDE. IT CAN ALSO PERFORM OTHER MATHEMATICAL WIZARDRY WHICH YOU WILL LEARN ABOUT LATER.

Right now, let's look at the keyboard symbols for these operations:

addition	+
subtraction	-
multiplication	*
division	/

Type in the following commands, pressing ENTER after each.

PRINT 6+4

PRINT 6-4

PRINT 6*4

PRINT 6/4

Notice that the computer prints the result of each mathematical problem immediately. But if you put a statement number in front of the command, the computer does not perform the math until you type RUN and then press ENTER. Commands entered without a statement number are said to be in the immediate mode of computer operation.

Now type in the following commands. Try to figure out what the computer will print before you enter each command. Can you figure out in what order the computer does each problem?

	RESULT
PRINT 6*4-3	_____
PRINT 6*4/3	_____
PRINT 6-4*3	_____
PRINT 6+4-3	_____
PRINT 6-4+3	_____
PRINT 4-3*6	_____
PRINT 4-3/6	_____

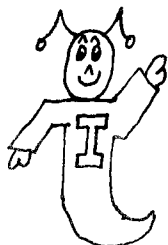
When the computer does more than one math operation, it follows certain rules. The computer performs multiplication and division before addition and subtraction. Otherwise, the computer performs operations from left to right.

For example:

$$6-4*3 \text{ equals } 6-12 \text{ equals } -6$$

Notice that the multiplication $4*3=12$ was done first, then the subtraction was done: $6 - 12 = -6$.

Parenthesis may be used to change the order of doing mixed calculations because the computer follows another rule:



CALCULATIONS INSIDE A SET OF
PARENTHESIS ARE PERFORMED
FIRST.

For example:

$5+6*2=$	$(5+6)*2=$
$5+12=$	$11*2=$
17	22

In the example on the right, the addition is performed first because the parenthesis tell the computer to do the calculation inside the parenthesis first.

Write down the answers to the following problems. Then check your answers using the computer.

$5+6*2+3=$	_____
$5+(6*2)+3=$	_____
$(5+6)*(2+3)=$	_____
$5+(6*2+3)=$	_____
$2*3+6*4-8=$	_____
$2*(3+6*4)-8=$	_____
$2*(3+6*4-8)=$	_____

Now find the \wedge key. We're going to use it in just a moment to help us take a short cut. For now, type in the following command:

```
PRINT 2*2*2*2*2*2*2*2*2*2    (That's ten 2's!)
```

What is the answer? _____

Now enter the following command:

```
PRINT 2^10
```

The answer is _____.

Do the same for each of the following:

	<u>ANSWER</u>
PRINT 6*6*6*6	_____
PRINT 6^4	_____
PRINT 4*4*4	_____
PRINT 4^3	_____
PRINT 3*3*3	_____
PRINT 6*2^3	_____

The \wedge symbol tells how many times to multiply a number.

For example, 2^8 multiplies the number 2 together 8 times.

$$2 * 8 = 16 \quad \text{but}$$

$$2^8 = 2*2*2*2*2*2*2*2 = 256$$

2^8 is read as 2 to the eighth power. This is called raising 2 to the eighth power.

The computer follows another rule. Power calculations are performed before multiplication or division.

Try to figure out the answers to the following problems.

Then check your answers using the computer.

$$3^5 = \underline{\hspace{2cm}}$$

$$5^3 = \underline{\hspace{2cm}}$$

$$5^3 + 1 = \underline{\hspace{2cm}}$$

$$5^3 + 2 = \underline{\hspace{2cm}}$$

$$5^2 + 1 = \underline{\hspace{2cm}}$$

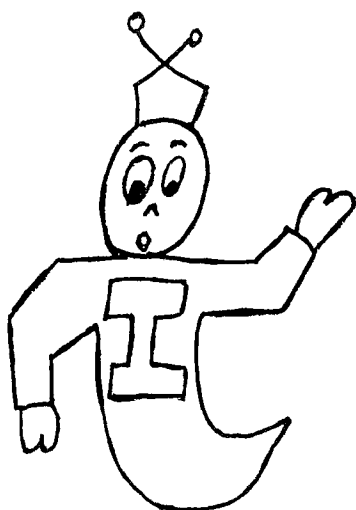
$$5^{(2+1)} = \underline{\hspace{2cm}}$$

$$5^{(1*3)} = \underline{\hspace{2cm}}$$

$$5^1 * 3 = \underline{\hspace{2cm}}$$

$$2 * 5^3 = \underline{\hspace{2cm}}$$

$$(2 * 5)^3 = \underline{\hspace{2cm}}$$



REMEMBER:

CALCULATIONS DONE INSIDE PARENTHESIS
ARE DONE FIRST.

