

This chapter continues developing programs that demonstrate the graphics capabilities of your computer. The programs deal with the use of color, animation and the generation of your own graphics characters on the screen.

The intent in this chapter is to give you some hints and examples that will help you expand your enjoyment and use of your computer. As you begin to develop your own graphics applications, you may want to refer to this material for ideas on how to approach the use of color and graphics. In time, you'll discover other ways to create specific programs and simulations. But for now the programs that are given here will demonstrate several techniques you'll enjoy using.

## Blocks of Color

In previous chapters we've experimented with several programs that placed color lines or squares on the screen. The program below shows you how to create larger blocks of color.

```

NEW
10 INPUT "COLOR CODE?":C
20 IF C<1 THEN 10
30 IF C>16 THEN 10
40 CALL CLEAR
50 CALL COLOR(2,C,C)
60 FOR I=1 TO 4
70 CALL VCHAR(2,I+2,42,4)
80 CALL VCHAR(19,I+2,42,4)
90 CALL VCHAR(2,I+24,42,4)
100 CALL VCHAR(19,I+24,42,4)
110 CALL VCHAR(12,I+13,42,4)
120 NEXT I
130 INPUT "PRESS ENTER KEY":
KEY$
140 GOTO 10
    
```

Check for valid color code.

Set color.

Each CALL VCHAR places a vertical strip of color on the screen.

Wait here.

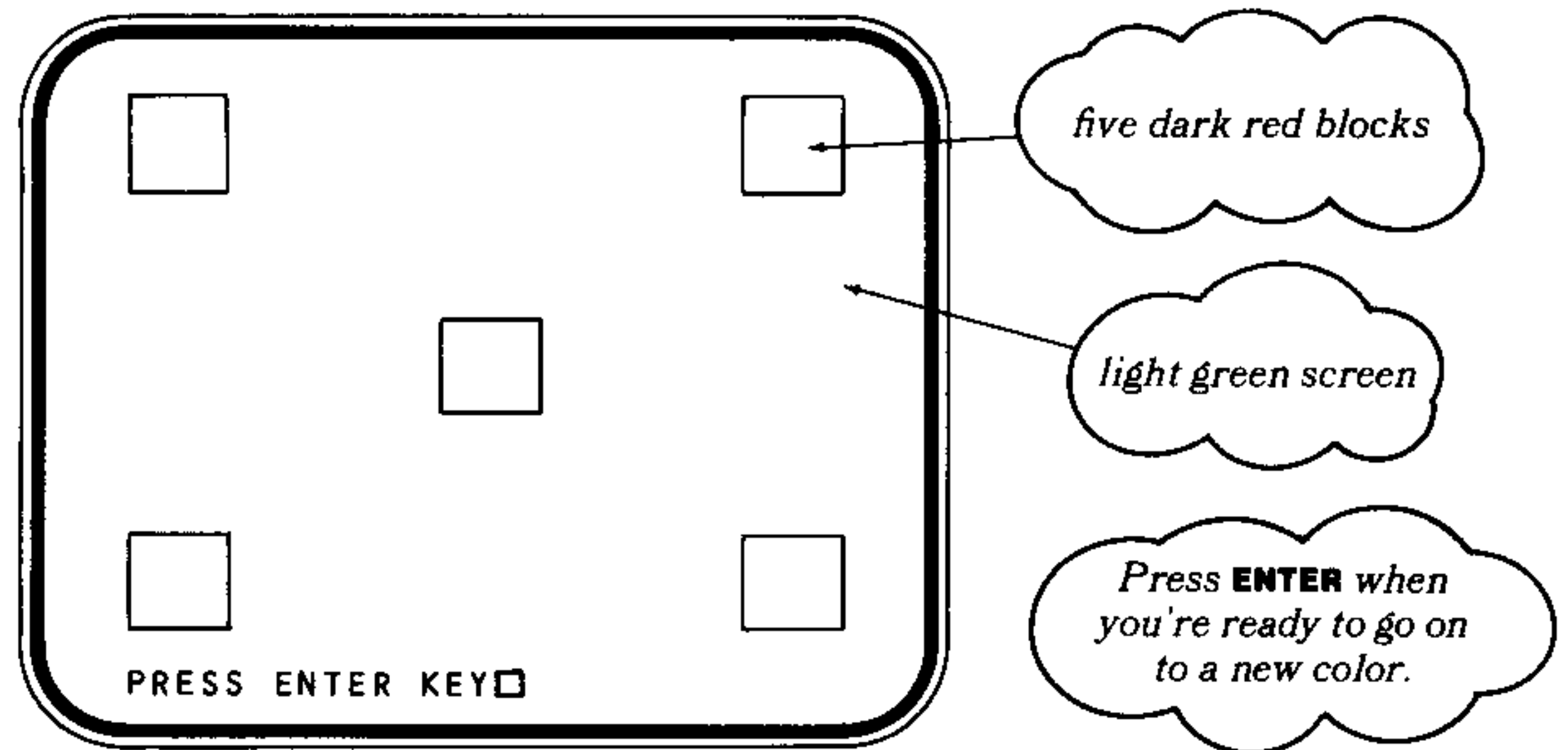
First the program stops and asks you to input a color code. (See *Appendix C* for the list of colors that correspond to the valid color codes, 1 through 16.) When you enter a code, the screen clears, the blocks of color are displayed, and the program waits for you to press a key before continuing. Notice the special use of the INPUT statement in line 130. We are just using INPUT to stop the program until we are ready to go on.

Now clear the screen and run the program. First you'll see

COLOR CODE?□

flashing cursor

Remember that color code 1 is transparent and code 4 is the screen color in the Run Mode. So let's enter 7 (dark red) as our first color code. When you press **ENTER**, you'll see this:

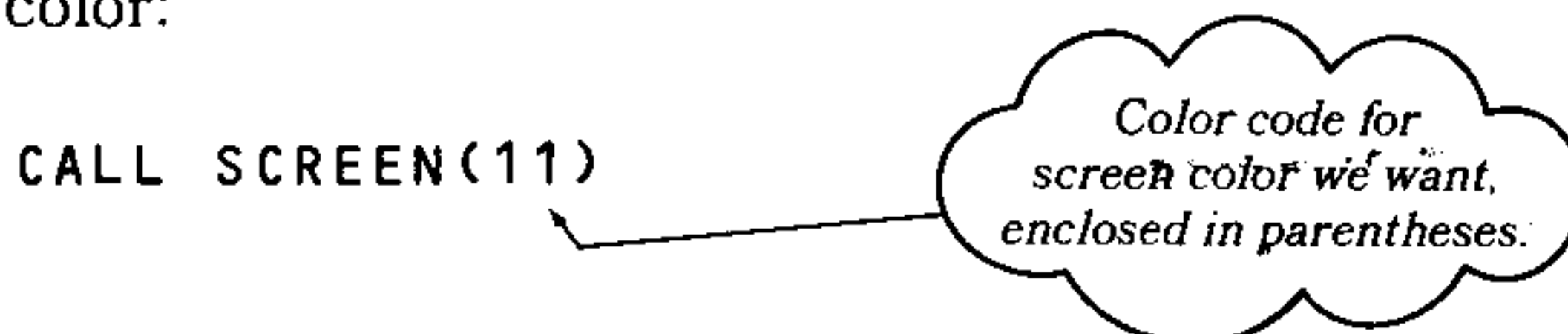


Experiment with several colors. Find the ones that produce clear sharp images against the normal color of the screen. Can you see how this technique could be used to create checkerboard patterns or the board area for a game like tic-tac-toe?

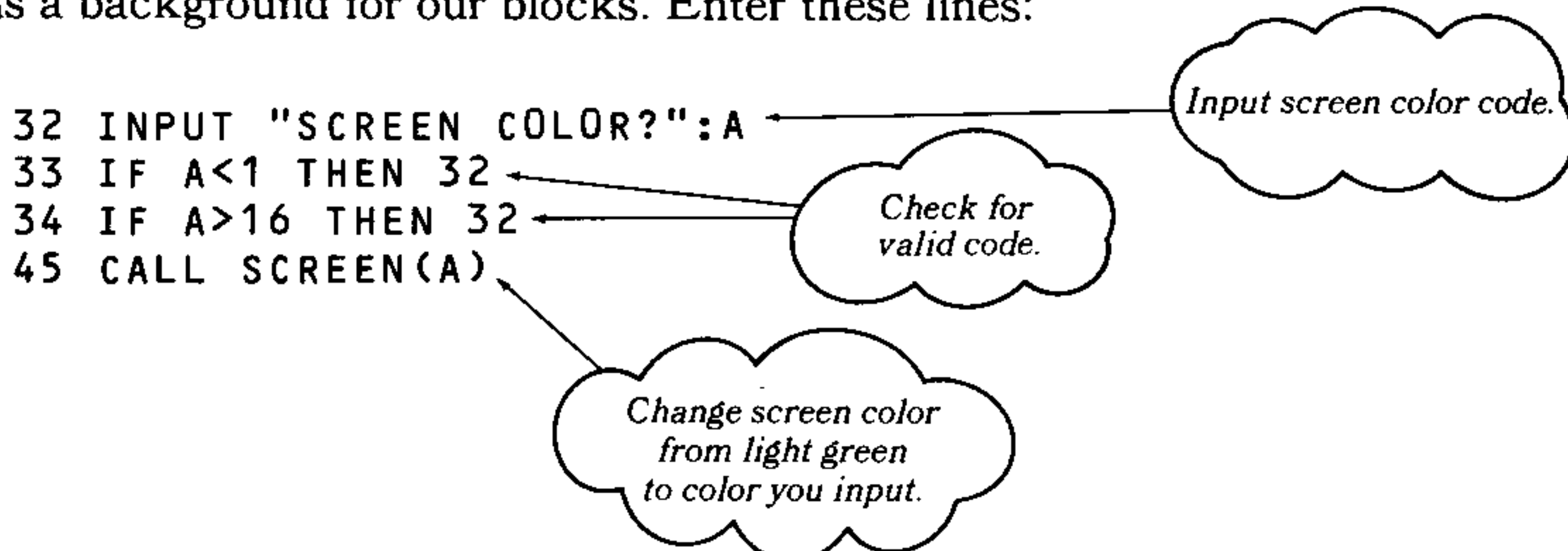
(Each block in this program is four "characters" wide and four "characters" high. If you'd like to see the "characters," press **CLEAR** to stop the program.)

### The CALL SCREEN Statement

So far, the color of the screen in the Run Mode has always been light green. Suppose, however, that we would prefer a different color as a background for our color design. Easy! All we have to do is to add a simple statement that changes the Run Mode screen color:



Let's edit the color block program we just entered so that we can use a different screen color as a background for our blocks. Enter these lines:



Now run the program again. This time, you'll be asked to enter *two* color codes. The first code determines the color of the blocks; the second sets the color of the screen.

# 5

## Experiment!

Experiment with different color combinations. Which give you the sharpest, clearest design? Which are most pleasing in an artistic sense? Then try changing the block design produced by the program. For example, can you make the blocks rectangular?

## Patterns

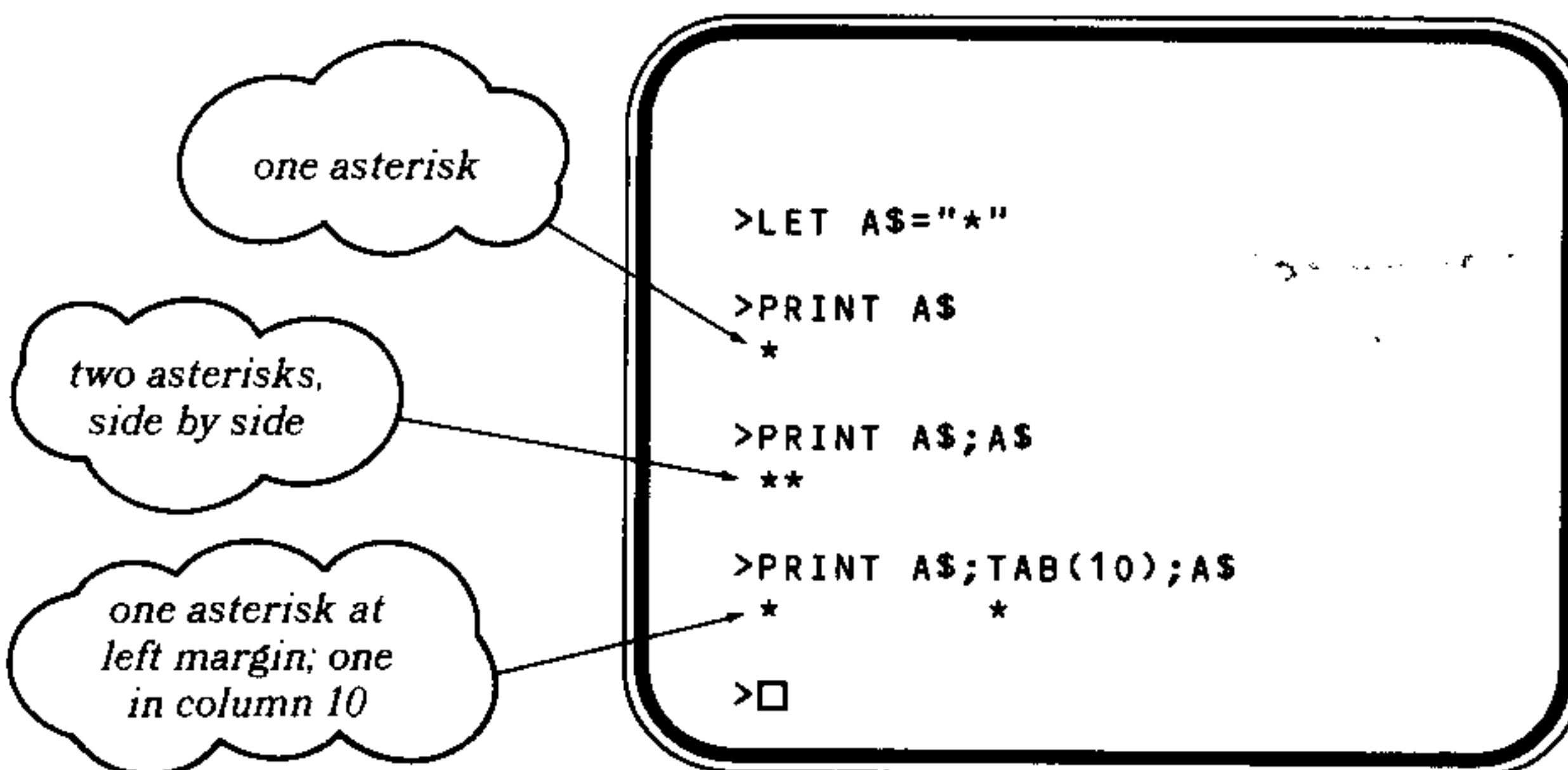
The two programs we'll develop next continue our exploration of computer graphics by showing how to construct patterns out of standard characters. The statements and functions used in the program are elements of BASIC you already know; however, you may see some new applications of these features.

### Rectangles and Squares

The first program allows you to place a rectangle or square of standard characters on the screen. Instead of using CALL HCHAR or CALL VCHAR and identifying the character by its *character code*, we'll assign a character to a string variable from the keyboard.

Try these examples in the Immediate Mode:

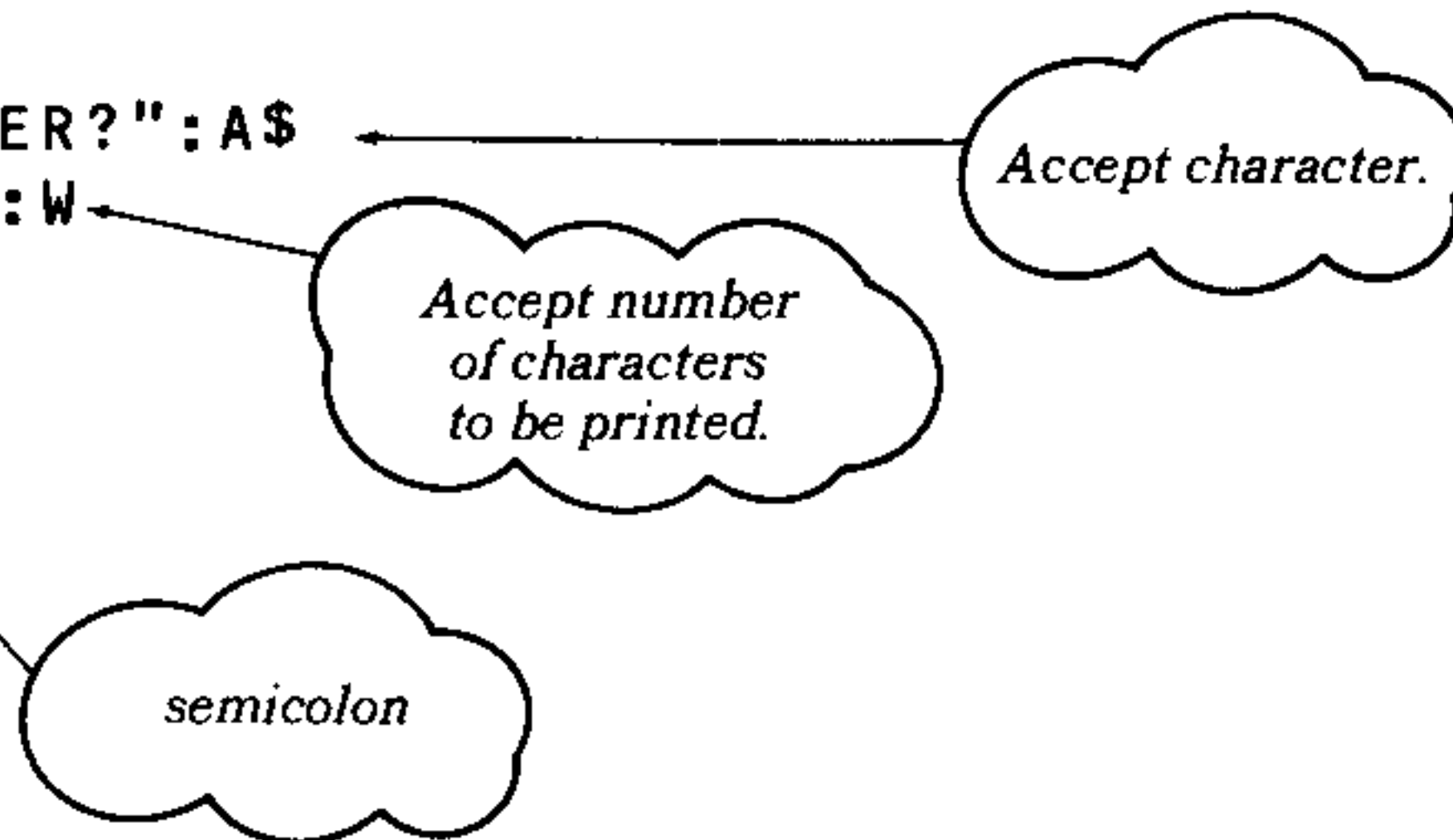
```
LET A$="*"
PRINT A$
PRINT A$;A$
PRINT A$;TAB(10);A$
```



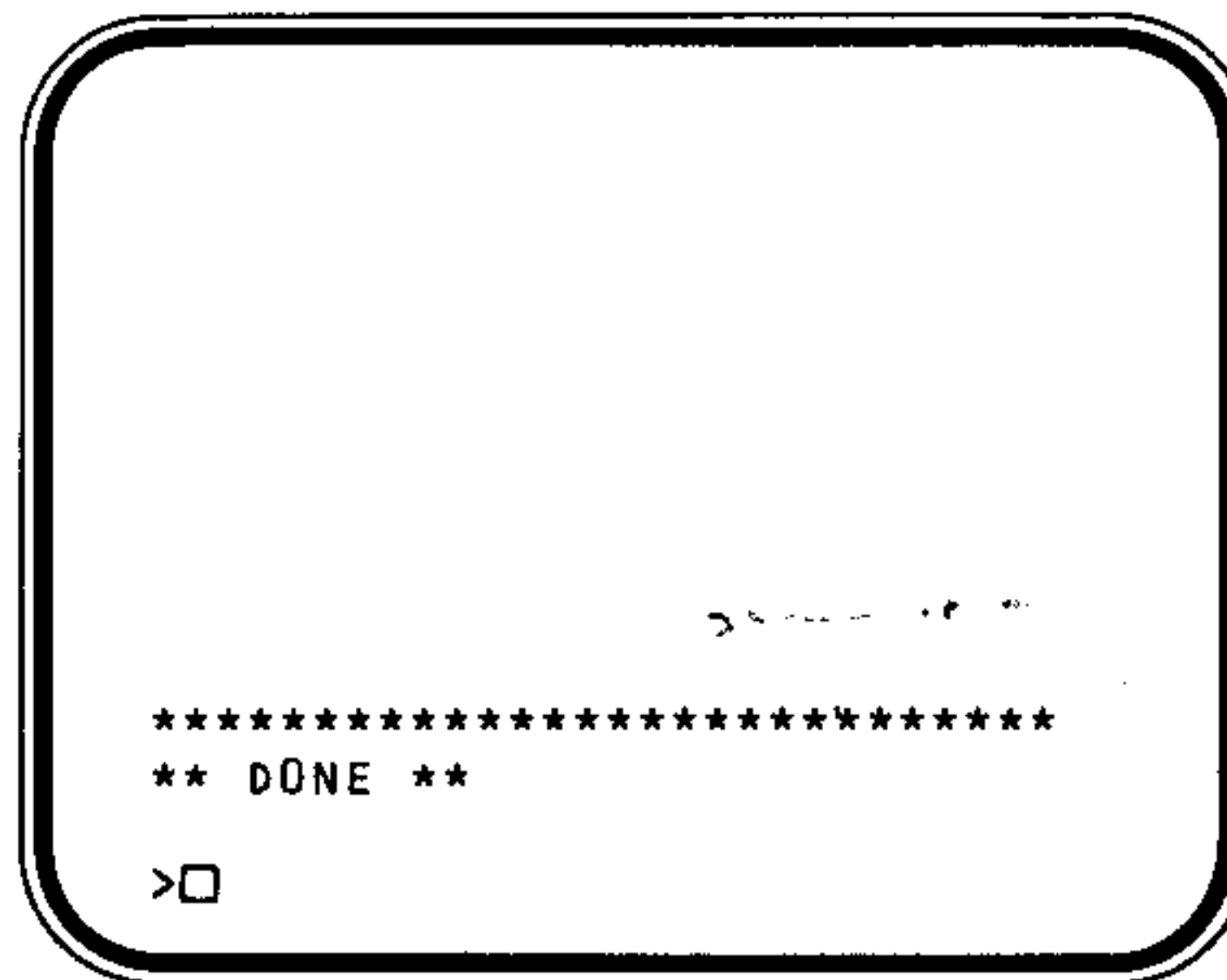
Try a few more Immediate Mode experiments on your own. For example, what would happen if you redefine A\$ as "\*\*\*" or as "(")? Try it and see what results! (If you need to review the TAB function and the print separators, see Chapter 3, page 57.)

This method is convenient if you want to print only a short line of characters. But what if you want to print a long line or vary the line length or character the program prints? INPUT statements and a FOR-NEXT loop will solve the problem. Type NEW; then enter this program:

```
20 INPUT "CHARACTER?":A$
40 INPUT "WIDTH?":W
60 CALL CLEAR
80 FOR X=1 TO W
100 PRINT A$;
120 NEXT X
140 END
```



When you run the program, you'll first be asked to input the character you want to use. Just type the character and press **ENTER**. Then you'll be asked for the "width" or the number of characters in the line you want to print. Type in the number and press **ENTER** to continue the program. Let's say that you entered \* as the character and 28 as the width. The screen will look something like this:



(Note that the semicolon in line 100 causes the characters to be printed in an unbroken row.)