^ IS DONE BEFORE * OR / ,

* OR / IS DONE BEFORE + OR - ;

OTHERWISE FROM LEFT TO RIGHT .

EXERCISE 11-1

You have seen how an INPUT statement can be used to input data from the keyboard. The INPUT command can also be used to print a message.

For example:

```
10 INPUT "INPUT X":X
```

will cause the message

```
INPUT X
```

to be printed. Then the computer will wait for the user to type in a number and press ENTER. The message in an INPUT statement must always be enclosed in quotes and must always end with a colon (:), as shown above.

Write a program that uses two INPUT statements. The first one should print "INPUT X". The second should print "INPUT Y". Then the program should calculate X times Y and print out the answer in the form "X*Y=";*(answer)*.

EXERCISE 11-2

An INPUT statement can be used to input more than one number at a time. For example:

```
10 INPUT X,Y,Z
```

This INPUT statement will wait for the user to type in three numbers separated by two commas:

```
? 6,3,5      (ENTER)
```

Write a program that inputs two numbers using one INPUT statement. This statement should print the message: "X=,Y=," so that the user knows what to input. Then have the computer figure out which of the two numbers is the largest by using IF-THEN statements. For instance, IF X>Y THEN go to the statement which prints out the message "X>Y". IF X<Y, THEN the computer should print out an appropriate message. IF X=Y, the computer should print out a different message.

LESSON #12: CHAR

In this lesson, you are going to learn more about GRAPHICS. In particular, you will be studying the CHAR command which can be used to make up new characters. Fantastic! So let's get started. . .

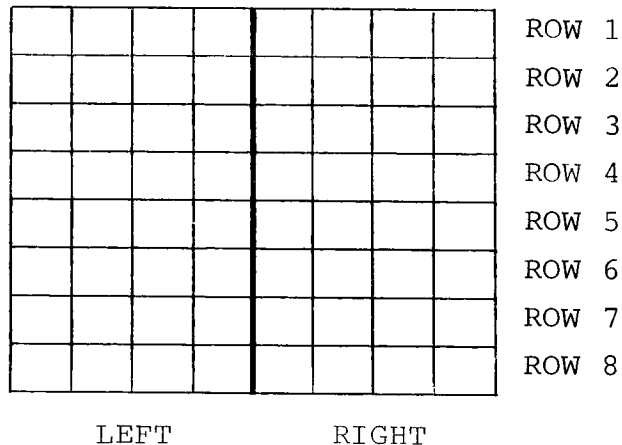
Type in and enter the following:

```

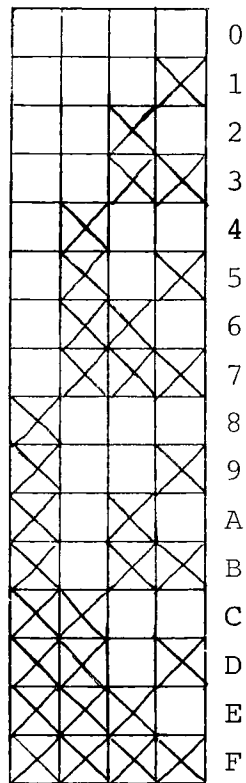
5 CALL CLEAR
10 CALL COLOR(9,7,7)    (Set 9 is red on red.)
20 CALL HCHAR(12,12,96) (Print character 96 at row 12,
                        column 12. Note codes 96 through
30 GOTO 30              103 are in set 9.)

```

Now RUN the program. The red square that has just been plotted is really made up of 64 tiny squares all lit up in red. In just a little while, you will be able to control which of these tiny squares get turned on and which get turned off. An enlarged picture of the 64 tiny squares is shown below. There are eight rows of eight horizontal squares ($8 \times 8 = 64$). For our purposes, we will consider the block to be split into a left side and a right side, as shown.



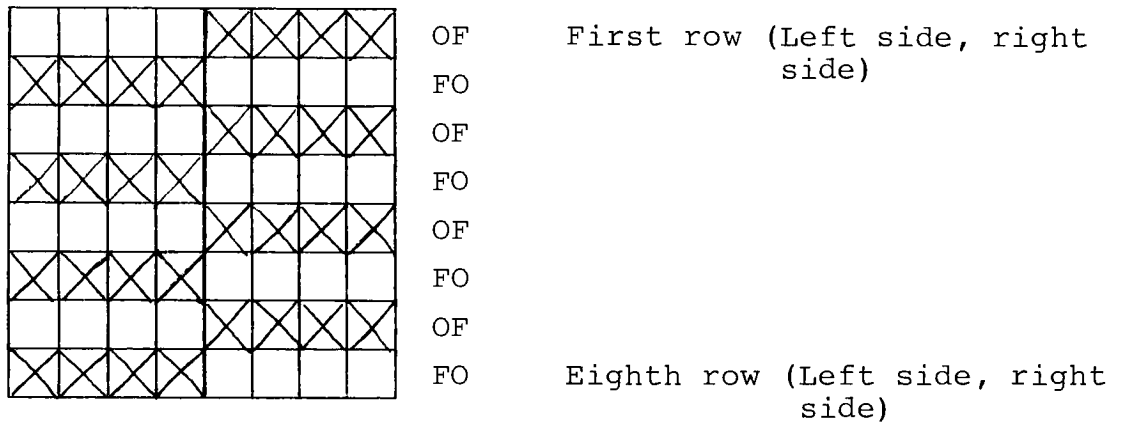
Now, by using letters and numbers, we can tell the computer which squares to turn on. The picture below shows the code for turning on the squares in a single row.



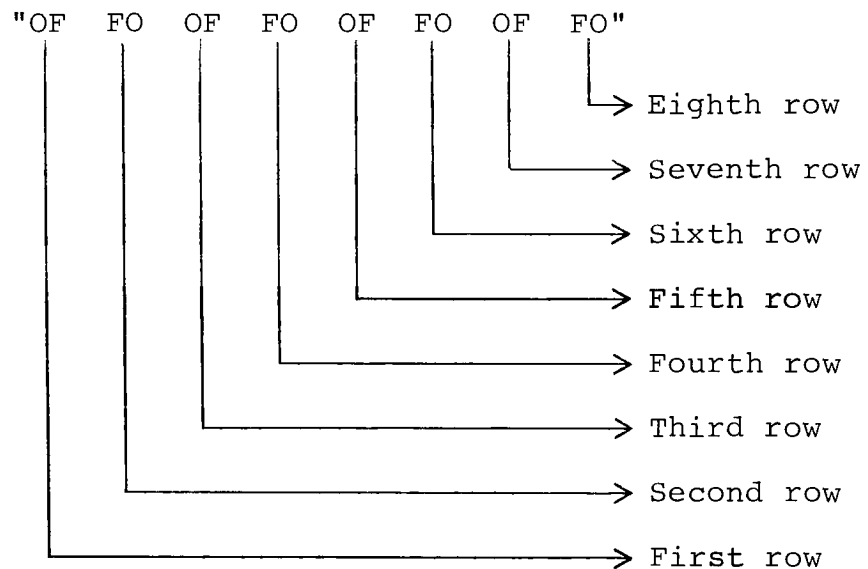
RIGHT OR
LEFT SIDE

NOTE: A 0 (zero) turns no squares on, a 1 turns the first square on, a 2 turns the second square on, a 3 turns the first two squares on, and so on up to F, which turns on all four squares on one side of the block. Thus, to tell which squares are to be turned on in one whole row consisting of two sides, two symbols are needed.

Therefore, since there are 8 rows, 16 symbols are needed to completely define a character within the block. An example is shown below:

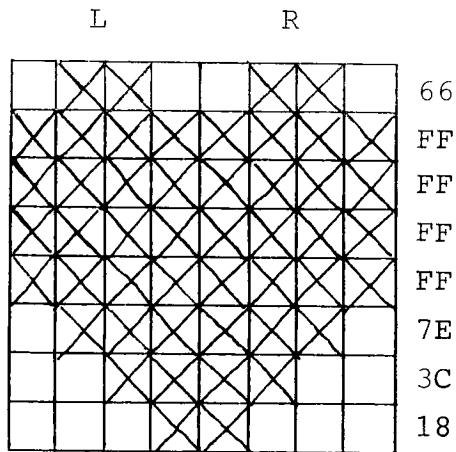


Next, all of the symbols defining a block are written together in one line as shown below:



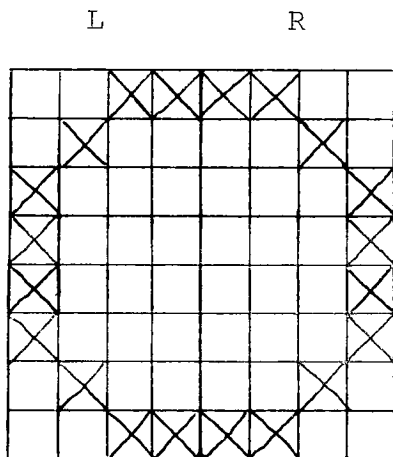
Quotation marks must go around the symbols.

The next example shows the symbols that would be needed to make a heart.