Run the program a few times, entering different characters and lengths. Then let's try adding some program lines that will allow us to make rectangles and squares of characters.

# 5

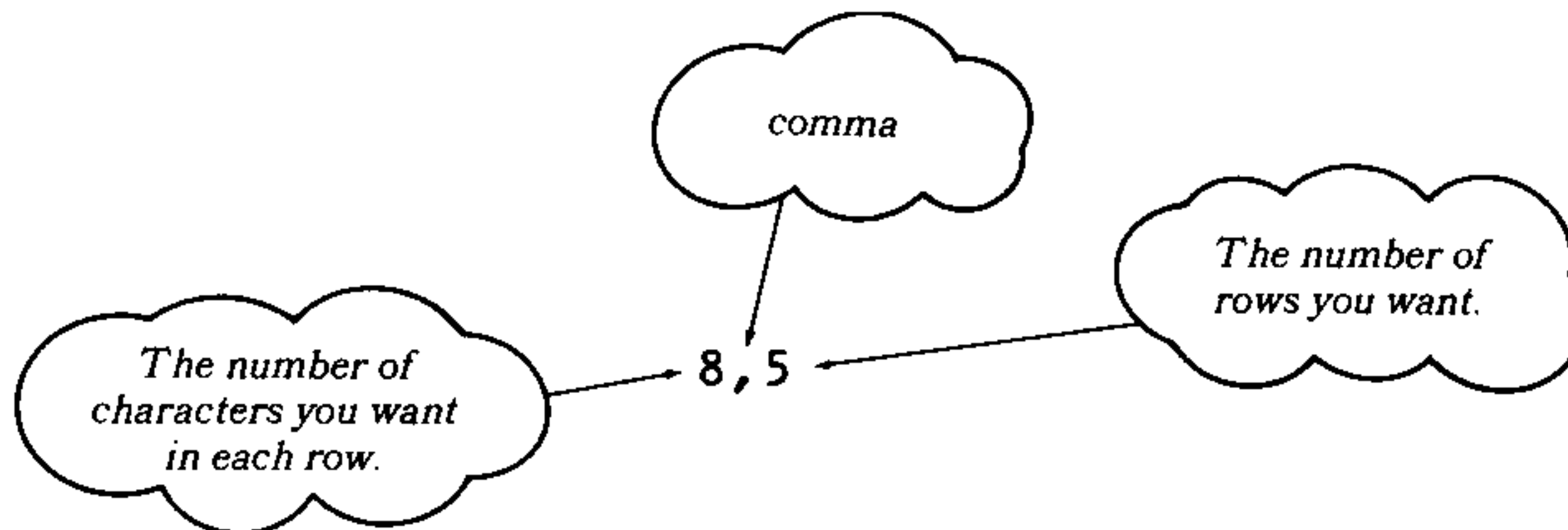
Enter these new lines:

```
40 INPUT "SIZE(WIDTH,HEIGHT)":W,H
70 FOR Y=1 TO H
130 PRINT
135 NEXT Y
140 GOTO 40
```

Replaces old line 40.

Replaces old line 140.

There are a couple of items that need to be explained about these lines. First, notice in line 40 that we are using one INPUT statement to assign values to *two* variables! When you input the width and height, you'll need to use this form:



Second, lines 70 and 135 set up a loop on the variable Y. Your original "X loop" is now nested inside the "Y loop."

Finally, line 130 prints an "empty" line. This line is needed to clear away the semicolon (;) in line 100 so that a new row will begin the next time the program loops through the "Y loop." (As you've seen already, the semicolon causes the characters to be printed on the same line throughout the loop on X.)

Before we list the program to see the changes, let's add a few more lines. We can use IF-THEN statements to "build in" some tests:

```
25 IF A$="XX" THEN 150
45 IF W+H=0 THEN 20
150 END
```

If character is XX,  
stop the program.

If both width  
and height are 0,  
ask for new character.

Here's what these tests provide. Line 25 gives you a handy way to stop the program by pressing the **X** key twice and then pressing **ENTER** when you're asked for a character input. If you want to experiment with a different character, all you have to do is to enter **0,0** as size inputs. The test in line 45 then sends you back to line 20 to input a new character.

Now clear the screen and list the program:

```

LIST
20 INPUT "CHARACTER?":A$
25 IF A$="XX" THEN 150
40 INPUT "SIZE(WIDTH,HEIGHT)
   ":W,H
45 IF W+H=0 THEN 20
60 CALL CLEAR
70 FOR Y=1 TO H
80 FOR X=1 TO W
100 PRINT A$;
120 NEXT X
130 PRINT
135 NEXT Y
140 GOTO 40
150 END
    
```

*If A\$=XX, stop!*

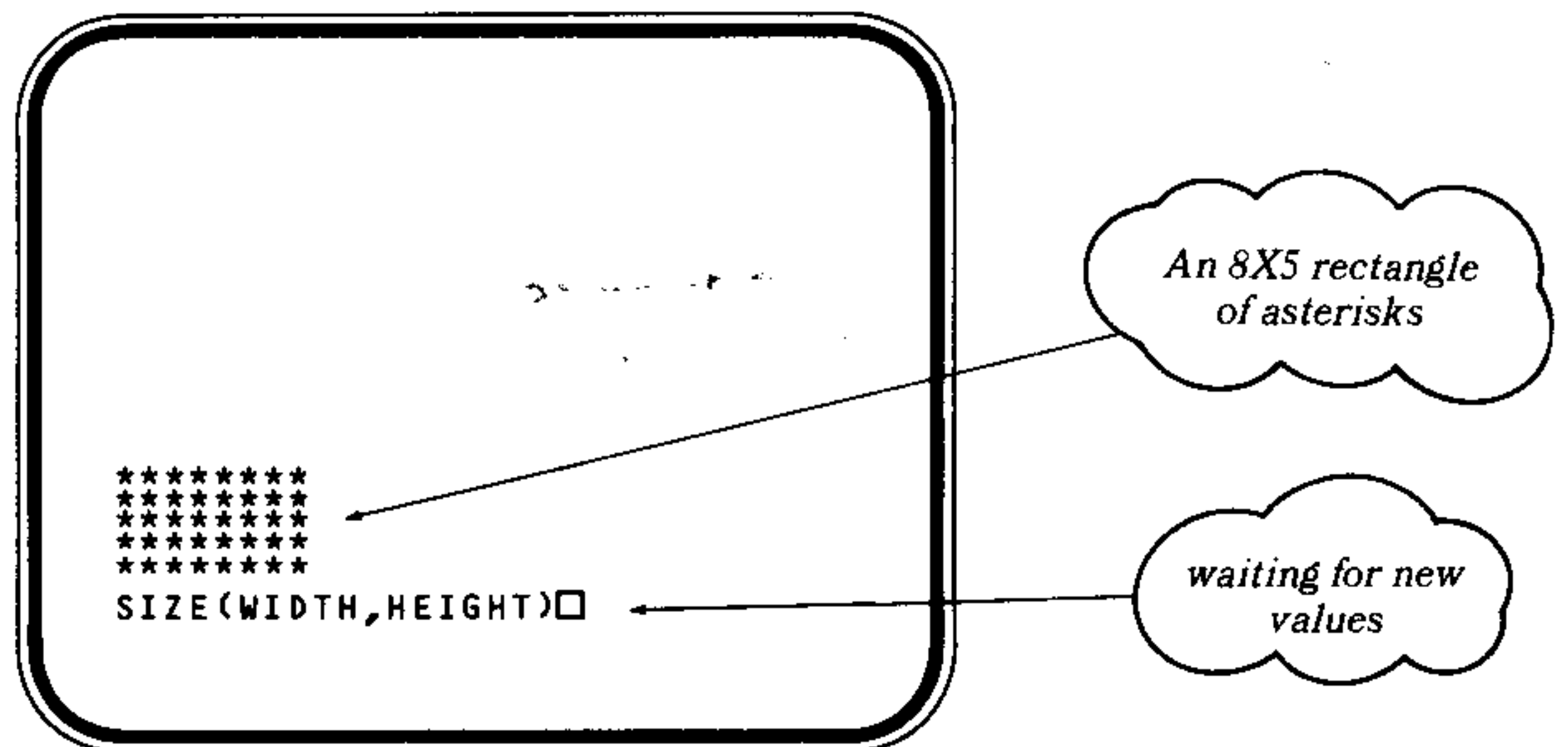
*If W and H both=0, get new character input.*

*Print a line of characters.*

*Start a new line.*

*Return for new size inputs.*

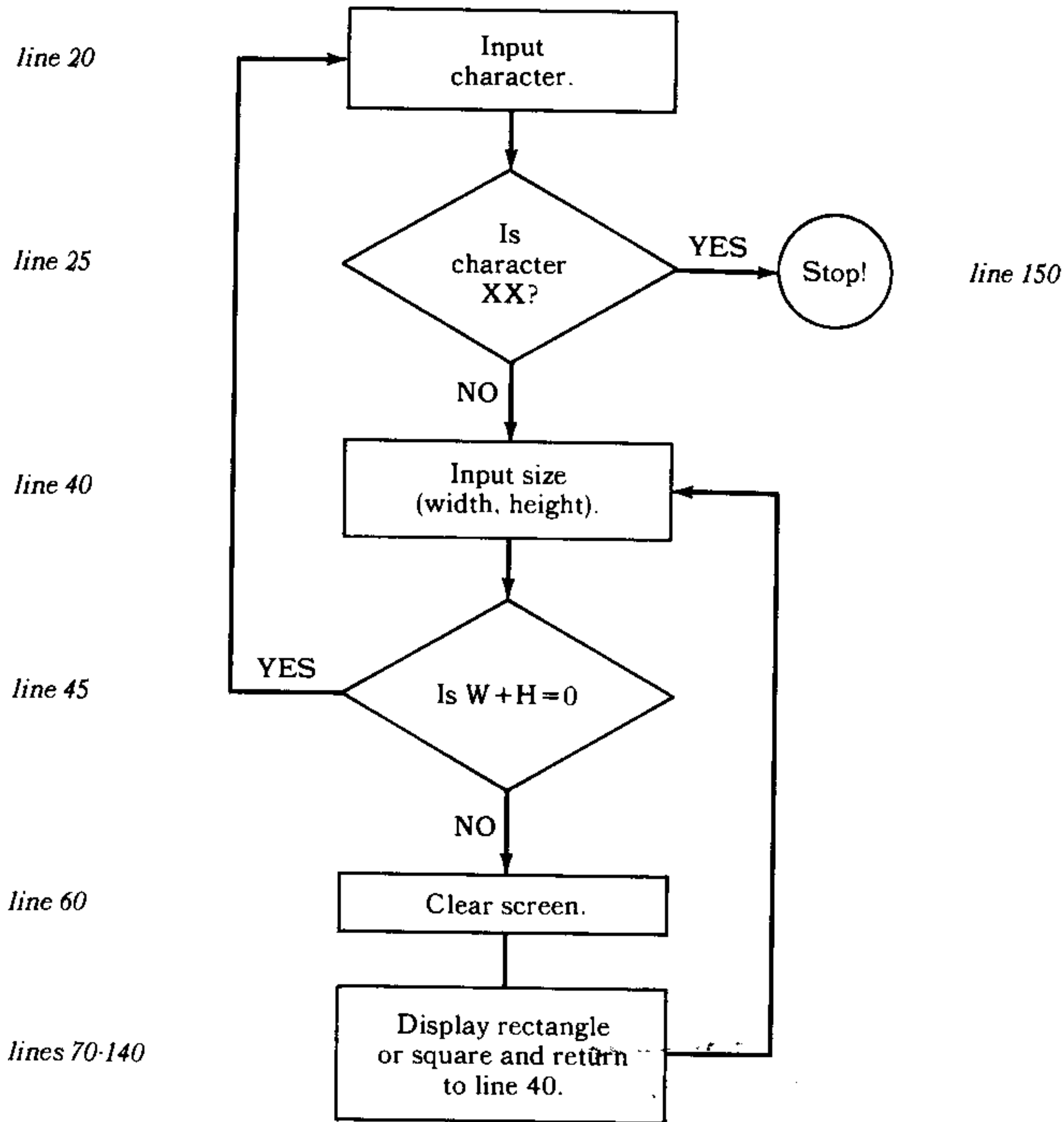
Clear the screen again and run the program. For this example, enter **\*** when the program asks CHARACTER? Then enter **8,5** when you're asked for width and height:



Next, enter the same value for both width and height, such as 8,8 or 5,5. With these inputs the program will create a square, rather than a rectangle.

# 5

Perhaps a flowchart will help to describe how the program works. The following diagram doesn't show the whole program in detail; it covers only the parts that relate to program control by input values.



## Experiment!

Experiment with the program. Try entering the control values (character input = XX, width and height both = zero) to see how the program reacts. Vary the width and height so that the display fills the screen or makes only tall thin bars and wide flat strips. What happens if you enter a width greater than 28 or a height greater than 24? (Try it and see what happens.) Can you add color to this program?

*"Holes"*

Let's expand the Rectangles and Squares program one more time. These new lines will create rectangles or squares with a random sprinkling of "holes" (blank spaces) in the display field. Enter the following lines:

```
15 RANDOMIZE
85 IF INT(2*RND)=0 THEN 100
90 PRINT " ";
95 GOTO 120
```

*INT(2\*RND)  
produces a 0 or a 1.*

*One space, enclosed in  
quotation marks and  
followed by a semicolon.*

Now clear the screen and list the changed program, so that we can discuss the effect of these additions.

```
LIST
15 RANDOMIZE
20 INPUT "CHARACTER?":A$
25 IF A$="XX" THEN 150
40 INPUT "SIZE(WIDTH,HEIGHT)
":W,H
45 IF W+H=0 THEN 20
60 CALL CLEAR
70 FOR Y=1 TO H
80 FOR X=1 TO W
85 IF INT(2*RND)=0 THEN 100
90 PRINT " ";
95 GOTO 120
100 PRINT A$;
120 NEXT X
130 PRINT
135 NEXT Y
140 GOTO 40
150 END
```

*If true, go to  
line 100 and PRINT  
the character. If false  
PRINT a blank space  
(line 90).*

*Skip line 100.*

The test with the RND function in line 85 causes a character to be printed whenever INT(2\*RND) is equal to 0, a space when INT(2\*RND) is equal to 1. Thus, approximately half of the time the program prints a character and half of the time a space. Run the program now, and observe the kind of pattern that emerges.

**Experiment!**

You'll be able to see the "holes" better by making the character into a color block. Add these lines to the program:

```
10 INPUT "COLOR?":C
45 IF W+H=0 THEN 10
50 CALL COLOR(3,C,C)
```

Now, when the program asks, COLOR?, type and enter a color code from 1 to 16. Notice that, to see the character in color, you'll have to enter a character input from set #3 (0,1,2,3,4,5,6, and 7 are the characters in this set).



# 5

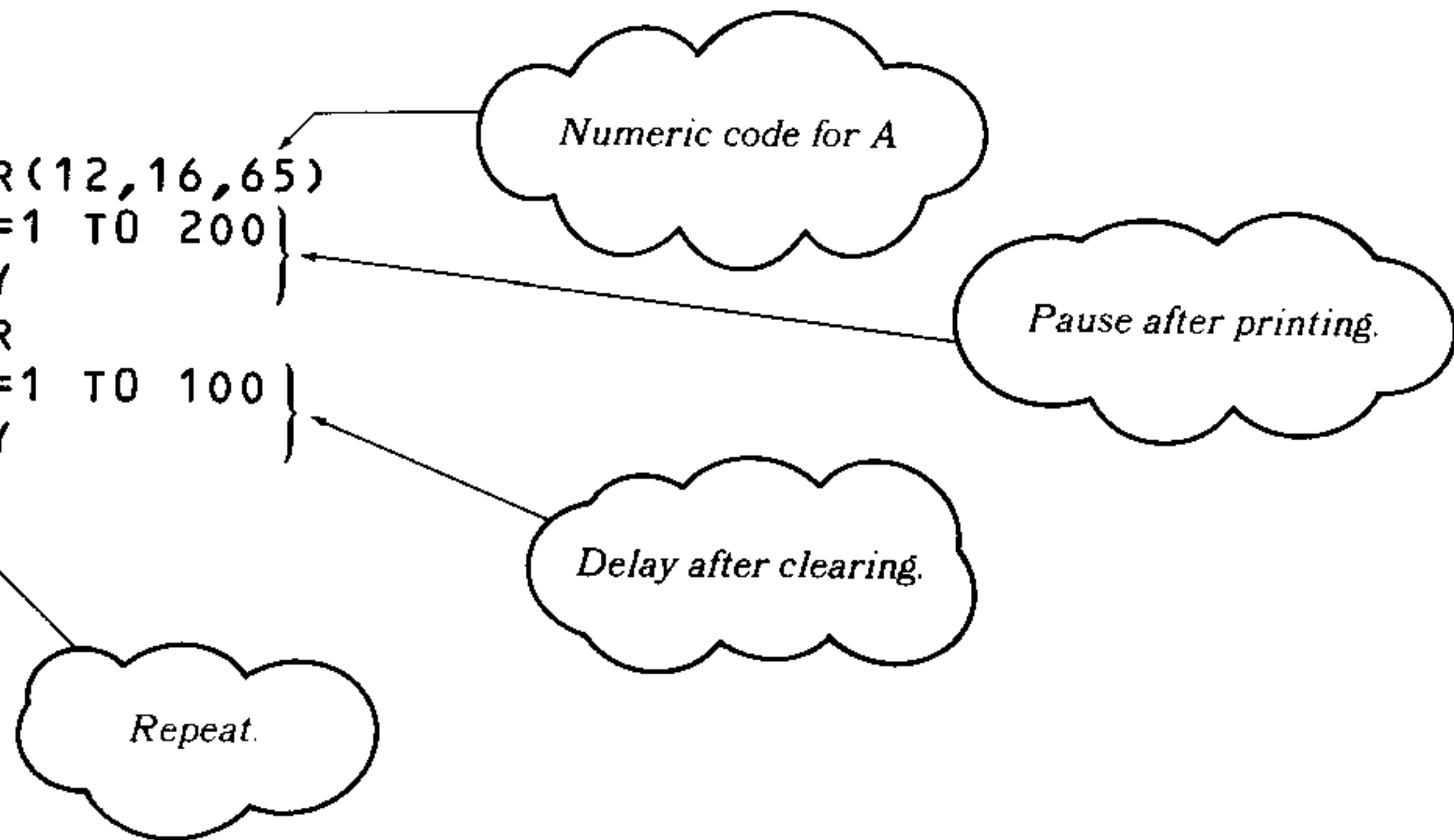
## Animation

Animation is the illusion of movement. In order to achieve this illusion in your graphics programs, it's necessary to keep changing your character or sets of characters. The following programs demonstrate some of the techniques used to create flashing and moving graphics on the screen.

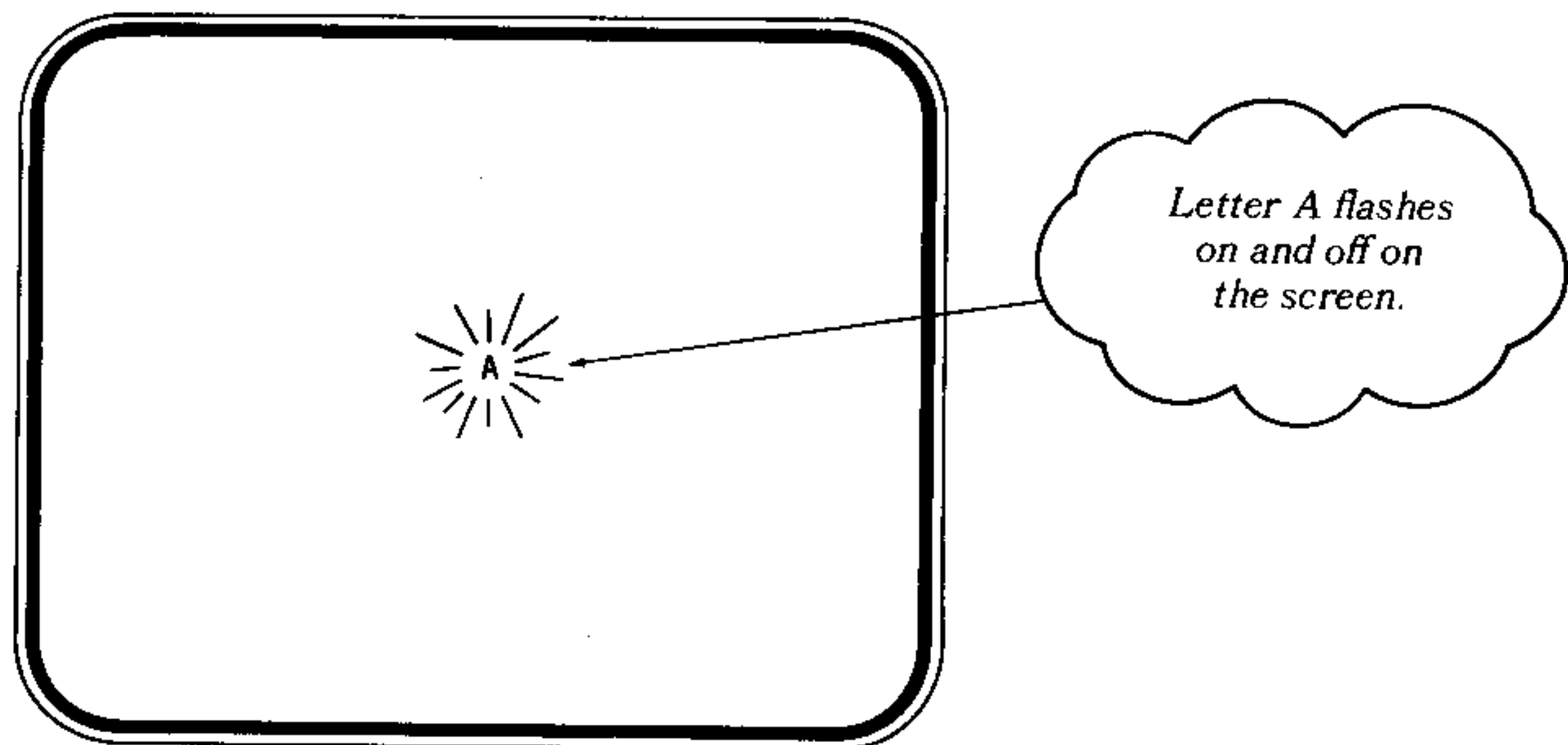
### Flashing Letters

One way to create a flashing graphic is to print a character (or set of characters), delay the program, clear the screen, delay the program again, and then repeat the process. The clearing of the screen and the delays have the effect of turning the character "on and off," making it appear to flash. Let's try a program that flashes the letter A in the center of the screen.