"66FFFFFFFF7E3C18" *(The letters must all be capitalized.)*

Here's one for you to try:



Fill in the blank with the proper string of characters.

" _____ "

The next step needed to print a special character is to define the new character using the CHAR command. For example,

```
CALL CHAR(96,"66FFFFFFFF7E3C18")
```

(Remember, all letters must be capitalized.)

defines character 96 to be the string of symbols which make up the heart.

Now type in the following lines and then RUN them.

```

10 CALL CLEAR
20 CALL COLOR(9,2,4)
30 CALL CHAR(96,"66FFFFFFFF7E3C18")
40 CALL HCHAR(12,3,96,10)
50 GOTO 50

```

Statement 20 defines the color for set 9: black on green. Statement 30 tells the computer that character 96 (which is in set 9) is to represent a heart. Statement 40 prints character 96 (the heart) 10 times in a horizontal row beginning at row 12 and column 3.)

Now let's tell the computer to print the circle you defined earlier. Add the following statements, then RUN the program again.

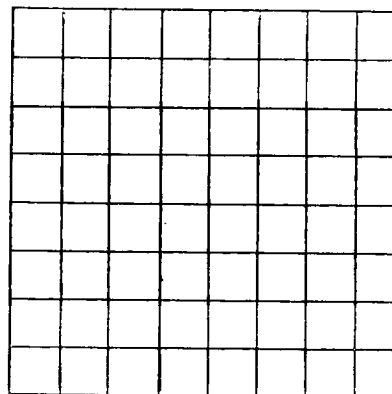
```

31 CALL CHAR(97,"3C4281818181423C")
41 CALL VCHAR(1,7,97,16)

```

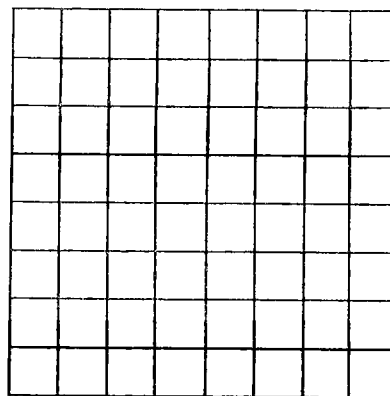
Now try these. Fill in the boxes that should be filled in.

"0F0E0C08103070F0"



LEFT RIGHT

"80C0E0F00F070301"



LEFT RIGHT

The program below makes use of the above two strings to produce a moving character. Type in the statements and RUN the program.

```

10 CALL CLEAR
20 CALL SCREEN(16)
30 CALL COLOR(2,2,16)
40 CALL CHAR(40,"0F0E0C08103070F0")
50 CALL CHAR(41,"80C0E0F00F070301")
60 FOR I=1 TO 1000
70 CALL HCHAR(12,12,40)
80 FOR J=1 TO 25
90 NEXT J
100 CALL HCHAR(12,12,41)
110 FOR J=1 TO 25
120 NEXT J
130 NEXT I

```

Statements 60 through 130 form a FOR-NEXT loop which alternately prints one character then the other one. Statements 70 and 100 do the actual printing. These print statements are separated by delay loops which slow down the action.

The next sample program shows how one can cause a character to move from place to place on the screen. Type it in and RUN it.

```
10 CALL CLEAR
20 CALL COLOR(9,13,12)
30 CALL CHAR(96,"003C7E7E7E7E3C00")
40 CALL SCREEN(12)
50 FOR I=2 TO 32
60 CALL CLEAR
70 CALL HCHAR(12,I,96)
80 FOR J=1 TO 10
90 NEXT J
100 NEXT I
110 FOR I=32 TO 2 STEP -1
120 CALL CLEAR
130 CALL HCHAR(12,I,96)
140 FOR J=1 TO 10
150 NEXT J
160 NEXT I
170 GOTO 50
```

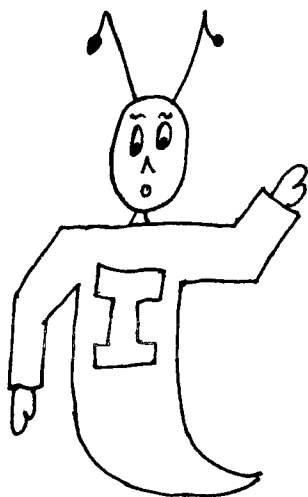
The ball appears to move because it is plotted at one position, erased, then plotted at the next position, erased, and so on. Statements 50 through 100 cause the ball to move from left to right. Statements 110 to 160 cause the ball to move from right to left.

TI basic allows you to make notes within a program so that you can remember what each part of the program does. These notes are called REMARKS.