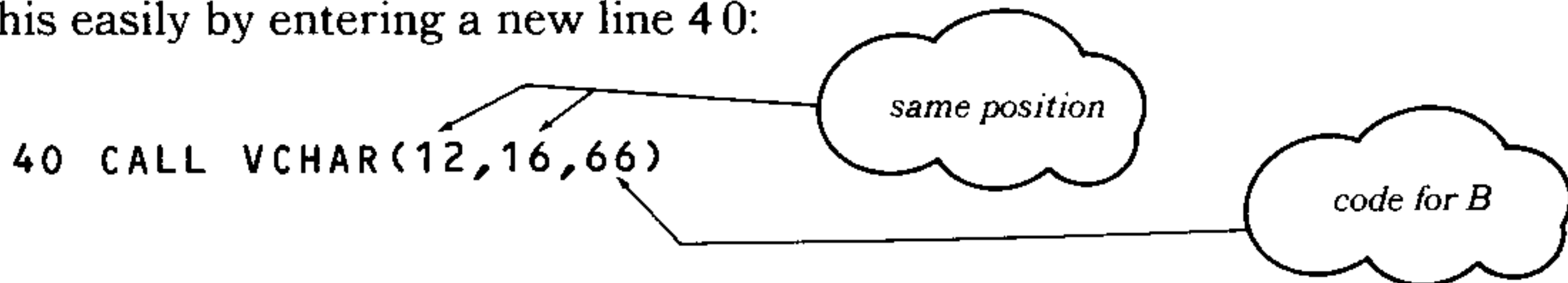
```
NEW
10 CALL VCHAR(12,16,65)
20 FOR DELAY=1 TO 200
30 NEXT DELAY
40 CALL CLEAR
50 FOR DELAY=1 TO 100
60 NEXT DELAY
70 GOTO 10
```



Now clear the screen and run the program.



Another way to simulate flashing is to replace one character with another in the same spot on the screen. Let's revise our program so that it alternately flashes A and B. We can do this easily by entering a new line 40:



Since we're replacing A with B, we don't have to clear the screen between printing the characters. However, we may want to add a CALL CLEAR at the beginning of the program. So enter this line:

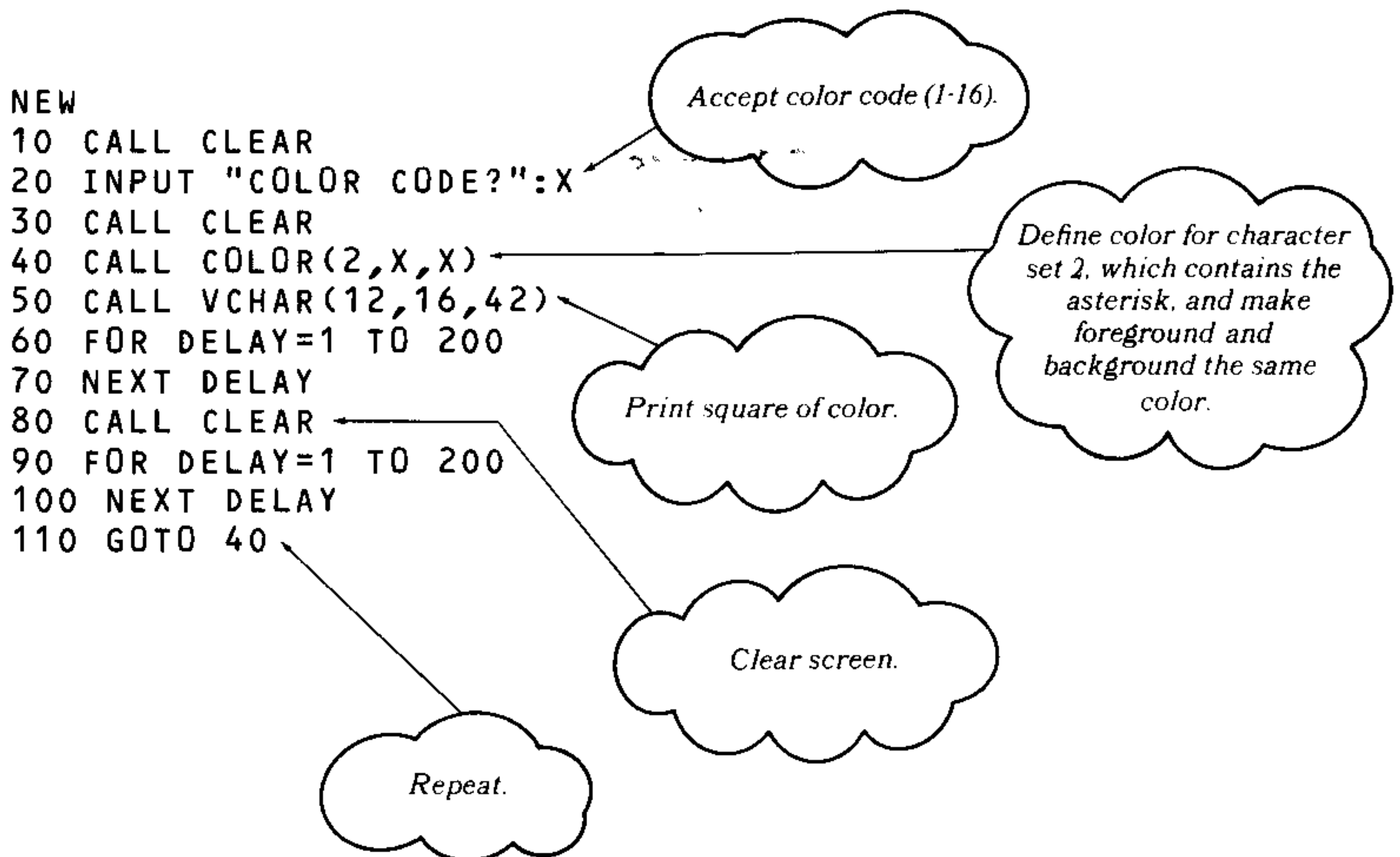
```
5 CALL CLEAR
```

and run the revised program. Do A and B appear to flash alternately on the screen? (You may want to increase the time delay in line 50, so that A and B will each stay on the screen the same length of time.)

From flashing characters to flashing color squares is an easy step, so we'll examine next a program that places a flashing color square on the screen.

### Flashing Color Squares

With this program we want to create a color square that flashes on the screen. We'll write the program so that we can input the color we want, and we'll use character 42 (the asterisk, in character set 2) to make our square. Here's the program:



# 5

Now run the program. First, it asks

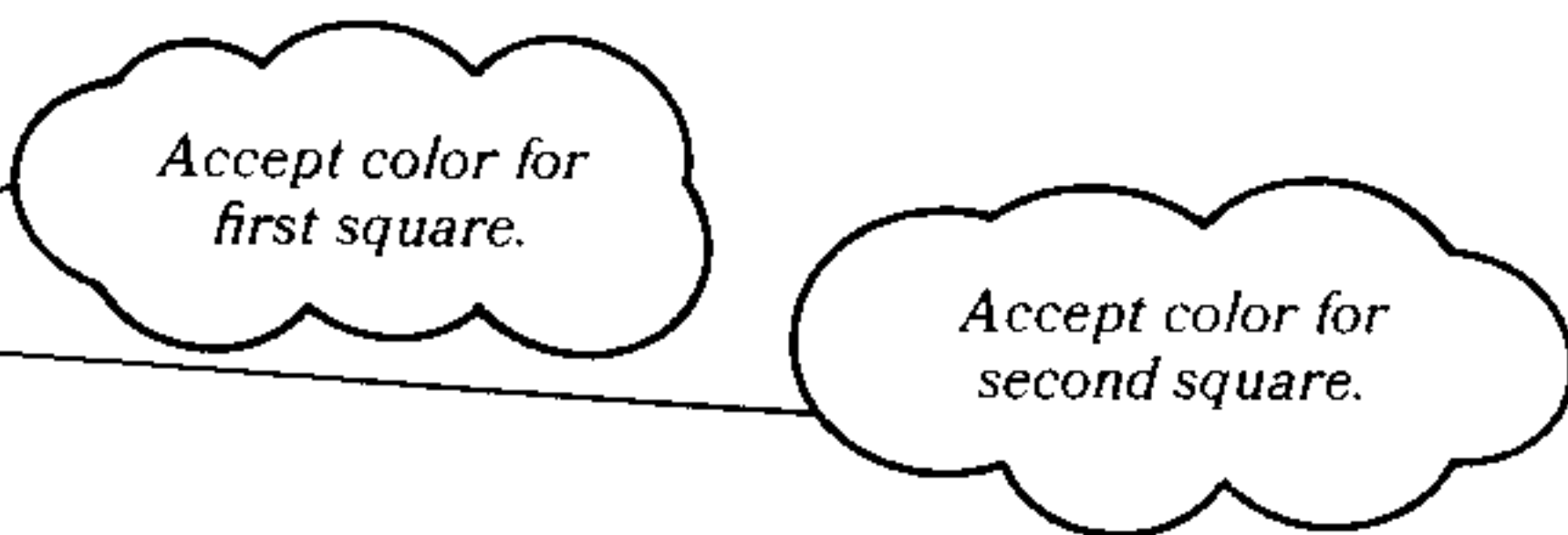
```
COLOR CODE?
```

and waits for you to input a valid color code. The codes are 1 through 16; remember, however, that code 1 is transparent and code 4 is the normal screen color in the RUN Mode. Squares of these colors will not show up on the screen.

When you type in a color code and press **ENTER**, you'll see the square flashing near the center of the screen.

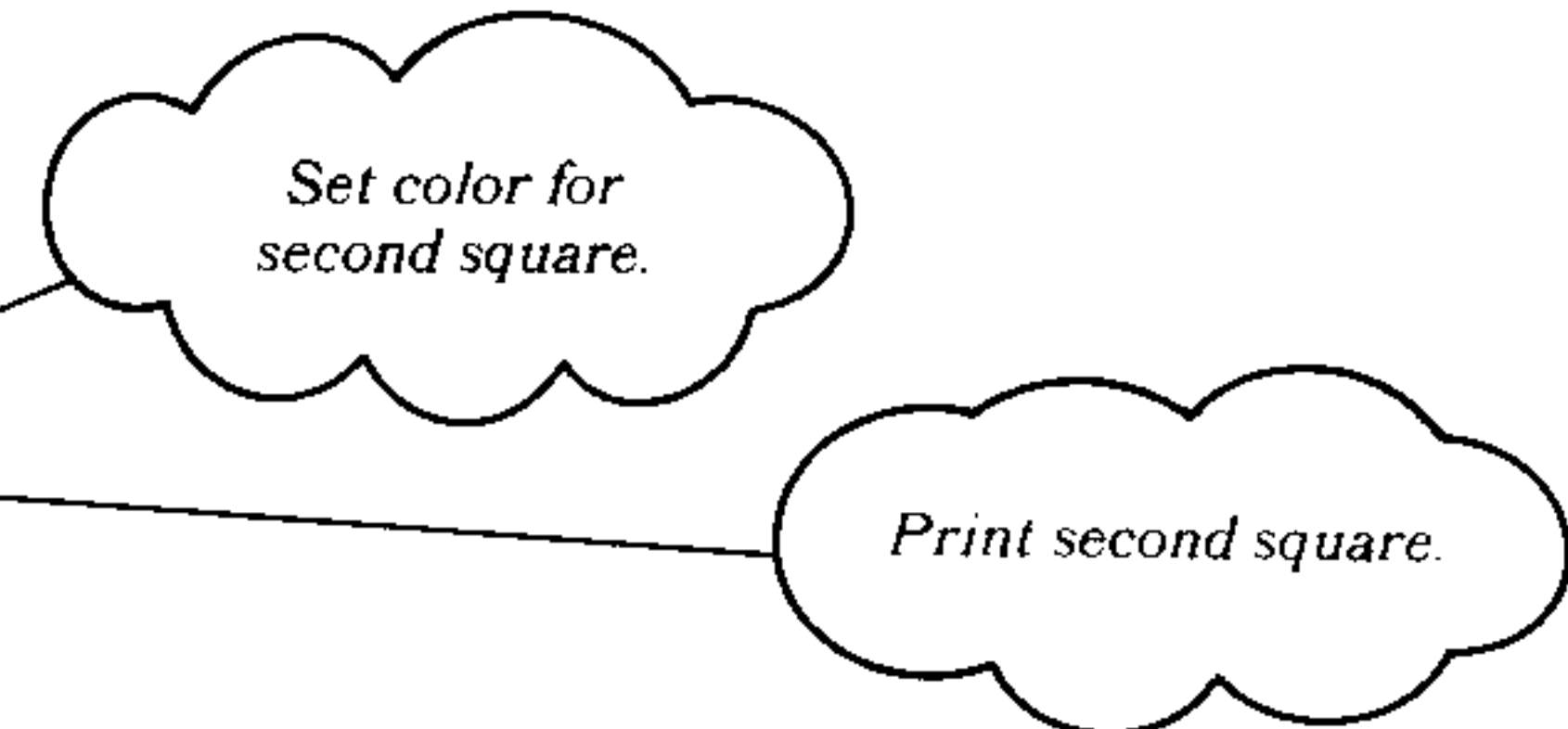
Next, let's change the program to create *two* color squares that alternately flash on the screen. To do so, we'll need to input two color codes. So enter these lines first:

```
20 INPUT "COLOR1?":X
25 INPUT "COLOR2?":Y
```



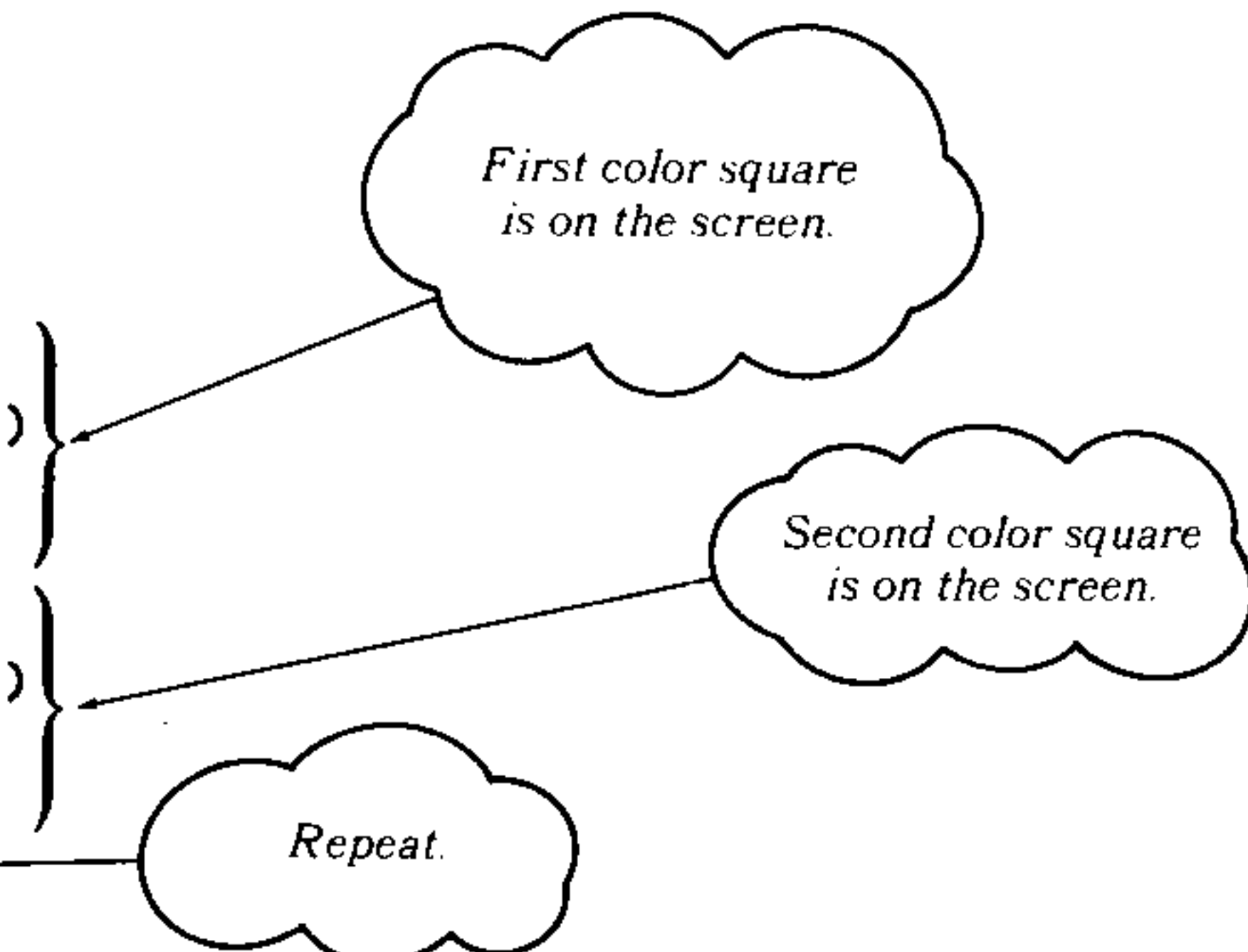
Now we'll replace our original line 80 with two new lines, to set the color and display the second square:

```
80 CALL COLOR(2,Y,Y)
85 CALL VCHAR(12,16,42)
```



Let's review these changes by listing the program. Clear the screen; then type **LIST** and press **ENTER**:

```
LIST
10 CALL CLEAR
20 INPUT "COLOR1?":X
25 INPUT "COLOR2?":Y
30 CALL CLEAR
40 CALL COLOR(2,X,X)
50 CALL VCHAR(12,16,42)
60 FOR DELAY=1 TO 200
70 NEXT DELAY
80 CALL COLOR(2,Y,Y)
85 CALL VCHAR(12,16,42)
90 FOR DELAY=1 TO 200
100 NEXT DELAY
110 GOTO 40
```



Select your two colors and run the program, typing in the color codes as the program asks for them. The two color squares will alternately flash on the screen.

Experiment with several color combinations to find those that give a good contrast. Here are a few examples to try:

<i>Color 1</i>	<i>Color 2</i>
6	5
11	14
14	16
9	11

### *Moving Color Squares*

With just a few simple changes in the previous program, we can make the color squares move across the screen as they flash. Add these lines:

```
35 FOR I=3 TO 28
50 CALL VCHAR(12,I,42)
85 CALL VCHAR(12,I,42)
105 CALL CLEAR
110 NEXT I
120 GOTO 10
```

Now list the program to review the changes:

```
CALL CLEAR
LIST
10 CALL CLEAR
20 INPUT "COLOR1?":X
25 INPUT "COLOR2?":Y
30 CALL CLEAR
35 FOR I=3 TO 28
40 CALL COLOR(2,X,X)
50 CALL VCHAR(12,I,42)
60 FOR DELAY=1 TO 200
70 NEXT DELAY
80 CALL COLOR(2,Y,Y)
85 CALL VCHAR(12,I,42)
90 FOR DELAY=1 TO 200
100 NEXT DELAY
105 CALL CLEAR
110 NEXT I
120 GOTO 10
```

When you've checked the program for accuracy, run it. Starting at column 3, the squares flash and travel across the screen, ending at column 28. Then the screen clears, and the program asks you for new color inputs.

### ***Experiment!***

If you want to speed up the flashing, shorten the time delay loops in lines 60 and 90. For a challenge, you might like to make the program flash *three* color squares! How would you do it?

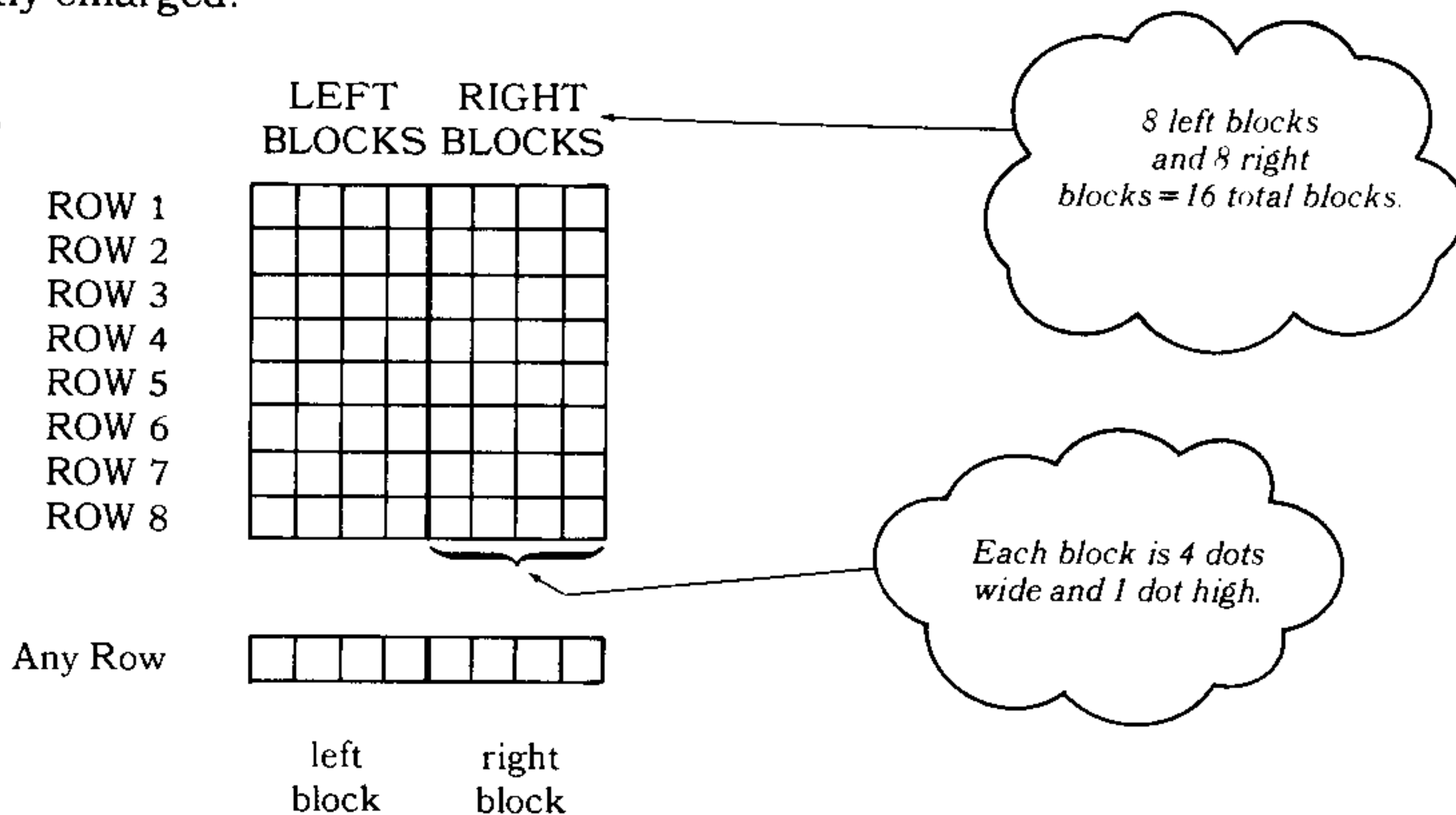
By this time you've seen several examples of the kind of graphics you can create with the standard characters of your computer. Next we'll show you how to develop characters of your own.

# 5

## The CALL CHAR Statement

The CALL CHAR statement gives you the capability of creating your own screen characters. In our first program we'll *redefine* some of the standard characters. Before we redefine a character, however, we must first look at the way a character is represented on the screen.

Each printing position on the screen is made up of sixty-four tiny dots. The dots are arranged in eight rows of eight dots each. Each row is partitioned into two blocks of four dots each. The diagrams below show how an 8-by-8 grid of dots would look if it were greatly enlarged.