Add the following statements to the last program:

```
5 REM THIS PROGRAM MOVES A BALL
6 REM ACROSS THE SCREEN
25 REM STATEMENT 30 DEFINES THE BALL
45 REM MOVE THE BALL RIGHT
105 REM MOVE THE BALL LEFT
```

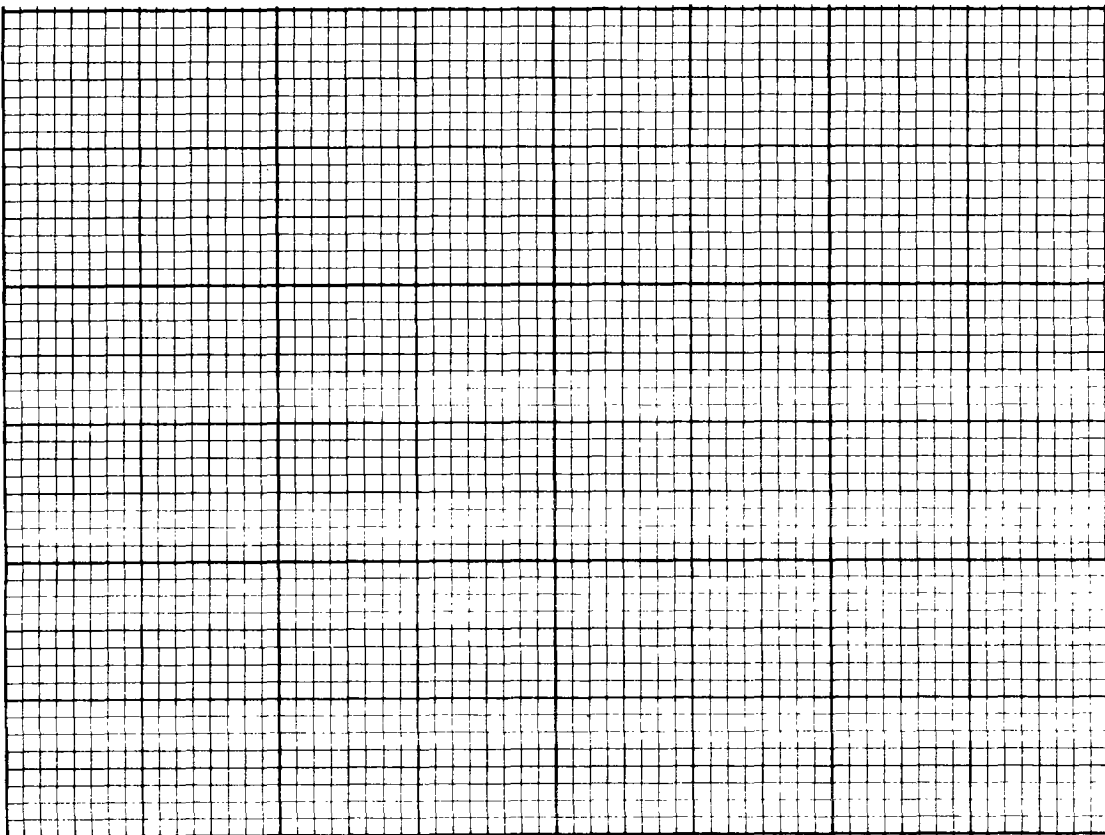
LIST the program to see how the statements fit in. Then RUN the program. Notice the REMark statements do not affect program operation.