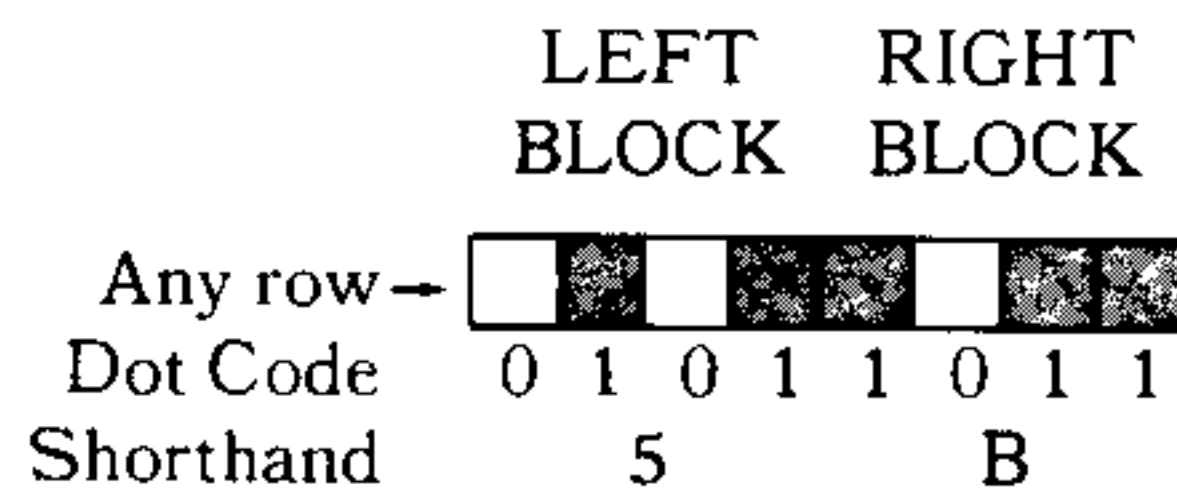
A character on the screen, either a standard character or one that you invent, is formed by dots within the 8-by-8 grid. By turning some dots "on" and leaving others "off," a character is created. Leaving all the dots "off" creates the space character (character code 32), for example. Turning all the dots "on" produces a solid spot on the screen.



All the standard characters are automatically set so that they turn on the appropriate dots to produce the images you have seen. To create a new character, we must tell the computer which dots to turn on or leave off in each of the 16 blocks within the printing region that contains the character. In your computer a shorthand system is used to specify which dots are on or off within a particular block. The table that follows contains all the possible on/off conditions for the dots within a given block and the shorthand notation for each condition.

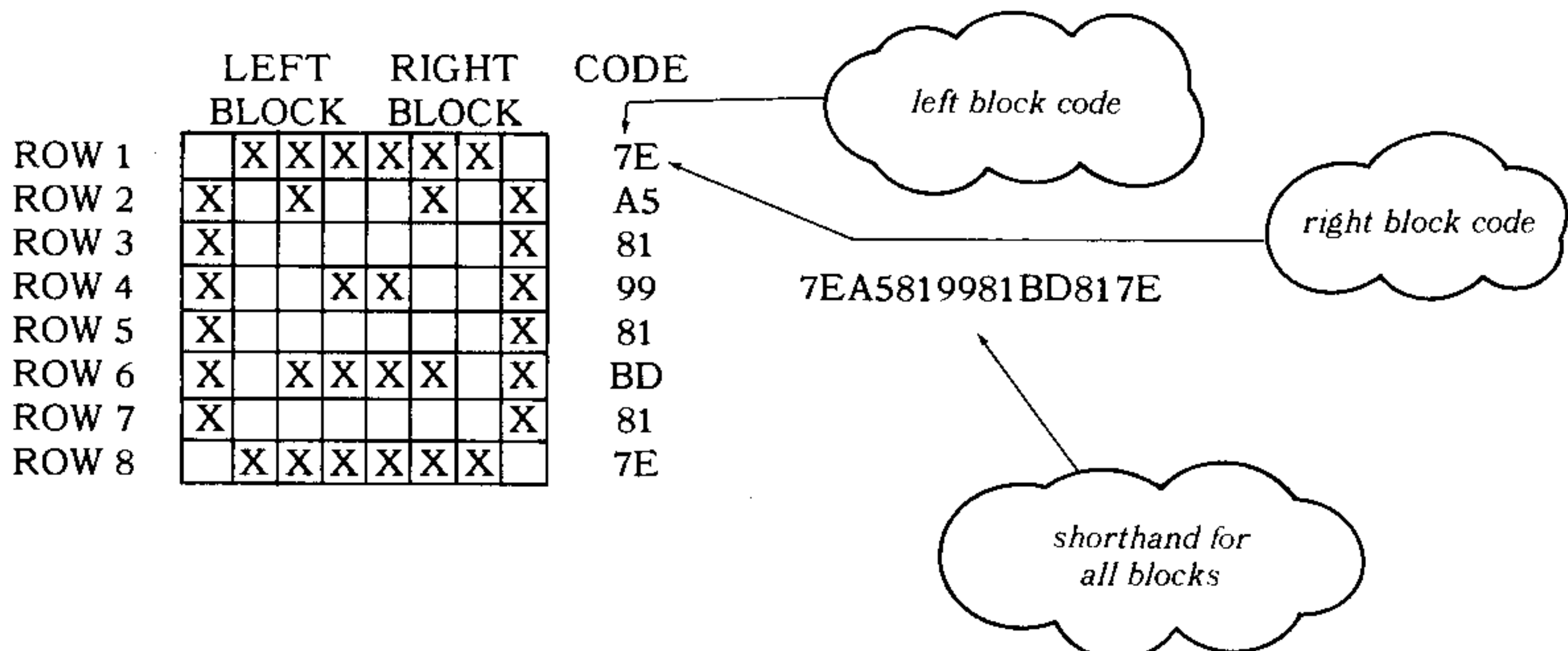
BLOCKS	DOT CODE (0=off; 1=on)	SHORT- HAND CODE
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

Let's take a look at one row (two blocks) to see how the "shorthand code" works.



The shorthand code for the row, then, is 5B.

The shorthand codes for an entire grid can be determined block by block, just by converting the on/off conditions of each row. The following example provides a translation of an entire grid into the shorthand code.



# 5

Therefore, if we want to "define" a character shaped as the X's on the grid indicate, we enter all the shorthand codes of the blocks as a single "string":

"7EA5819981BD817E"

In the shorthand code, then, one number or letter represents a whole block (4 dots) on the grid. Two letters and/or numbers represent a whole row.

Based on the table, if all the dots in all the blocks were to be turned on, the shorthand code for this condition would be:

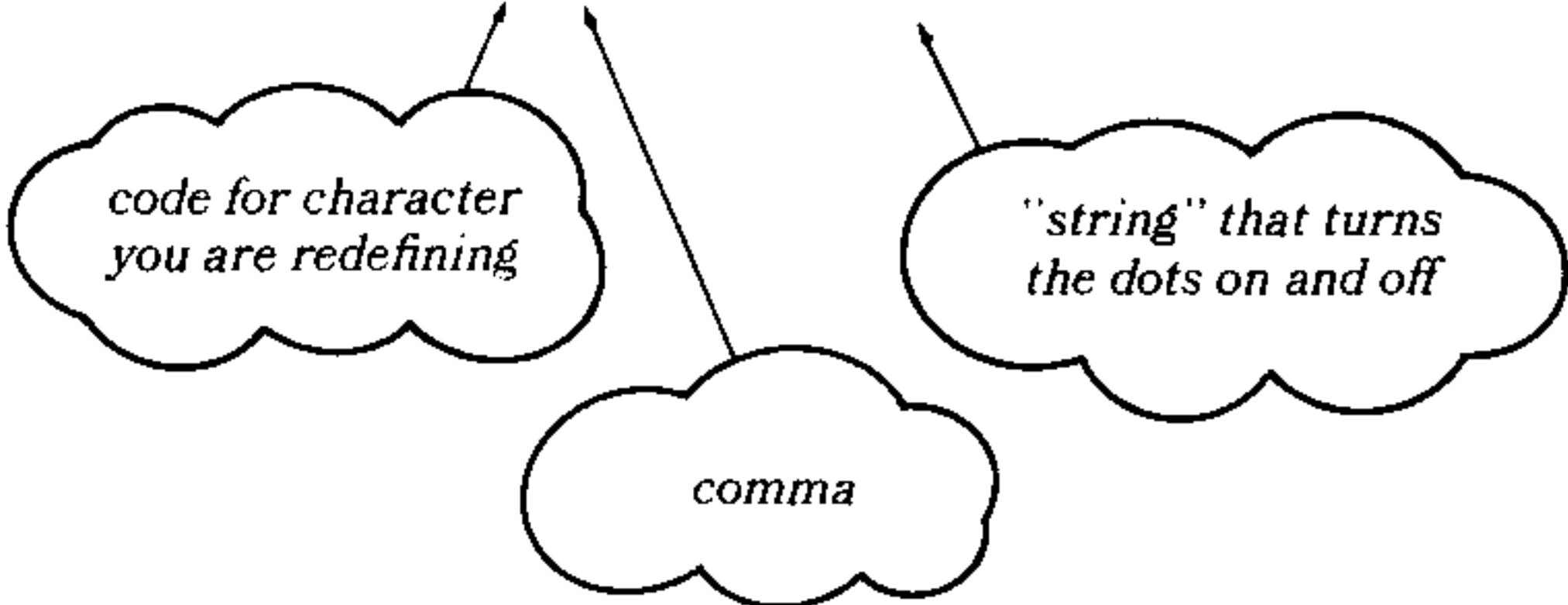
"FFFFFFFFFFFFFFFF"

One F for each block

This code may seem long, since it represents all 16 blocks within the grid. But it is still shorter than trying to write down all 64 separate conditions dot by dot.

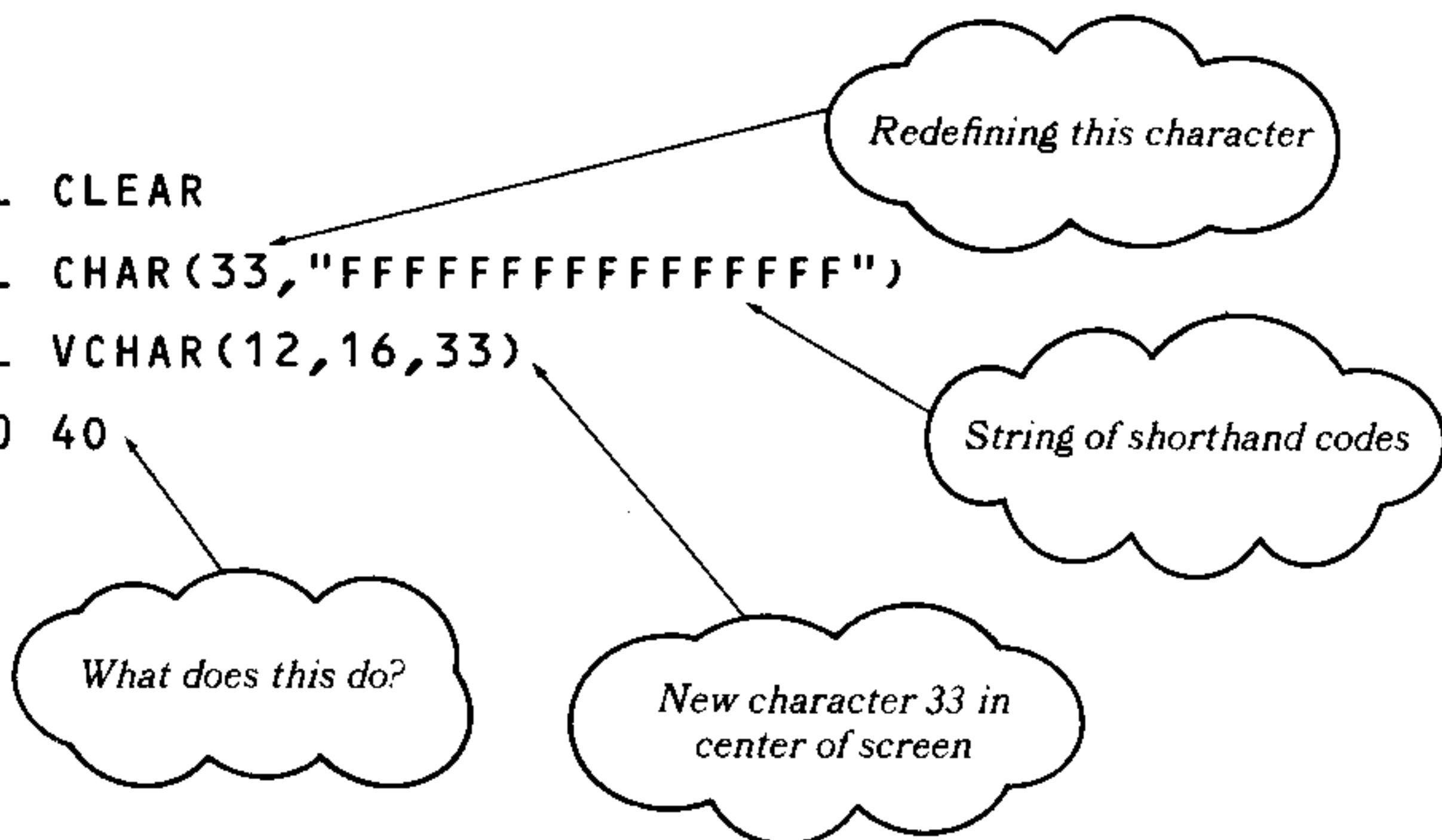
Once you've decided which dots you want on and off and worked out the code, you're ready to use the CALL CHAR statement. It looks like this:

```
CALL CHAR(33,"FFFFFFFFFFFFFFFF")
```

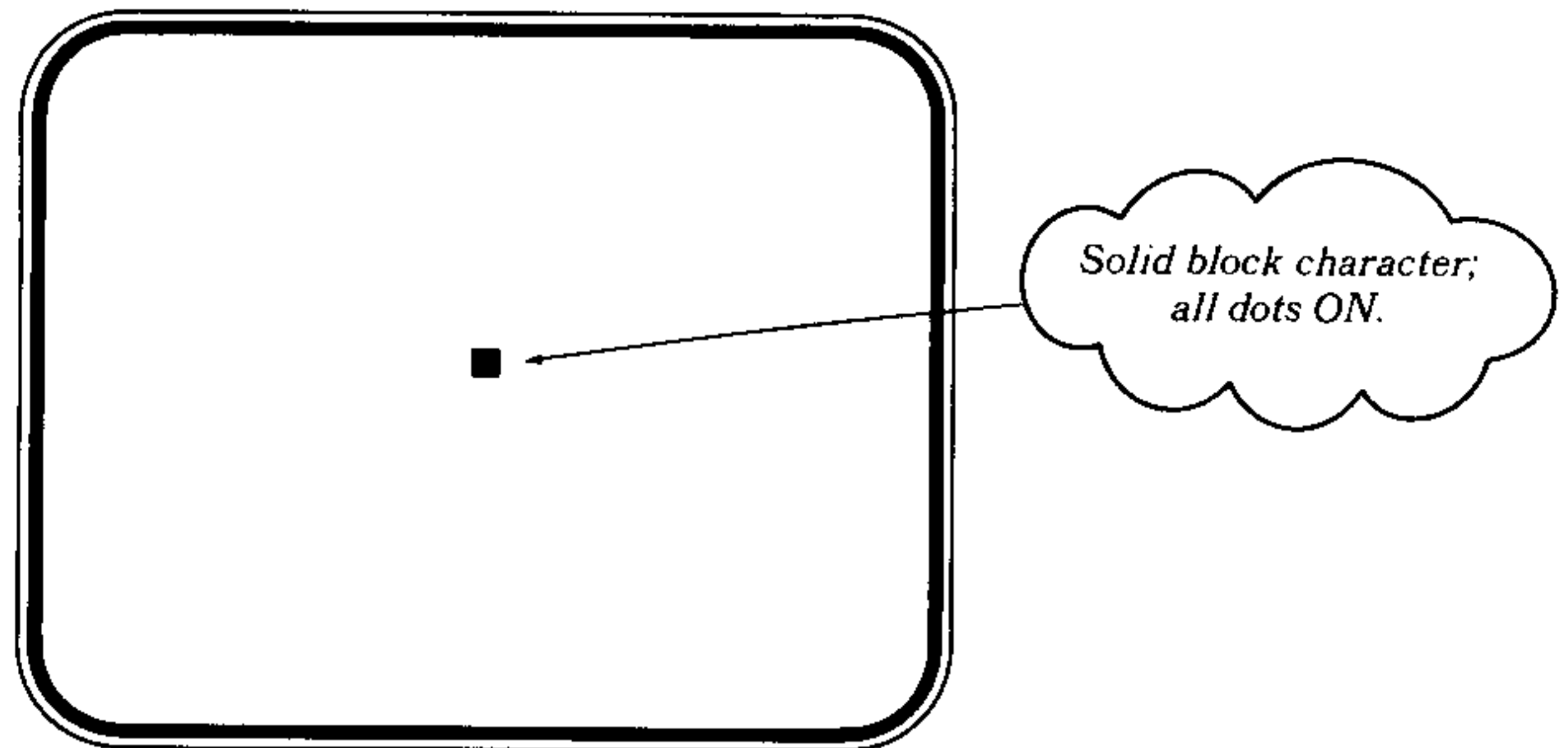


Let's try a simple program that redefines character code 33(!) as a character with all the dots turned on. The new character is then printed in the center of the screen, giving you a chance to see exactly how big one of the individual print areas really is. Enter these lines:

```
NEW  
10 CALL CLEAR  
20 CALL CHAR(33,"FFFFFFFFFFFFFFFF")  
30 CALL VCHAR(12,16,33)  
40 GOTO 40
```



Run the program and observe your newly defined character on the screen!

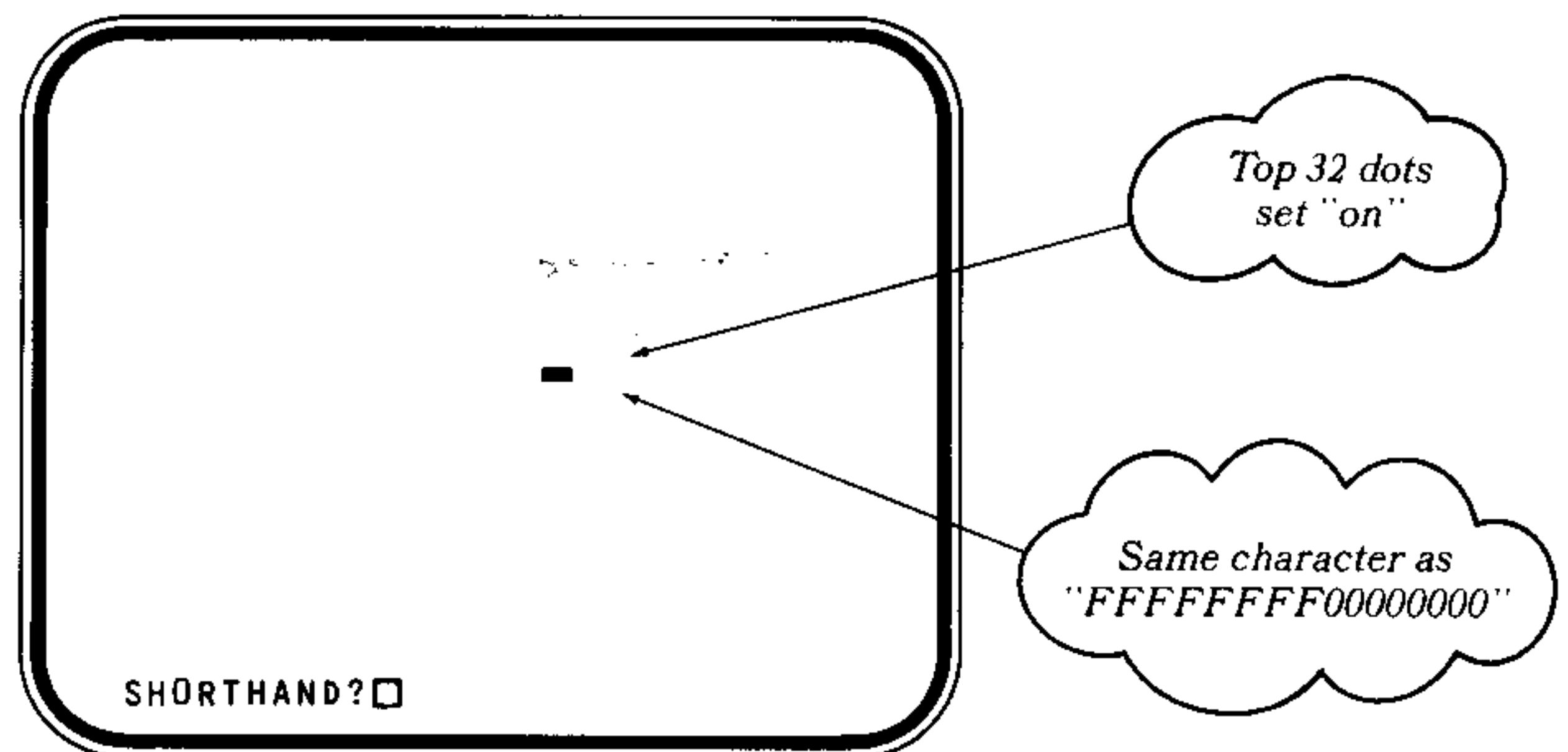


So that you can experiment with other shorthand codes, let's edit the program. Type these new lines:

```
5 INPUT "SHORTHAND?":A$
20 CALL CHAR(33,A$)
40 GOTO 5
```

This time, when you run the program, you'll be asked to input the shorthand code for the character you are defining. Try the following examples.

Enter: **FFFFFFFF**



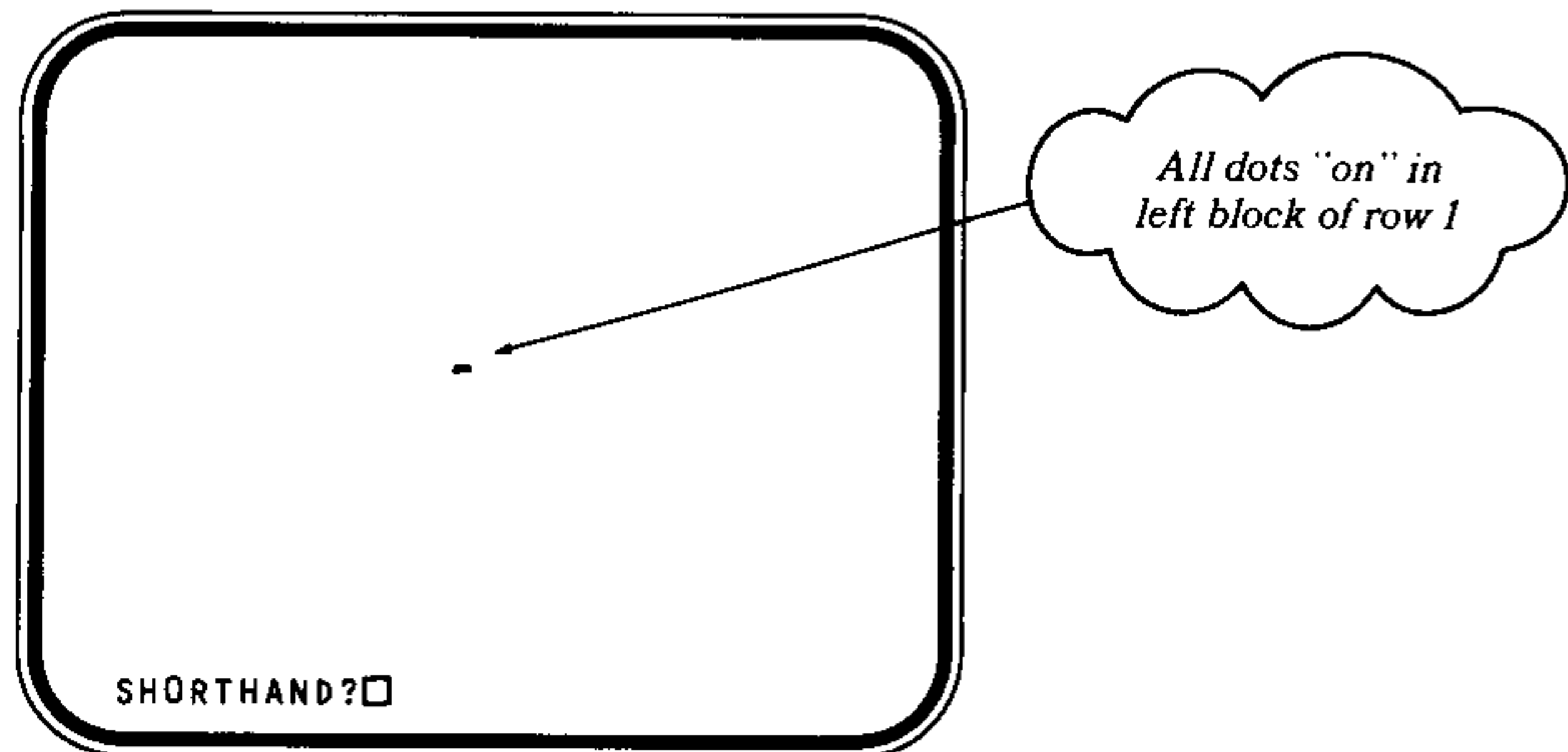
When you stop the program by pressing **CLEAR**, the character that you created changes back into the character from the standard character set. In this case, character code 33 is restored to an exclamation point (!), and that symbol appears near the center of the screen.



# 5

Entering FFFFFFFF is the same as entering FFFFFFFF00000000. That is, the CHAR routine fills out the right side of the string variable with zeros when there are less than 16 characters in the string. Knowing this fact allows you to easily examine all the shorthand codes individually. Just enter 0, 1, and so on up to F at the INPUT request.

Enter: F



Try different combinations of the shorthand codes. See if you can generate any interesting characters. Then let's revise the program again to print more than just one of our redefined characters. Enter these lines:

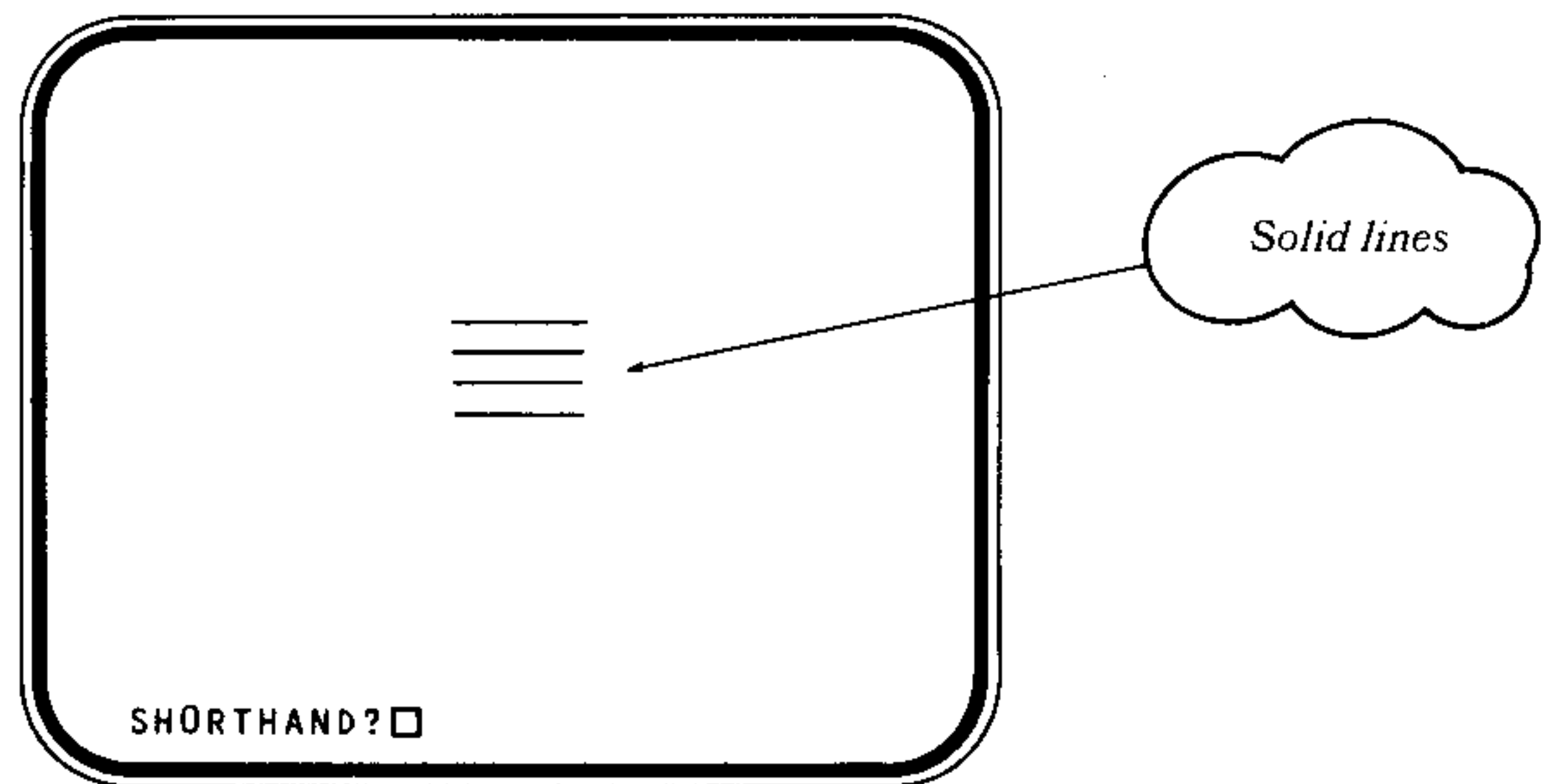
```
30 FOR I=1 TO 4
40 CALL VCHAR(12,I+13,33,4)
50 NEXT I
60 GOTO 5
```

Now list the program to see the changes:

```
LIST
5 INPUT "SHORTHAND?":A$
10 CALL CLEAR
20 CALL CHAR(33,A$)
30 FOR I=1 TO 4
40 CALL VCHAR(12,I+13,33,4)
50 NEXT I
60 GOTO 5
```

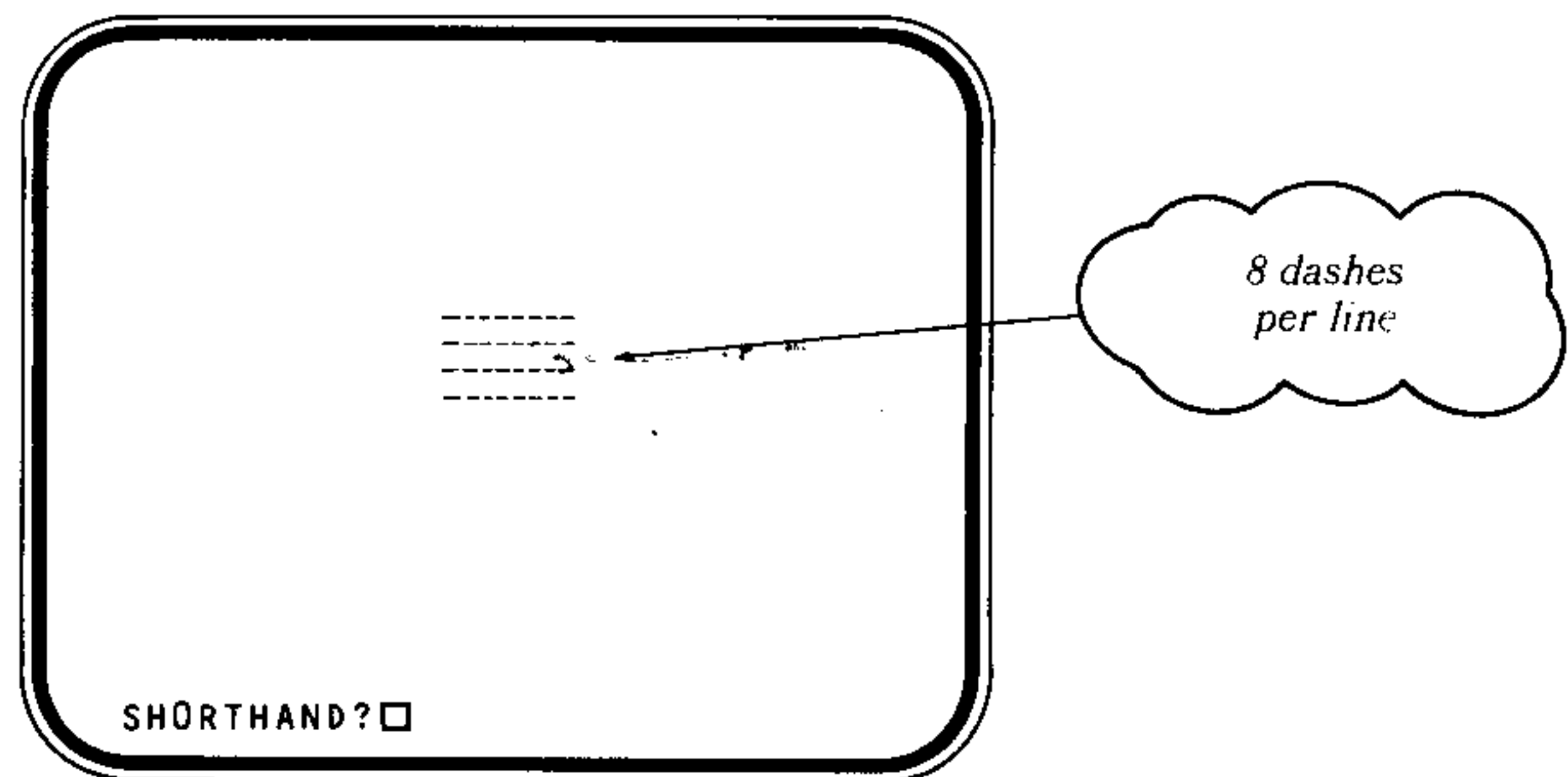
When you run this program and enter a shorthand code for a new character, that character is displayed 16 times in the center of the screen. The 16 characters appear in a square four characters wide by four characters high. Try the following:

Enter: **FF**



A single print of the character with the shorthand code FF puts something like a long dash on the screen. Printing four of these characters side by side draws a line on the screen! To get dashes across the screen you must leave a space by setting two dots in each block "off." To do this, the code is 33.

Enter: **33**



Notice that, when you stop the program, the center of the screen fills with 16 exclamation points (!).

Now enter some other codes and experiment with the program until you feel comfortable with the shorthand codes. To help you work out the codes, draw up several 8-by-8 grids and mark off your "dots-on, dots-off" design. Then figure out the code you need for each block of the grid.

# 5

## A Block Figure with CALL CHAR

Now that you've had some experience with defining your own characters, let's see if we can create a small "human" figure by turning dots on and off.

To begin, you need to create the figure on a character grid worksheet, like the one below. (Later, when you are creating your own characters, you may want to make copies of the worksheet, not only to design your symbols, but also to use in translating the symbol into the shorthand code of the CALL CHAR statement.)

CHAR Worksheet

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1	<input type="checkbox"/>	<input type="checkbox"/>	—	0	0000
ROW 2	<input type="checkbox"/>	<input type="checkbox"/>	—	1	0001
ROW 3	<input type="checkbox"/>	<input type="checkbox"/>	—	2	0010
ROW 4	<input type="checkbox"/>	<input type="checkbox"/>	—	3	0011
ROW 5	<input type="checkbox"/>	<input type="checkbox"/>	—	4	0100
ROW 6	<input type="checkbox"/>	<input type="checkbox"/>	—	5	0101
ROW 7	<input type="checkbox"/>	<input type="checkbox"/>	—	6	0110
ROW 8	<input type="checkbox"/>	<input type="checkbox"/>	—	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
INPUT TO CHAR:	_____			F	1111

Using the worksheet, we'll mark ones (1's) in the positions where the dots will be turned on:

*CHAR Worksheet*

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1	1	1	1	0	0000
ROW 2	1	1	1	1	0001
ROW 3	1	1	1	2	0010
ROW 4	1	1	1	3	0011
ROW 5	1	1	1	4	0100
ROW 6	1	1	1	5	0101
ROW 7	1	1	1	6	0110
ROW 8	1	1	1	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
				F	1111

INPUT TO CHAR: \_\_\_\_\_

Now, let's look at the same figure with the "on" dots shaded in, and let's fill in the shorthand codes for developing the character. This form of the worksheet shows you what the character will look like on the screen.

*CHAR Worksheet*

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1	■	■	99	0	0000
ROW 2	■	■	5A	1	0001
ROW 3	■	■	3C	2	0010
ROW 4	■	■	3C	3	0011
ROW 5	■	■	3C	4	0100
ROW 6	■	■	3C	5	0101
ROW 7	■	■	24	6	0110
ROW 8	■	■	24	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
				F	1111

INPUT TO CHAR: 995A3C3C3C3C24 24

---

# 5

---

By filling in the worksheet for both the character and the shorthand codes, we know that one line of our program will be

```
LET A$="995A3C3C3C3C2424"
```

But before we actually start our program, we need to discuss a bit further the process of defining a character. In our previous examples we *redefined* an already existing character, the exclamation point (character code 33). There are other character codes, however, that are *undefined* by the computer. These are available for you to use in building a customized character set in your graphics programs. The undefined character codes are grouped into the following sets (for color graphics):

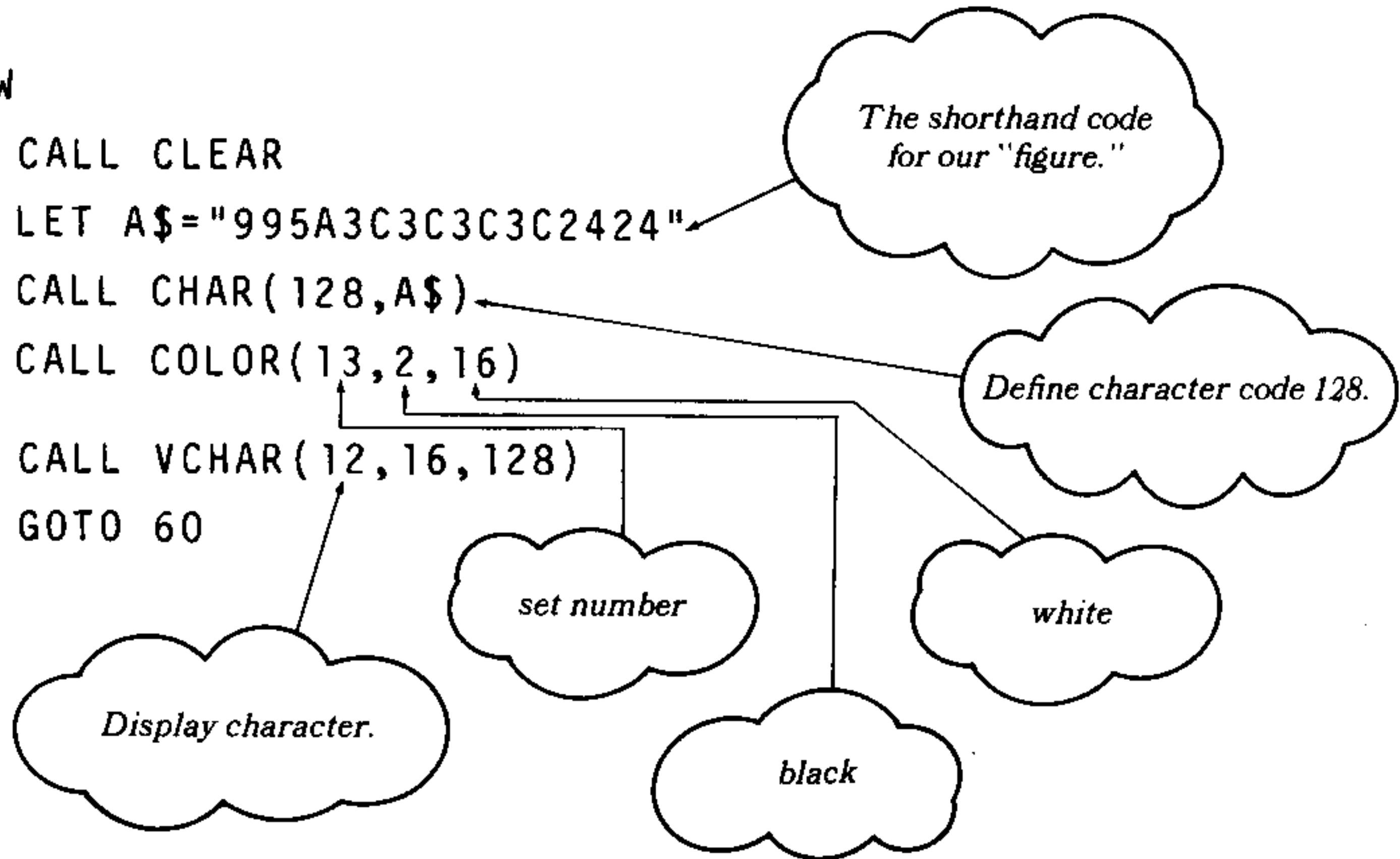
Set #13	Set #14	Set #15	Set #16
128	136	144	152
129	137	145	153
130	138	146	154
131	139	147	155
132	140	148	156
133	141	149	157
134	142	150	158
135	143	151	159

These extra character codes allow you to design special graphics characters for use in your own programs without giving up the standard keyboard characters. For example, you might want to design differently colored underline characters to highlight certain parts of a displayed message or develop a gameboard on the screen with directions displayed in standard text characters.

These codes and their corresponding set numbers are used in the CALL CHAR, CALL HCHAR, CALL VCHAR, and CALL COLOR statements exactly as we used the defined character codes and their set numbers. Let's use code 128 in our sample program.

OK, we're ready to begin our program. Enter these lines:

```
NEW
10 CALL CLEAR
20 LET A$="995A3C3C3C3C2424"
30 CALL CHAR(128,A$)
40 CALL COLOR(13,2,16)
50 CALL VCHAR(12,16,128)
60 GOTO 60
```



# 5

Now run the program and observe the small "person" on the screen. Remember, the figure is only one character in size, so look closely. When you're ready to stop the program, press **CLEAR**.

Would it be possible to animate our little figure? Yes, it would! By changing our program and incorporating one of the techniques we covered under ANIMATION, we can turn our character into Mr. Bojangles, the dancing man!

## *Mr. Bojangles*

As it's written presently, our program defines only one character. To make Mr. Bojangles appear to move, we'll need to define two characters that are alternately displayed in the same position. So we'll go to our CHAR Worksheets to design our two new characters.

### CHAR Worksheet

First Figure

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1	1	1	99	0	0000
ROW 2	1	1	5A	1	0001
ROW 3	1	1	3C	2	0010
ROW 4	1	1	3C	3	0011
ROW 5	1	1	3C	4	0100
ROW 6	1	1	3C	5	0101
ROW 7	1	1	44	6	0110
ROW 8	1	1	84	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
INPUT TO CHAR:	<u>"995A3C3C3C3C4484"</u>			D	1101
				E	1110
				F	1111

CHAR Worksheet

Second Figure

	LEFT BLOCK	RIGHT BLOCK	CODE	SHORT- HAND CODE	DOTS
ROW 1		1 1	<u>18</u>	0	0000
ROW 2	1	1 1	<u>99</u>	1	0001
ROW 3	1 1 1 1	1 1 1 1	<u>FF</u>	2	0010
ROW 4		1 1 1 1	<u>3C</u>	3	0011
ROW 5		1 1 1 1	<u>3C</u>	4	0100
ROW 6		1 1 1 1	<u>3C</u>	5	0101
ROW 7		1	<u>22</u>	6	0110
ROW 8		1	<u>21</u>	7	0111
				8	1000
				9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
				F	1111

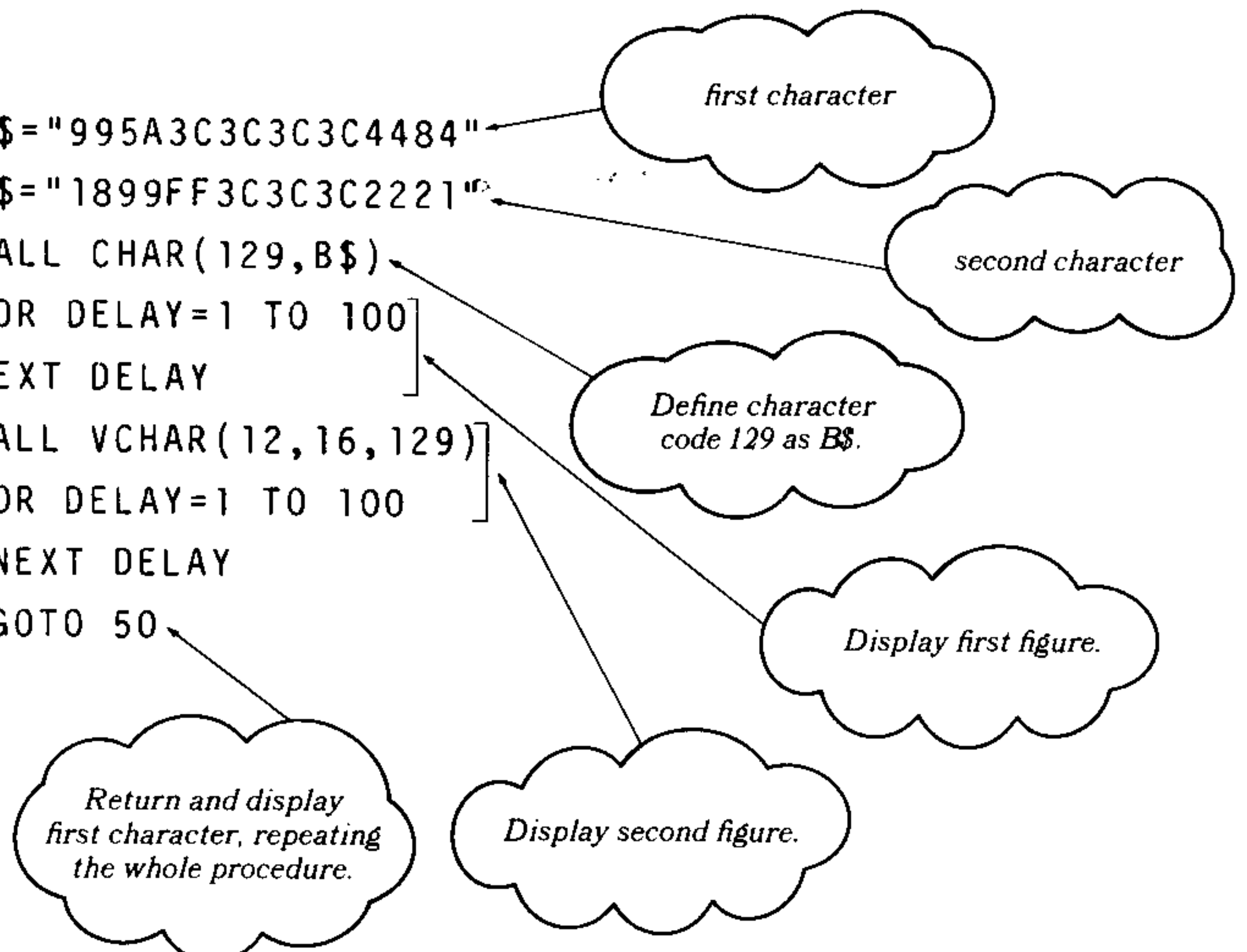
INPUT TO CHAR: "1899FF3C3C3C2221"

Now we're ready to edit the program. Enter these lines:

```

20 A$="995A3C3C3C3C4484"
25 B$="1899FF3C3C3C2221"
35 CALL CHAR(129,B$)
60 FOR DELAY=1 TO 100
70 NEXT DELAY
80 CALL VCHAR(12,16,129)
90 FOR DELAY=1 TO 100
100 NEXT DELAY
110 GOTO 50

```





---

# 5

---

Clear the screen and list the changed program so that you can see how it fits together:

```
LIST
10 CALL CLEAR
20 A$="995A3C3C3C3C4484"
25 B$="1899FF3C3C3C2221"
30 CALL CHAR(128,A$)
35 CALL CHAR(129,B$)
40 CALL COLOR(13,2,16)
50 CALL VCHAR(12,16,128)
60 FOR DELAY=1 TO 100
70 NEXT DELAY
80 CALL VCHAR(12,16,129)
90 FOR DELAY=1 TO 100
100 NEXT DELAY
110 GOTO 50
```

Now run the program and watch Mr. Bojangles dance! (To stop the program, press **CLEAR**.)