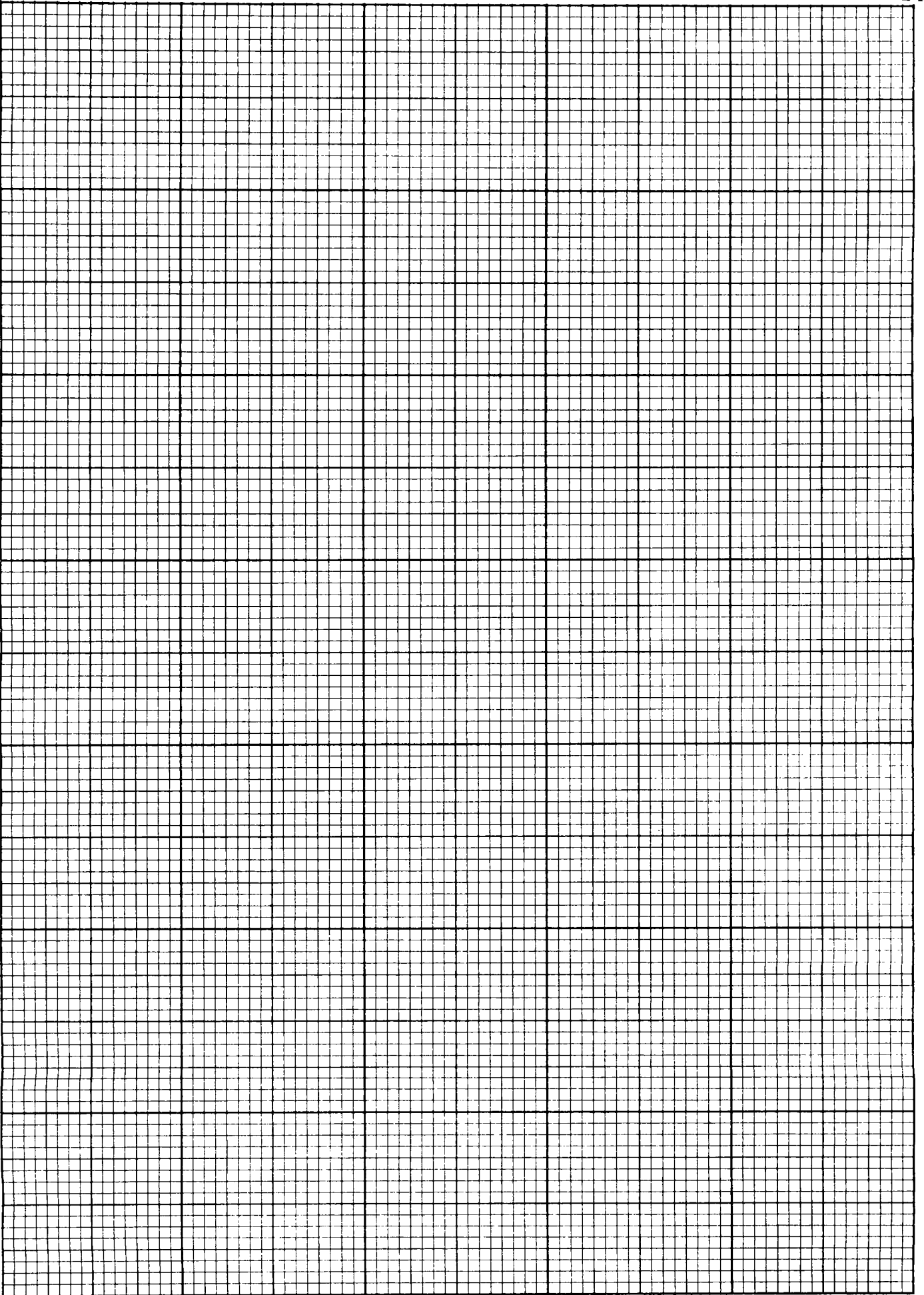


REMEMBER! TO DEFINE YOUR OWN CHARACTER, FIRST DEFINE THE SET COLOR FOR YOUR CHARACTER. THEN DEFINE THE CHARACTER USING CALL CHAR(CODE,"STRING"). FINALLY, PRINT THE CHARACTER USING CALL HCHAR(ROW,COLUMN,CODE,NUMBER OF CHARACTERS) OR VCHAR(ROW,COLUMN, CODE,NUMBER OF CHARACTERS).

EXERCISE 12-1

Now it's your turn. Create a character of your own and use it to make an interesting design on the screen.





_____ name

VOLUME III REVIEW QUIZ

Fill in the blank with the correct word from the ANSWER POOL.
(If you get stuck, turn back to the correct page and review.)

ANSWER POOL

INT	immediate	<=
4	14	random numbers
INPUT X	IF-THEN	I=I+1
75	<	<>
>	-4	16
RANDOMIZE	PRINT 50*RND	
CALL CHAR(96,"0F1E2D3C4B5A0000")		"FFFFFFFFFFFFFFFF"

1. _____ are numbers which follow no obvious pattern (p.99).
2. _____ causes a different sequence of random numbers to be generated every time the program is RUN (p.100).
3. _____ causes whole numbers (integers) to be generated every time the program is RUN (p.102).
4. _____ is a statement that would generate random numbers between 1 and 50 (p.103).
5. _____ equals INT(4.999) (p.104).
6. _____ equals INT(-3.02) (p.104).
7. _____ is a conditional program statement (p.108).

8. _____ is a statement which causes I to be equal to one more than it was before (p.109).
9. _____ is a command that can be used to input a number from the keyboard and assign it to the variable X (p.110).
10. _____ is the symbol for greater than (p.113).
11. _____ is the symbol for less than (p.113).
12. _____ is the symbol for less than or equal to (p.117).
13. _____ is the symbol for not equal to (p.117).
14. _____ is the computer mode in which commands are executed as soon as you enter them (p.124).
15. _____ equals $16-20/5+2$ (p.125).
16. _____ equals $(16-2)*5+5$ (p.126).
17. _____ equals $8+2\wedge 3$ (p.127).
18. _____ is the statement which would define "0F1E2D3C4B5A0000" to be the character whose code is 96 (p.135).
19. _____ is the string of 16 symbols which would represent a solid block of 64 lighted dots (p.133).