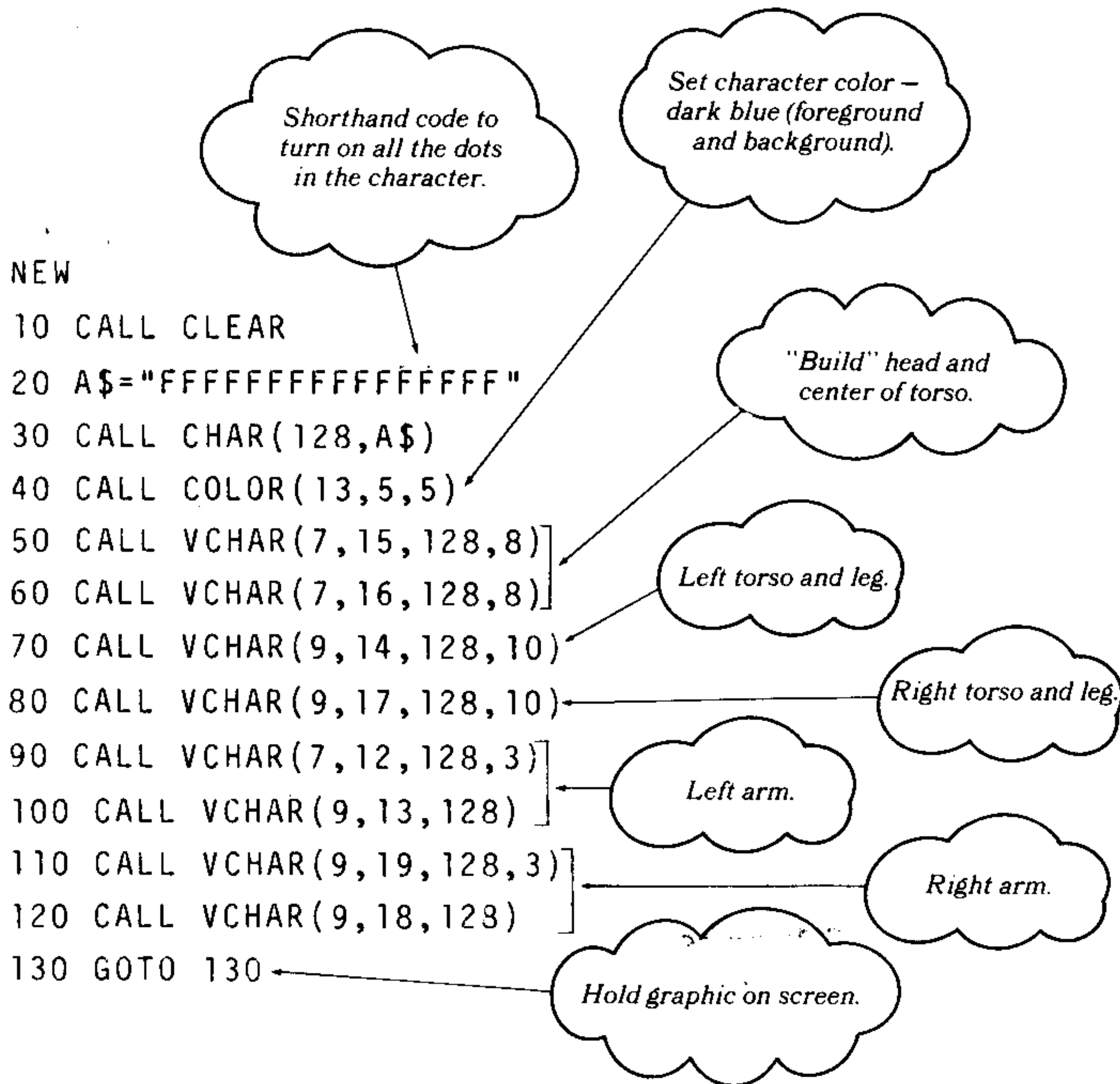
### ***Experiment!***

After running the program a few times, you might like to add a FOR-NEXT loop to make Mr. Bojangles dance across the screen (see page 107 for an example of this technique). Also, try creating other pairs of characters and placing their shorthand codes in lines 20 and 25. Can you turn Mr. Bojangles into an acrobat who flips from his hands to his feet and back again?

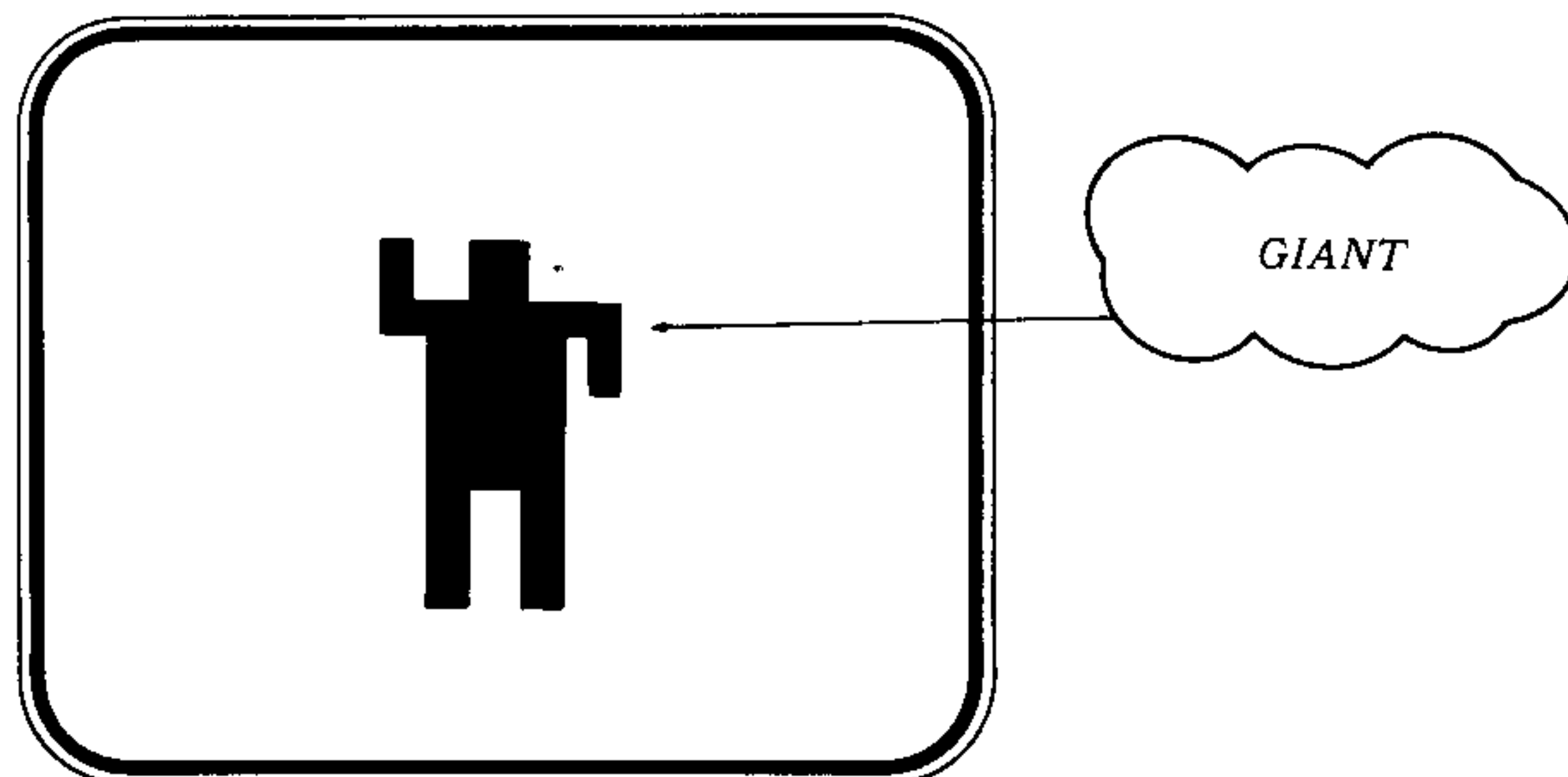
As we've mentioned, Mr. Bojangles is pretty small – only one character in size. Not all the designs you can create are limited to this small size. You can combine several small characters to construct bigger graphics that cover more of the screen. Our next program shows how to design a larger graphic using one small color character as our "building block."

*The Giant*

If you define one special character where all the dots are "on," you can then use it to paint in the rest of a large figure. The following program takes the small character just mentioned and creates a "giant" figure similar to the Mr. Bojangles character. Enter the program and see what it does:



When you run the program, you'll see a larger version of Mr. Bojangles:



# 5

Our dark-blue "giant" is rather angular and blocky, since it's created from a single angular character. You might like to rework the program, adding extra defined characters that allow you to soften the edges of the figure.

## Experiment!

Experiment with some other block designs using your "all-dots-on" character. Then try defining other characters to include in your graphics programs. The examples shown below will help to get you started.

*CHAR Worksheet*

	LEFT BLOCK	RIGHT BLOCK	CODE
ROW 1			<u>01</u>
ROW 2			<u>03</u>
ROW 3			<u>07</u>
ROW 4			<u>0F</u>
ROW 5			<u>1F</u>
ROW 6			<u>3F</u>
ROW 7			<u>7F</u>
ROW 8			<u>FF</u>

INPUT TO CHAR: "0103070F1F3F7FFF"

*CHAR Worksheet*

	LEFT BLOCK	RIGHT BLOCK	CODE
ROW 1			<u>18</u>
ROW 2			<u>3C</u>
ROW 3			<u>7E</u>
ROW 4			<u>FF</u>
ROW 5			<u>FF</u>
ROW 6			<u>7E</u>
ROW 7			<u>3C</u>
ROW 8			<u>18</u>

INPUT TO CHAR: "183C7EFFFF7E3C18"

*CHAR Worksheet*

	LEFT BLOCK	RIGHT BLOCK	CODE
ROW 1			<u>FF</u>
ROW 2			<u>7E</u>
ROW 3			<u>3C</u>
ROW 4			<u>18</u>
ROW 5			<u>18</u>
ROW 6			<u>3C</u>
ROW 7			<u>7E</u>
ROW 8			<u>FF</u>

INPUT TO CHAR: "FF7E3C18183C7EFF"

### Summary of Chapter 5

Chapter 5 has dealt entirely with computer graphics – the colorful patterns and designs you can create with your computer. Only two new statements have been introduced:

CALL SCREEN	Allows you to change the Run Mode screen to any color you choose.
CALL CHAR	Defines a character code for a character you create.

In addition to these two statements, you've experimented with the following techniques:

- Creating large color blocks on the screen
- Making patterns and designs from standard characters
- Animating your graphics
- Creating your own characters by turning dots "on" and "off"

The ability to create color graphics can add a lot of excitement to your computer programming. We hope that you've enjoyed this introduction to graphics and that you'll find even more creative ways to use your computer.

This chapter concludes our "introductory tour" through the TI BASIC language. You are now well launched into your programming career. For more advanced features you may want to consult the "BASIC Reference" section of the *User's Reference Guide*.

Here are some sections we suggest:

- Additional editing techniques – see EDIT
- Automatic line numbering – see NUMBER
- BREAK
- TRACE
- Recording programs on the TI Disk Memory System or a cassette tape recorder – see SAVE and OLD

If you'd like to consult a programming book on an intermediate level, we can recommend an excellent one: Herbert Peckham's *Programming BASIC with the TI Home Computer* (New York: McGraw-Hill Book Company, 1979). You'll find a coupon for ordering this book on page 143.

Congratulations and best wishes for continued success in BASIC programming!

## APPENDIX A

### Musical Tone Frequencies

The tones produced by the computer are generated by the CALL SOUND statement. (See Chapter 1 for an explanation of the CALL SOUND statement.)

The *frequency* designated in the CALL SOUND statement determines the tone that is produced. The acceptable value range for frequencies is from 110 to 44,733 Hertz (cycles per second). Noninteger entries within this range are acceptable as inputs in the CALL SOUND statement, but they are rounded to nearest integers by the computer before execution.

The following table gives frequency values (rounded to integers) for four octaves in the tempered scale (one half-step between notes). While these values do not, of course, represent the entire range of tones – or even of musical tones – they can give you a basis for musical programs. (See Appendix D for a frequency-generating program.)

<i>Frequency</i>	<i>Note</i>	<i>Frequency</i>	<i>Note</i>
110	A	440	A (above middle C)
117	A <sup>#</sup> ,B <sup>b</sup>	466	A <sup>#</sup> ,B <sup>b</sup>
123	B	494	B
131	C (low C)	523	C (high C)
139	C <sup>#</sup> ,D <sup>b</sup>	554	C <sup>#</sup> ,D <sup>b</sup>
147	D	587	D
156	D <sup>#</sup> ,E <sup>b</sup>	622	D <sup>#</sup> ,E <sup>b</sup>
165	E	659	E
175	F	698	F
185	F <sup>#</sup> ,G <sup>b</sup>	740	F <sup>#</sup> ,G <sup>b</sup>
196	G	784	G
208	G <sup>#</sup> ,A <sup>b</sup>	831	G <sup>#</sup> ,A <sup>b</sup>
220	A (below middle C)	880	A (above high C)
220	A (below middle C)	880	A (above high C)
233	A <sup>#</sup> ,B <sup>b</sup>	932	A <sup>#</sup> ,B <sup>b</sup>
247	B	988	B
262	C (middle C)	1047	C
277	C <sup>#</sup> ,D <sup>b</sup>	1109	C <sup>#</sup> ,D <sup>b</sup>
294	D	1175	D
311	D <sup>#</sup> ,E <sup>b</sup>	1245	D <sup>#</sup> ,E <sup>b</sup>
330	E	1319	E
349	F	1397	F
370	F <sup>#</sup> ,G <sup>b</sup>	1480	F <sup>#</sup> ,G <sup>b</sup>
392	G	1568	G
415	G <sup>#</sup> ,A <sup>b</sup>	1661	G <sup>#</sup> ,A <sup>b</sup>
440	A (above middle C)	1760	A

All characters that print on the screen (letters, numbers, and symbols) are identified by numeric *character codes*. The standard characters are represented by character codes 32 through 127. These ninety-six codes are grouped into twelve *character sets* for color graphics purposes.

**Standard Character Codes**

<b>Set #1</b>		<b>Set #2</b>		<b>Set #3</b>		<b>Set #4</b>	
Code #	Character	Code #	Character	Code #	Character	Code #	Character
32	(space)	40	(	48	0	56	8
33	!	41	)	49	1	57	9
34	"	42	*	50	2	58	:
35	#	43	+	51	3	59	;
36	\$	44	,	52	4	60	<
37	%	45	-	53	5	61	=
38	&	46	.	54	6	62	>
39	'	47	/	55	7	63	?

<b>Set #5</b>		<b>Set #6</b>		<b>Set #7</b>		<b>Set #8</b>	
Code #	Character	Code #	Character	Code #	Character	Code #	Character
64	@	72	H	80	P	88	X
65	A	73	I	81	Q	89	Y
66	B	74	J	82	R	90	Z
67	C	75	K	83	S	91	[
68	D	76	L	84	T	92	\
69	E	77	M	85	U	93	]
70	F	78	N	86	V	94	^
71	G	79	O	87	W	95	_

<b>Set #9</b>		<b>Set #10</b>		<b>Set #11</b>		<b>Set #12</b>	
Code	Character	Code	Character	Code	Character	Code	Character
96	`	104	H	112	P	120	X
97	A	105	I	113	Q	121	Y
98	B	106	J	114	R	122	Z
99	C	107	K	115	S	123	{
100	D	108	L	116	T	124	
101	E	109	M	117	U	125	}
102	F	110	N	118	V	126	~
103	G	111	O	119	W	127	DEL

There are thirty-two additional character codes (128-159) available for use in defining special characters for graphics programs. (See Chapter 5 for a discussion of character definition.) Again, these codes are grouped into four sets for color graphics.

**Special Character Codes**

<b>Set #13</b>	<b>Set #14</b>	<b>Set #15</b>	<b>Set #16</b>
128	136	144	152
129	137	145	153
130	138	146	154
131	139	147	155
132	140	148	156
133	141	149	157
134	142	150	158
135	143	151	159

---

## APPENDIX C

### Color Codes

---

---

Sixteen colors are available for color graphics programs in TI BASIC. These colors are designated by numeric codes in the CALL COLOR and CALL SCREEN statements. (See Chapter 2 for a discussion of CALL COLOR and Chapter 5 for an explanation of CALL SCREEN.)

#### Color Codes

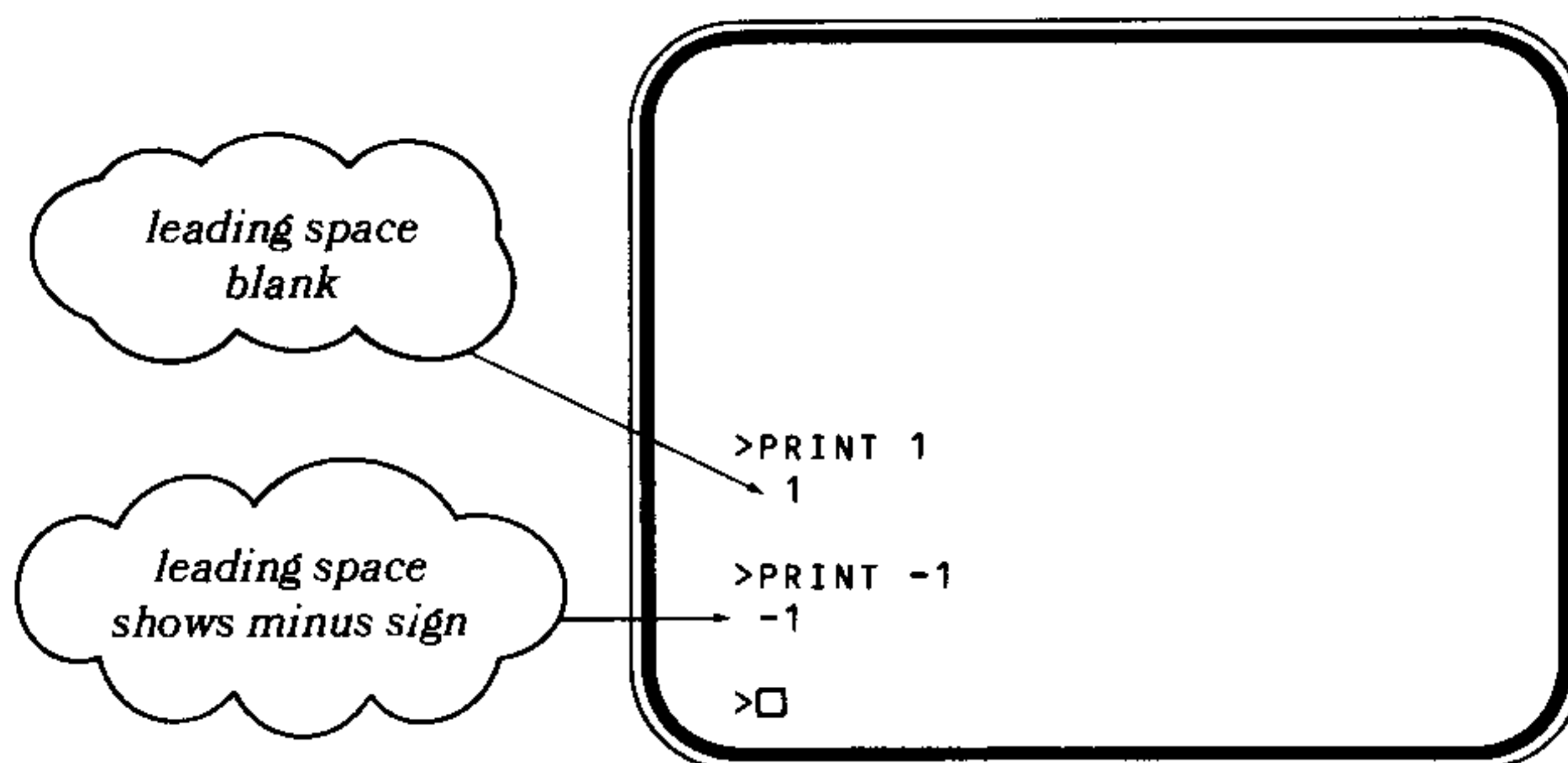
<i>Color</i>	<i>Code #</i>	<i>Color</i>	<i>Code #</i>
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

If your computer is to be a useful tool, you'll need to know about some of its computational powers. This appendix first discusses the ways your computer handles and displays numbers and then shows you how to perform exponentiation (powers and roots of numbers). Next is a section on the order in which mathematical operations are performed. Finally, certain other mathematical functions are listed for you. You'll find that your computer can eliminate much of the drudgery of computation, leaving you with more time to explore the theory and fun of mathematics.

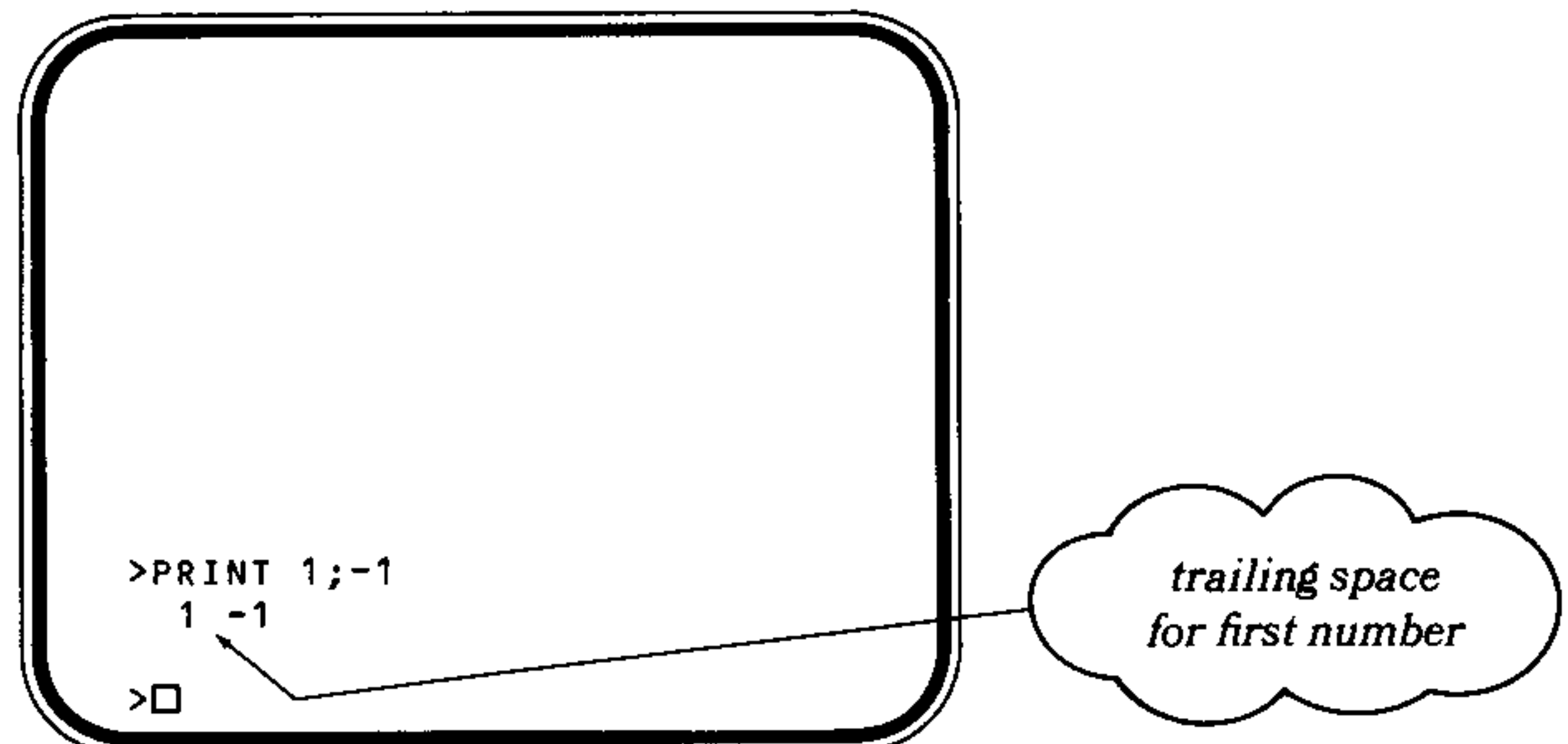
### Decimal Notation

The Texas Instruments computer accepts and displays numbers, within certain limits, in the traditional decimal form.

In Chapter 3, we mentioned briefly that numbers are displayed with a *leading space* and a *trailing space*. The *leading space* is reserved for the *sign* (positive or negative) of the number. If the number is positive, this space will be blank. If the number is negative, this space will show a minus sign. Here's an example of both situations:



The *trailing space* is there to make sure that two numbers on the same line of the screen will always have at least one space between them, even if you use a semicolon as a PRINT separator. (The semicolon instructs the computer to leave no spaces between PRINT items.) Try this Immediate Mode example to see the effect of the trailing space:





---

## APPENDIX D

### Mathematical Operations

---

---

Without this trailing space the two numbers would appear like this:

1-1

The screen shows a maximum of ten digits for any number. If an integer (whole number) consists of ten digits or less, the computer shows the number *without* a decimal point to the right:

```
>PRINT 1;12345;1234567890
1 12345 1234567890
>□
```

If the number is a decimal fraction with ten digits or less, the computer automatically places the decimal point in the correct position:

```
>PRINT 1/8;7.525/5;159.1395/5
.125 1.505 31.8279
>□
```

Notice the first example above,  $1/8 = .125$ . If a number is less than +1 and greater than -1, so that the digit to the left of the decimal point would be zero, the zero is not displayed.

Most of the time, the numbers you see and work with will be shown in this normal display format. But what about numbers that consist of *more* than ten digits, such as

723,895,274,524  
0.00000000014896

The computer can also handle numbers like these, but it must use a special display format to do so.

### Floating Point Form or Scientific Notation

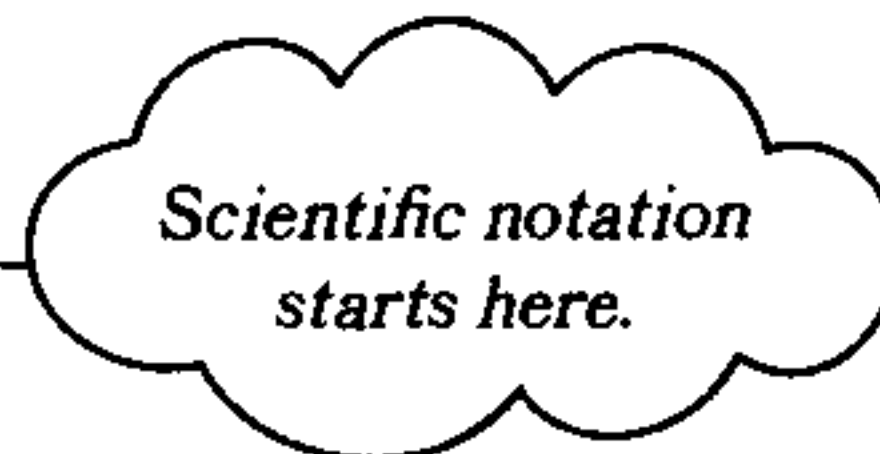
To display numbers with more than ten digits, your computer uses a special kind of notation. You'll see several names in computer books referring to this type of notation; two of the more common names are *floating point form* and *scientific notation*. Here we'll refer to the special display format as scientific notation.

Before we discuss scientific notation, let's try a program to see how whole numbers (integers) look in this display format. Enter these lines:

```
NEW
10 LET A=10
20 FOR I=1 TO 12
30 PRINT A
40 LET A=A*10
50 NEXT I
60 END
```

Now clear the screen and run the program. You'll see these results:

```
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
1.E+10
1.E+11
1.E+12
```



As soon as the value of A becomes an integer with more than ten digits, the computer switches over to the special display format. Here's what this format represents:

```
1.E+10 means  $1 \times 10^{10}$  or 10,000,000,000
1.E+11 means  $1 \times 10^{11}$  or 100,000,000,000
1.E+12 means  $1 \times 10^{12}$  or 1,000,000,000,000
```

Numbers that are printed in scientific notation will always have this form:  
base number E exponent

The base number (*mantissa*) is always displayed with one digit (1 through 9) to the left of the decimal point. There can be a maximum of six digits in the mantissa (one to the left of the decimal point; up to five to the right of the decimal). "E" stands for "× (times) 10 raised to some power," and the *exponent* (power) is always displayed with a plus or minus sign (+ or -) followed by a one- or two-digit number (1 through 99).

*Note:* If you attempt to print a number with an exponent greater than 99 but less than the computer's limits, you'll see this format:

```
mantissa E + **
or
mantissa E - **
```

---

## APPENDIX D

### Mathematical Operations

---

The two asterisks indicate that the number is within the valid computing range of the computer, but the exponent is too large to be displayed in the allotted space. (For a discussion of the computational ranges, see the "BASIC Reference" section of the *User's Reference Guide*.)

Here are several examples of integers that are displayed by the computer in scientific notation:

```
>PRINT 1234512345123
      1.23451E+12

>PRINT 45678900000000
      4.56789E+13

>PRINT 98765432100
      9.87654E+10

>□
```

Notice that the *sign* of the exponent tells us how to convert scientific notation back into standard decimal form. If the sign is a +, we move the decimal point to the right. If the sign is a -, we move the decimal point to the left. The *exponent* tells us how many places to move the decimal point:

1.11111E+10 means 11111100000

We have moved the decimal ten places to the right:

11111100000.

Integers with more than ten digits, then, are always displayed in scientific notation. Now let's see how the computer handles noninteger numbers (numbers with fractional parts). Consider the number 0.000000000000123. It will not fit into the ten-digit display, so the computer shows it in scientific notation. Try this:

```
>PRINT 0.000000000000123
      1.23E-13

>□
```

*Tells how many places to move the decimal to the left.*

The following program generates some very small noninteger numbers:

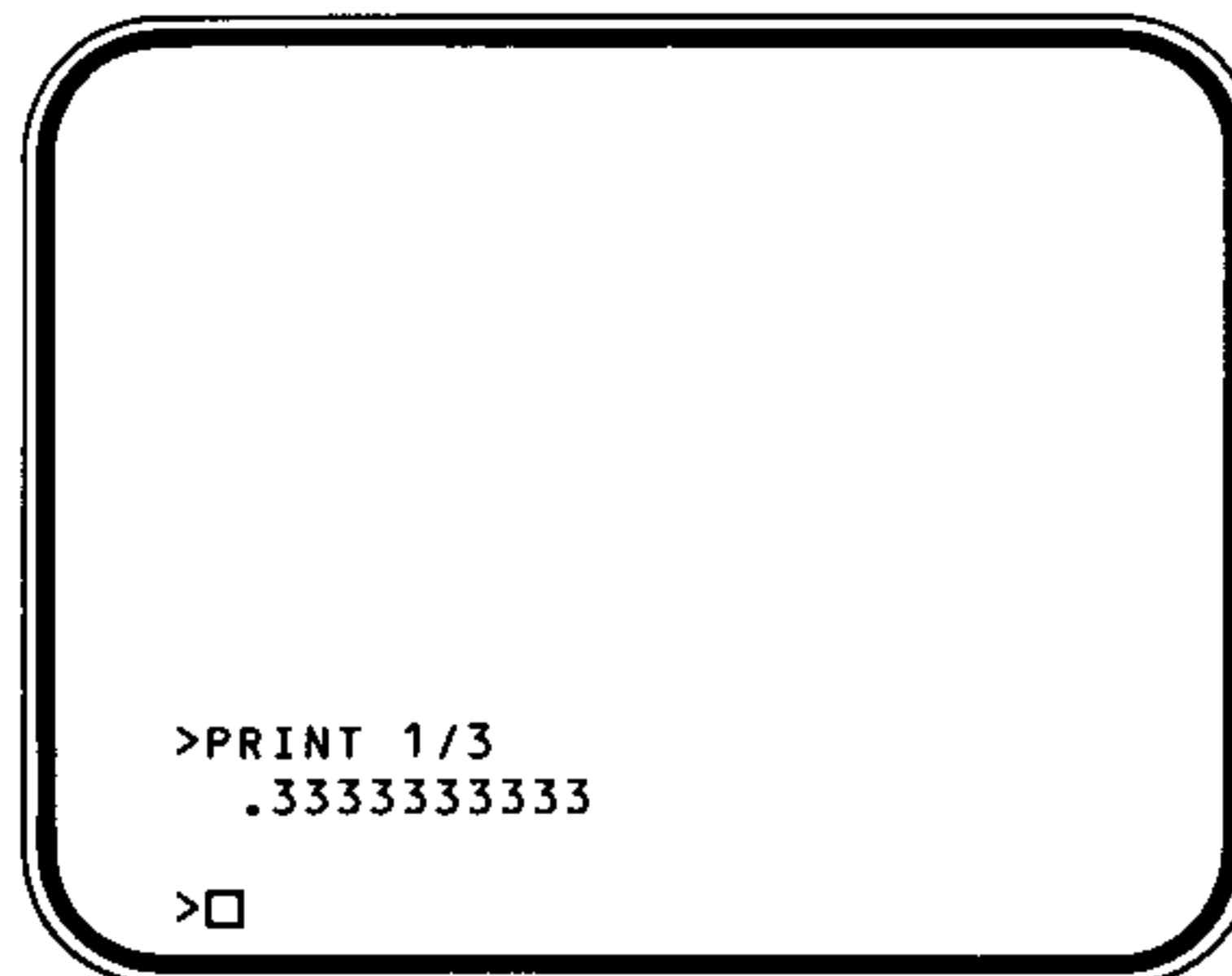
```
NEW
10 LET A=10
20 FOR I=1 TO 14
30 PRINT A
40 LET A=A/10
50 NEXT I
60 END
```

Clear the screen and run the program. The results are:

```
10
1
.1
.01
.001
.0001
.00001
.000001
.0000001
.00000001
.000000001
.0000000001
.00000000001
1.E-11
1.E-12
```

This program and the previous examples we've seen might lead us to think that nonintegers with more than ten digits are always displayed in scientific notation, just as integers are. This is not always true, however. *Noninteger numbers with more than ten digits are printed in scientific notation only if they can be presented more accurately in scientific notation than in the normal form.*

This point is very important. Consider an example that we've tried before:



---

## APPENDIX D

### Mathematical Operations

---

We know that  $.3333333333\dots$  is a repeating decimal that goes on infinitely. Why, then, does the display show the result in normal form? The answer is that  $.3333333333$  is more accurate than  $3.33333E-1$ ; that is, more *significant digits* (digits that reflect the actual mathematical value of the number) can be shown in normal form than in scientific notation.

Scientific notation is just a "shorthand" method for writing long numbers, whether they are very large or very small quantities. It allows the computer to handle, in the most accurate form possible, numbers that otherwise could not be adequately displayed in the ten-digit normal form.

#### *Entering Numbers in Scientific Notation*

Up to this point, we've only entered numbers in the normal decimal form. It is also possible, however, to enter numbers in scientific notation. Try this example:

```
>PRINT 1.23456E10
1.23456E+10
>□
```

NOTE: "E" must be an upper-case letter when you enter a number in scientific notation.

Notice that, unless you enter a minus sign before the mantissa and/or the exponent, these are assumed to be positive.

```
>PRINT 2.574E13
2.574E+13

>PRINT -5.5E-11
-5.5E-11
>□
```

If you enter a number in scientific notation, but the computer can show it in normal form, it will do so. Try this:

```
>PRINT 5.555E3
5555
>□
```

Whenever you are using extremely large or small numbers in a computation, entering the numbers in scientific notation can be very handy.

### Exponentiation

In the previous section we talked about *exponents* and *powers* of 10. Now we need to discuss some of the "higher math" capabilities of your computer; specifically, *powers* and *roots*.

#### *Powers*

Quite often in mathematical calculations, we must raise some number to a *power*, such as

$$8^3 \text{ (or } 8 \times 8 \times 8)$$
$$25^2 \text{ (or } 25 \times 25)$$

To perform exponentiation (raising a number to a power) on the computer, we do this:

```
>PRINT 8^3
512
>PRINT 25^2
625
>□
```

The *exponentiation symbol* ( $\wedge$ ) tells the computer that the number that follows is a power.

---

## APPENDIX D

### Mathematical Operations

---

Let's say that we have this mathematical expression to evaluate:

$$y = x^3$$

We want to find all the values for  $y$  where  $x$  equals 1 through 10. So we enter this short program:

```
NEW
10 CALL CLEAR
20 FOR X=1 TO 10
30 Y=X^3
40 PRINT "Y=";Y
50 NEXT X
60 END
```

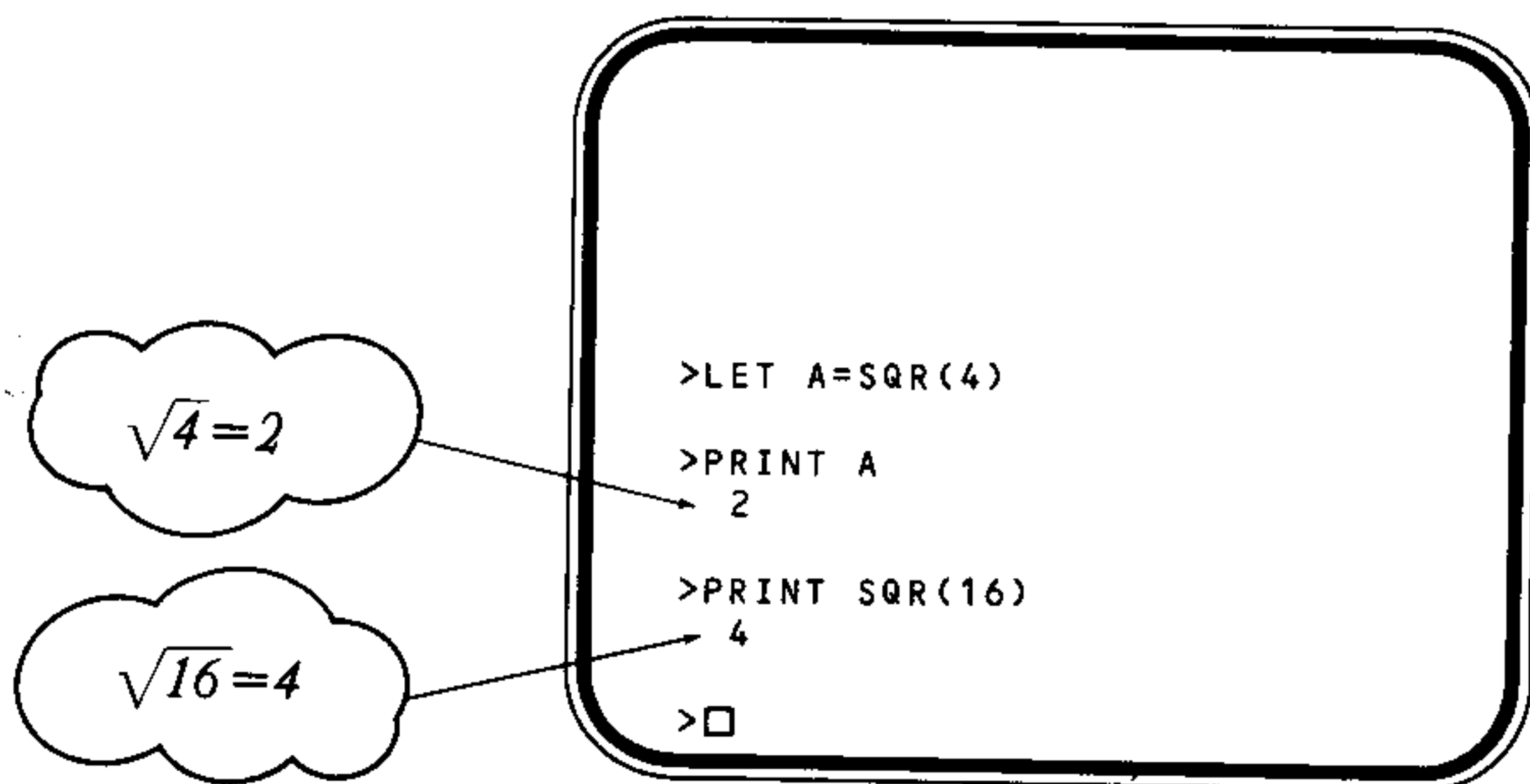
When we run the program, we'll see the following values for  $y$ :

```
Y= 1
Y= 8
Y= 27
Y= 64
Y= 125
Y= 216
Y= 343
Y= 512
Y= 729
Y= 1000
```

The computer completes the program for us very quickly! We have the values we need and can go on to other computations.

### Roots

Finding a *root* of a number is another very common mathematical problem. The *square root* is one we've all heard of – and probably used – at some point in our educations. Since many, many calculations call for square roots, this function is built into the computer:



The letters SQR stand for "square root of" and instruct the computer to find the square root of the number or expression contained within the parentheses.

Other roots must be computed by using a form of exponentiation. Computing a root of a number is the same function as raising the number to a power which is the reciprocal of the root; that is,

$$\sqrt[3]{125} \text{ is the same as } 125^{(1/3)}$$

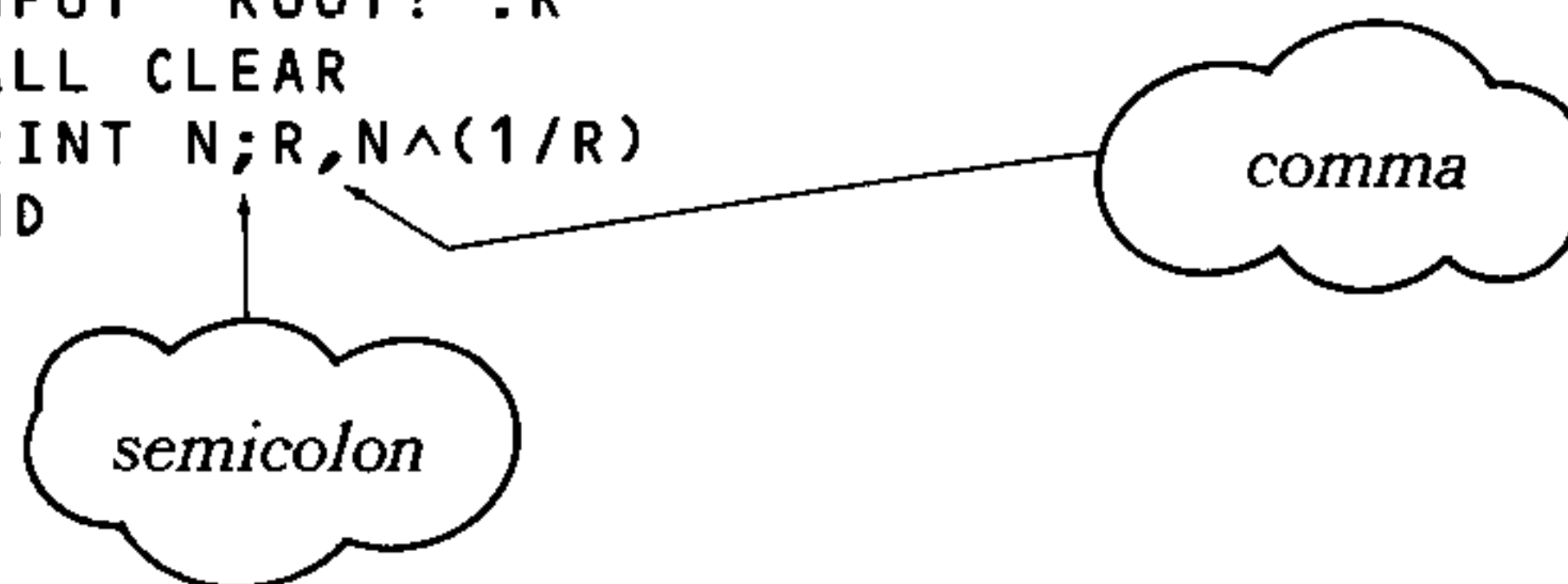
Try this example:

```
>PRINT 125^(1/3)
5.
>□
```

Notice that we had to use parentheses around the exponent 1/3. The parentheses notify the computer that the whole expression makes up the exponent. (You'll see why this is necessary when we discuss "Order of Operations.")

Here's a program that helps you compute any root of any number (within the computer's limits and the bounds of mathematical rules, of course).

```
NEW
10 CALL CLEAR
20 INPUT "NUMBER?":N
30 INPUT "ROOT?":R
40 CALL CLEAR
50 PRINT N;R,N^(1/R)
60 END
```



When you run the program, you'll first be asked to input the number for which you want to find the root. Let's enter 27 for our example.