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

YELLOW PROJECT 1

CREATE a program that will allow one to INPUT five numbers. Print the even ones in one column and the odd ones in a second column.

YELLOW PROJECT 2

Write a program that will have the computer pick 40 random integers and print them in a table with four columns.

YELLOW PROJECT 3

CREATE a program that will cover the screen with lights, and then turn off lights that will draw a rocket ship. Use colors if your computer supports color. Be CREATIVE!

YELLOW PROJECT 4

CREATE a program that will first draw a square, then have a dot of light travel downward inside the box until it hits the side. Then have it stop and go upward until it hits the top side. Have it go down and up five times.

YELLOW PROJECT 5

CREATE a program that represents a die being rolled. Integers from 1 to 6 should occur randomly. Have your program pretend to roll the die 1000 times and keep track of how many times each integer is selected. Print out the results in a table.

YELLOW PROJECT 6

CREATE a program that will allow 360 degrees of direction to be represented by numbers from 0 to 255. For example: 90 degrees would be 63; 180 degrees would be 127; 270 degrees would be 191; 360 degrees would be 255.

Your program should calculate and print the direction when you INPUT the number.

If you have game paddles with your computer, use them to enter the numbers instead of the INPUT command.

YELLOW PROJECT 7

Can you CREATE a program that will complete and display this chart on the screen?

X	X*X	X*X*X	X*X*X*X
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	
5	25		
6			
7			
8			
9	81		

YELLOW PROJECT 8

CREATE a program that will randomly pick nine numbers between 1 and 10, but will not pick the same number more than once. Have your program display the numbers on the screen as it selects them.

Send us a LIST of your working program, and we will send you your YELLOW PROGRAMMER III card. This YELLOW page must be included.

The cost of creating for you a PROGRAMMER'S card is included in the cost of this manual. By sending in this colored page, we know this cost has already been paid. If you are sharing this manual and cannot remove this page, then you may also receive your PROGRAMMER'S card and have your name listed in the newsletter by sending us a copy of your working program for this project along with \$2.00 and a self-addressed, stamped envelope.

Send to:

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

Your name _____

Phone # _____

Address _____

City, State _____

TI-99/4A

Zip _____ Birthdate _____

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

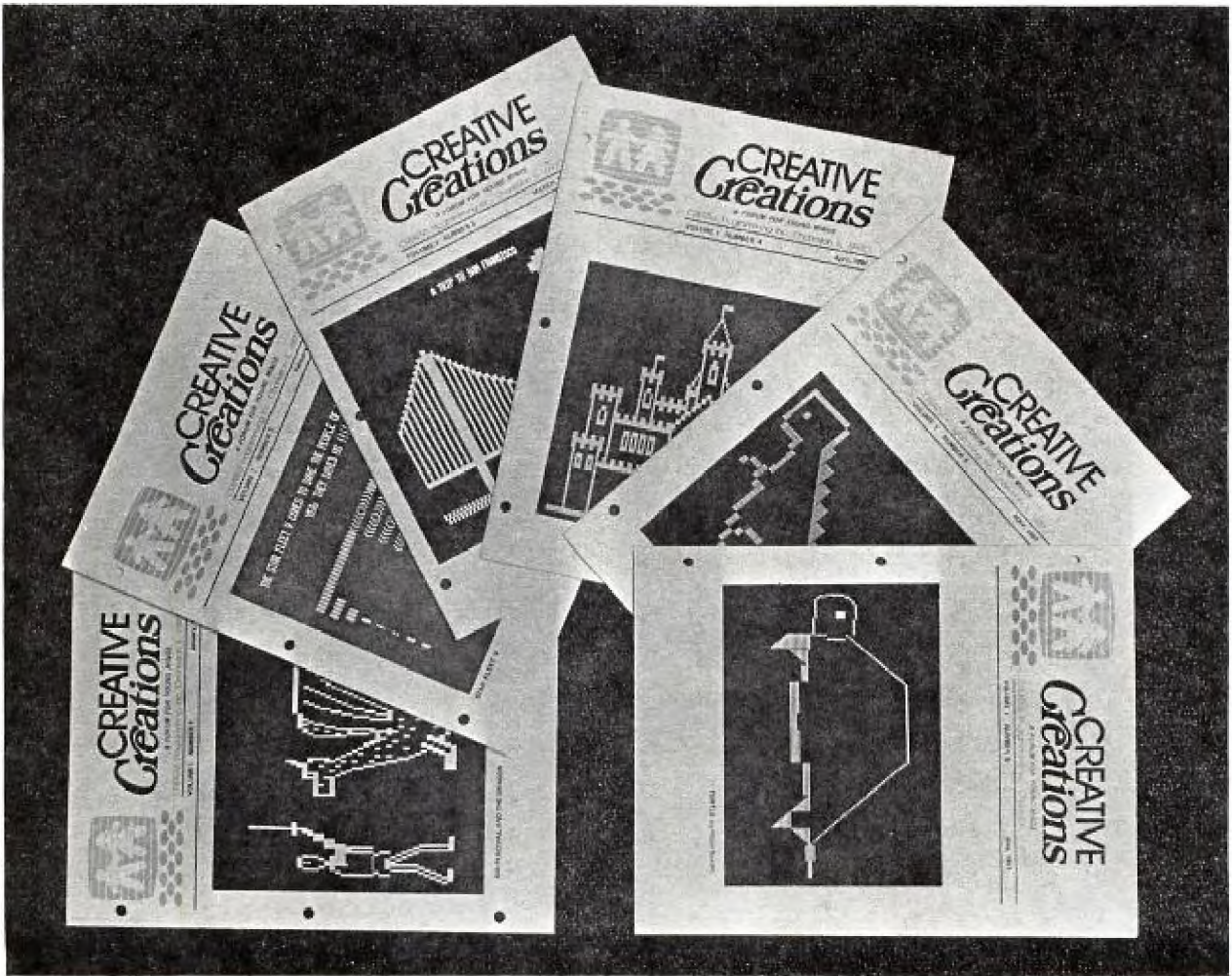
CREATIVE Creations

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 _____

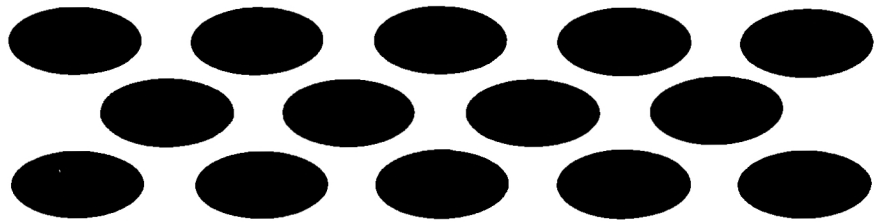
City _____ State _____ Zip _____

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
604 6th St, Charleston, IL 61920
WETHERS
A DIVISION OF THE WETHERS GROUP
A SUBSIDIARY OF THE WETHERS GROUP

ADDENDUM FOR CREATIVE MINDS

Function Keys

Both the TI-99/4 and the TI-99/4A consoles come with keyboard overlays identifying special computer functions (**BACK**, **BEGIN**, **QUIT**, etc.). To access TI-99/4A overlay functions, as well as any function or symbol that appears on the *front* of a key, hold down the **FCTN** key while pressing the appropriate function or symbol key.

When you are using the TI-99/4A console, note that, with the applications software packages, you should type parentheses, (), and not brackets, [], unless the owner's manual states otherwise. Also, the owner's manual enclosed with an applications software package may reference only the TI-99/4 function keys (**SHIFT C** for **CLEAR**, **SHIFT R** for **REDO**, etc.). Most Command Module, Diskette, and Cassette programs are compatible with both consoles; however, the keystroke sequences used in accessing the special functions of these programs differ somewhat for each keyboard.

The following chart shows the relationship between the function keys on the TI-99/4 and the TI-99/4A for most software applications.

FUNCTION KEYS

<i>Name</i>	<i>TI-99/4 Keys</i>	<i>TI-99/4A Keys</i>
AID	SHIFT A	FCTN 7
CLEAR	SHIFT C	FCTN 4
DELeTe	SHIFT F	FCTN 1
INSert	SHIFT G	FCTN 2
QUIT	SHIFT Q	FCTN =
REDO	SHIFT R	FCTN 8
ERASE	SHIFT T	FCTN 3
LEFT arrow	SHIFT S	FCTN S
RIGHT arrow	SHIFT D	FCTN D
DOWN arrow	SHIFT X	FCTN X
UP arrow	SHIFT E	FCTN E
PROC'D	SHIFT V	FCTN 6
BEGIN	SHIFT W	FCTN 5
BACK	SHIFT Z	FCTN 9
ENTER	ENTER	ENTER

Although the preceding list of function keys applies to the majority of software packages, the keystroke sequences required for certain Command Modules are different. The sequences for each of these modules are listed here.