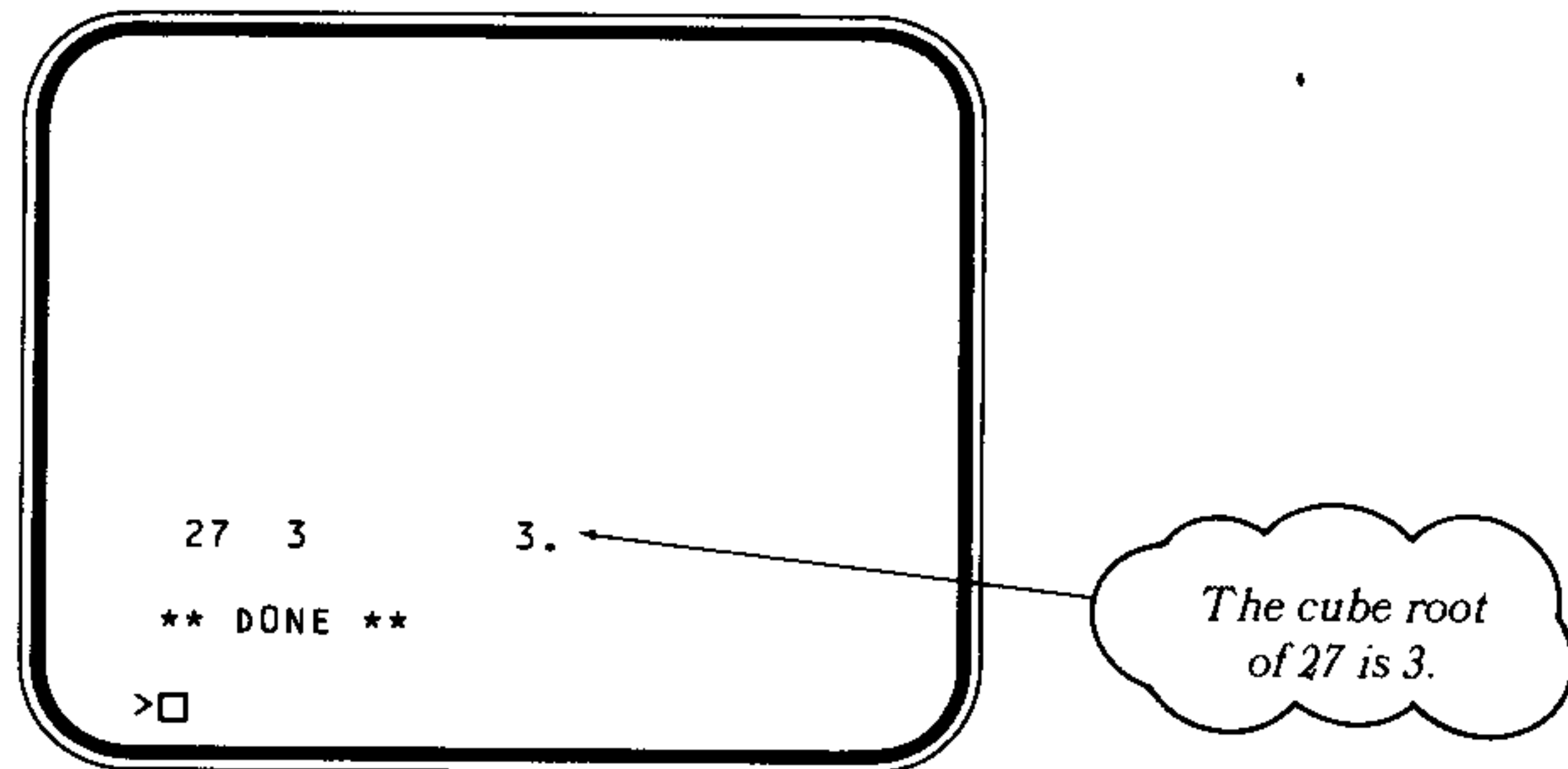
---

## APPENDIX D

### Mathematical Operations

---

Next you're asked for the root you want to find. Let's say we want the *cube root*, so we type 3 and press **ENTER**.

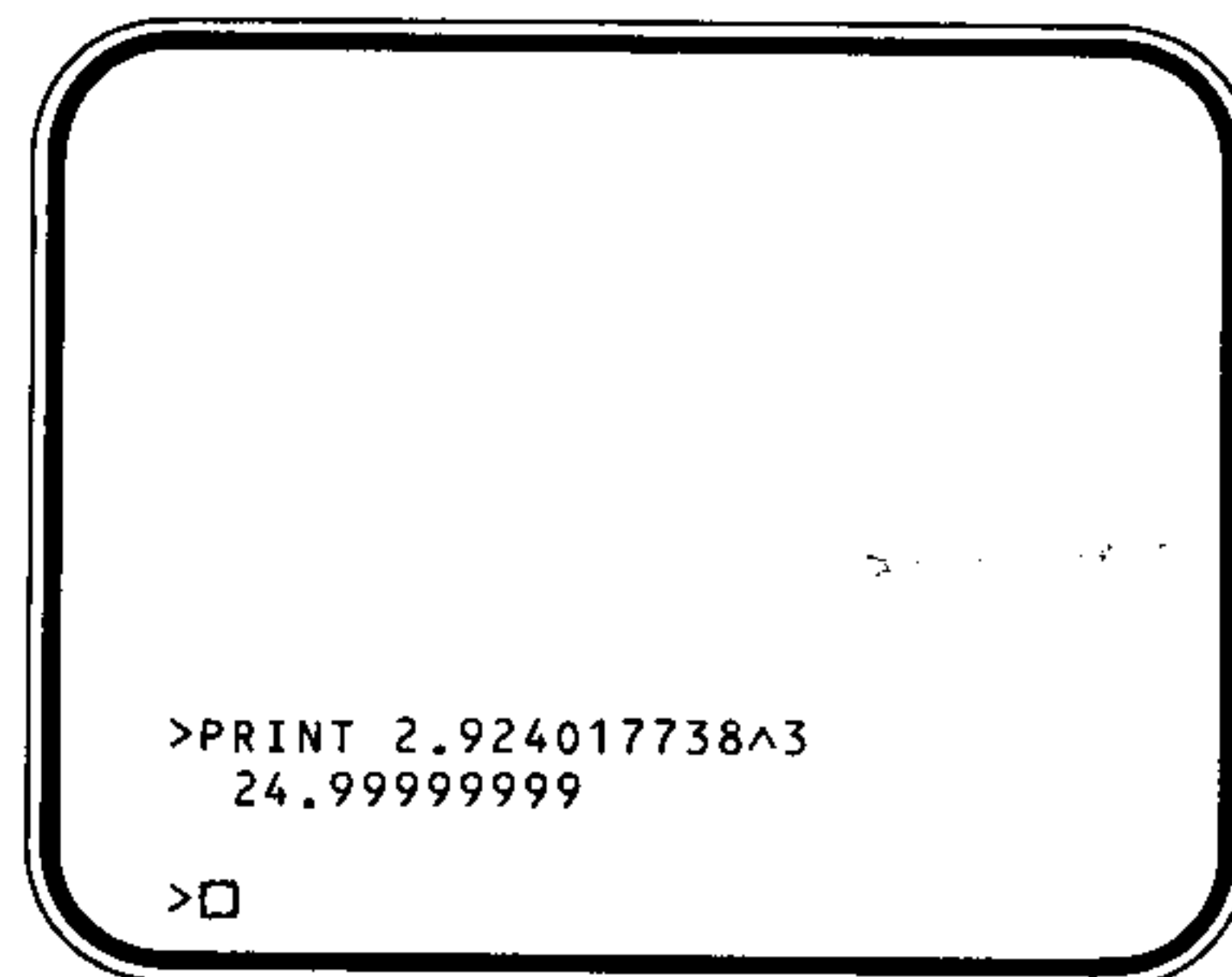


```
27 3 3.  
** DONE **  
>□
```

The cube root of 27 is 3.

Run the program again, and this time enter 2401 for the number and 4 for the root. Did you get the answer 7?

Of course, not all numbers work out to results that are nice, neat integers. Try the program again, entering 25 for the number and 3 for the root. You'll get 2.924017738 as your answer. Now check the answer in the Immediate Mode, by raising 2.924017738 to the power of 3:



```
>PRINT 2.924017738^3  
24.99999999  
>□
```

You don't quite get back to your original 25. That's because 2.924017738 is not the "exact" cube root of 25; it's an "approximate" root, rounded to ten digits so that it can be displayed.

All computing devices must "round off" calculated results at some point. Where a computer rounds a result depends on the computational and display limits of the machine. To make sure that the accuracy of the last displayed digit is within certain limits, most computers and many calculators actually perform computations internally with more digits than they can display. These extra or "guard" digits are retained in the computer's internal registers, but they can't be shown on the screen, due to space limitations.

We can, however, demonstrate the presence of these internal "computational" digits. Let's use the same problem we performed earlier:

```
>LET A=25^(1/3)

>PRINT A
  2.924017738

>PRINT A^3
  25.

>□
```

The "memory box" labeled A retains all the internal digits as well as the rounded result shown on the screen. Therefore, with the greater accuracy provided by the internal digits, we get back our original 25 when we raise A to the power of 3.

One special note of caution: Your computer will give you an error message if you try to raise a negative number of a fractional power; therefore, you cannot use the exponentiation routine to find roots of a negative number *without* taking other steps. See the Sign (SGN) and Absolute Value (ABS) functions in the "BASIC Reference" section of the *User's Reference Guide*.

### **Order of Operations**

In Chapter 3 we discussed the order the computer follows to complete problems involving multiplication, division, addition, and subtraction. We also demonstrated that an expression within parentheses is evaluated before the rest of the problem is solved. The order of operations, then, was listed as:

- (1) Complete everything inside parentheses.
- (2) Complete multiplication and division.
- (3) Complete addition and subtraction.

Now we need to add another level to this order. Exponentiation (raising a number to a power or finding a root of a number) is performed before any other mathematical operation. So our new order becomes:

- (1) Complete parenthetical expressions.
- (2) Complete exponentiation.
- (3) Complete multiplication and division.
- (4) Complete addition and subtraction.

Let's try some examples that help to demonstrate these concepts.

---

## APPENDIX D

### Mathematical Operations

---

First, we'll define some variable names for the quantities we'll be using in our calculations. Enter these lines:

```
LET A=5
LET B=2
LET C=10
LET D=4
```

Now we're ready for the calculations:

```
>PRINT B*C^B
  200

>PRINT A+B*C^B
  205

>PRINT ((A+B)*C)^B/D
  1225

>□
```

Here's the order the computer followed in each of these examples:

*First problem*     $10^2=100$   
                   $2 \times 100=200$

*Second problem*  $10^2=100$   
                   $2 \times 100=200$   
                   $5 + 200=205$

*Third problem*     $5 + 2 = 7$   
                   $7 \times 10 = 70$   
                   $70^2 = 4900$   
                   $4900 \div 4 = 1225$

Notice that this last problem utilized two sets of parentheses, one within the other. In this situation the computer evaluates the innermost set of parentheses first.

As you saw when we discussed the roots of numbers, the exponent of a number can also be a numeric expression enclosed in parentheses. Let's try a few more examples, using the values already stored in the computer's memory.

```
>PRINT ((A+B)*(A+B))^(B/D)
7.

>PRINT B^(D/B)+A*C
54.

>□
```

The first problem essentially squared the number 7 and then took the square root of the result:

$$\begin{aligned}(A+B) &= 5+2=7 \\ (A+B)*(A+B) &= 7 \times 7=49 \\ B/D &= 2 \div 4=.5 \\ 49^{.5} &= \sqrt{49}=7\end{aligned}$$

The second problem is solved like this:

$$\begin{aligned}D/B &= 4 \div 2=2 \\ B^{(D/B)} &= 2^2=4 \\ A*C &= 5 \times 10=50 \\ 4+50 &= 54\end{aligned}$$

The following program not only demonstrates the computational power of your computer, but also plays a scale for you!

The relationship between the frequencies of notes in the tempered scale can be algebraically expressed as

$$y = xk^n$$

where  $x$  = the frequency of the first note of the scale,

$k$  = a constant,  $\sqrt[12]{2}$ ,

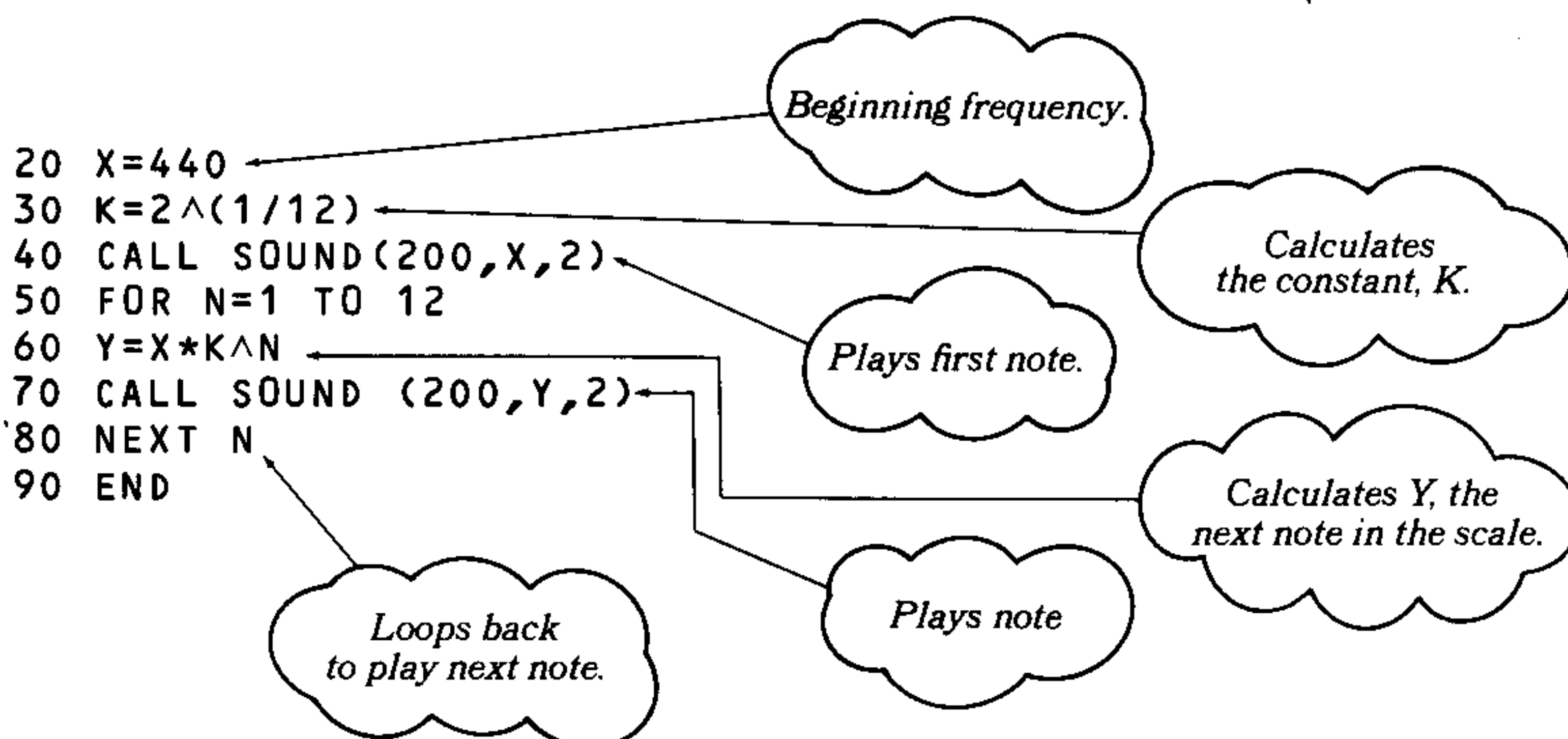
$n$  = the number of half-steps between note  $x$  and note  $y$

$y$  = the frequency of the next note you want to play

## APPENDIX D

### Mathematical Operations

There are twelve notes in the tempered scale, and between each note and the next is one half-step. The following program, starting with a frequency of 440 (A above middle C on a piano keyboard), calculates and plays each note in the scale:



Run the program and listen to the music!

### Other Mathematical Functions

Several other mathematical functions, in addition to those we've already covered, are available in TI BASIC. We won't discuss these in detail, but we want to list some of them for you, because they can be a great help in performing mathematics with your computer.

#### Trigonometric Functions

These trigonometric functions are available:

**SIN( )** – Finds the *sine* of the number or numeric expression enclosed in parentheses.

*A number or numeric expression goes here.*

**COS( )** – Finds the *cosine* of the number or numeric expression enclosed in parentheses.

**TAN( )** – Finds the *tangent* of the number or numeric expression enclosed in parentheses.

**ATN( )** – Finds the *arctangent* of the number or numeric expression enclosed in parentheses.

**Note:** All trigonometric functions are performed by the computer in *radians*, rather than *degrees*. Therefore, if your data is measured in degrees, you'll need to convert the measurement to radians before using it with the function. (To convert an angle from degrees to radians, multiply by  $\pi/180$ . To convert from radians to degrees, multiply by  $180/\pi$ .)

### Logarithms

The computer calculates the *natural log* and *natural antilog* (based on  $e = 2.718281828$ ) of a number:

A number or numeric expression goes here.

**LOG ( )** Computes the natural logarithm of the number or numeric expression enclosed in parentheses.

**EXP ( )** Computes the natural antilogarithm of the number or numeric expression enclosed in parentheses.

To convert the natural logarithm of a number to the *common log* of the number, simply divide the natural log by the natural log of 10. For example, if you want to find the common log of 3, you would use this procedure:

```
>A=LOG(3)/LOG(10)
>PRINT A
.4771212547
>□
```

common log of 3

### Absolute Value

Calculations often require the use of the absolute value of a number. This has the effect of making the number positive, regardless of its sign. Here's how to instruct the computer to find and utilize the absolute value of a number:

A number or numeric expression goes here.

**ABS ( )** Finds the absolute value of the number or numeric expression in parentheses.

There are other mathematical functions available, and you'll find them listed and discussed under "Functions" in the "BASIC Reference" section of the *User's Reference Guide*. The functions we've illustrated here, however, should help you discover many ways to use your computer as a computational tool.

# Index

<b>A</b>	<i>Page</i>		
Absolute value	141	<b>I</b>	
Adding program lines	31-32	IF-THEN statement	83-85
Animation	104-120	Immediate mode, definition of	7
<b>B</b>		INPUT statement	33-35
BASIC, definition of	5	INT function	69-72
Branching	83-84	<b>L</b>	
<b>C</b>		LET statement	13-17
CALL CHAR	108-110	Line number	28
CALL CLEAR	10-11	LIST command	28-29
CALL COLOR	40-45	Logarithms	141
CALL HCHAR	20-25	Loop	
CALL KEY	93-94	delay loop	41-43, 51-53
CALL SCREEN	97-98	FOR-NEXT	48-57
CALL SOUND	17-20	GO TO	38-45
for noise	18-19	loop counter	54
for one tone	17	nested loop	53-57
for three tones	18	<b>M</b>	
for two tones	18-19	Mathematics	
CALL VCHAR	20-25	Absolute value	141
Character codes	21, 41-42, 125	Decimal notation	127-128
Character, definition of	21	Exponentiation	133-137
defining customized		Logarithms	141
character set	108-118	Order of operation	67-69, 137-140
"grid"	108-113	Parentheses	67-69, 137-140
standard set	42, 125	Scientific notation	69, 129-133
Character grid worksheet	114-116	Trigonometric functions	140
Color codes	42, 126	Musical tone frequencies	124
Commands	28-29	<b>N</b>	
Computer programming,		NEW command	26, 28-29
definition of	5	Normal display form	58, 61, 127-128
Cursor-control keys	12-13	Numbers	
Cursor, definition of	8	Display of numbers	127-133
<b>D</b>		Random numbers	73-78
Defining characters	108-118	Rounding of numbers	136
Deleting program lines	32-33	Numeric variables	13-17
Duration of tone	17	<b>O</b>	
<b>E</b>		Order of operation	
Editing programs	31-37	in mathematics	67-69, 137-140
Error correction	12-13, 26	in programs	27-28
Error messages	11-12, 45-46, 56-57, 80, 85	<b>P</b>	
Exponentiation	133-137	Powers	133-134
<b>F</b>		PRINT statement	
FOR-NEXT statement	48-57	definition	8
Functions		with arithmetic operations	16
INT	69-72	with colon	62
RND	73-78	with comma	57-60
TAB	63-66	with numeric variables	14-16
<b>G</b>		with semicolon	60-62
GO TO statement	38-45	with string variables	35-36, 59-62
Graphics "grid" (character		with TAB	63-66
positioning)	20-21	Program structure	27-28, 47
Graphics line	20	Prompting message with INPUT	34
Graphics subprograms		Prompting symbol, purpose of	8, 26
CALL COLOR	40-45		
CALL HCHAR	20-25		
CALL SCREEN	97		
CALL VCHAR	20-25		
		<b>R</b>	
		Random numbers	73-80
		Changing range	75-79
		Setting limits	82
		RANDOMIZE	74-75
		RND function	73-80
		Roots	134-137
		RUN command	27-29
		<b>S</b>	
		Scientific notation	69, 129-133
		Scrolling, definition of	9
		with Immediate Mode graphics	24
		"Shorthand" codes	108-114
		Simulation, definition of	73
		Dice-rolling simulations	77-79
		Statements	
		CALL CHAR	108-110
		CALL CLEAR	10-11
		CALL COLOR	40-45
		CALL HCHAR	20-25
		CALL SCREEN	97-98
		CALL SOUND	17-20
		CALL VCHAR	20-25
		FOR-NEXT	48-57
		GO TO	38-45
		IF-THEN	83-85
		INPUT	33-35
		LET	13-17
		PRINT	8-10
		RANDOMIZE	74-75
		String variables	35-37
		<b>T</b>	
		TAB function	63-66
		Tones	17-20
		Trigonometric functions	140
		<b>V</b>	
		Variables, definition of	13
		Numeric	13-17
		String	35-37
		Volume of Tone	17
		<b>W</b>	
		"Wrap-around" line	55

# Programming BASIC with the TI Home Computer

by Herbert D. Peckham

When you've completed *Beginner's BASIC*, you may want to explore further with the help of an intermediate level book. We recommend *Programming BASIC with the TI Home Computer* by Herbert D. Peckham (McGraw-Hill, 1979).

A well-known author and educator, Mr. Peckham has published numerous books on programming in BASIC. His new book takes you further into the full range and power of TI BASIC and your Texas Instruments computer. The easy-to-understand examples and relaxed style of the book can help you expand your programming skills and develop your own customized computer applications.

Use the coupon below to order *Programming BASIC with the TI Home Computer* from Texas Instruments Incorporated.

## ORDERING INSTRUCTIONS

*Shipping Inside U.S.* Prepaid orders (check or money order) will be sent by postage-paid Third or Fourth Class postage. Allow 4 to 6 weeks for delivery. If you desire shipment other than Third or Fourth Class, additional postage should be included with your order along with specific directions as to method of shipment. Enter additional amount for shipping on order form.

*Shipping Outside U.S.* Orders prepaid in U.S. funds only will be accepted. Specify method of shipment and enter postage or shipping charge amount in proper blank on order form. This amount should be included in book payment.

Please send orders to: Texas Instruments Incorporated  
P. O. Box 3640, M.S. 84M  
Dallas, Texas 75285

To: Texas Instruments Incorporated  
P. O. Box 3640, M.S., 84M  
Dallas, Texas 75285

Please send me *Programming BASIC with the TI Home Computer*.

Name: \_\_\_\_\_

Address: \_\_\_\_\_

City: \_\_\_\_\_ State: \_\_\_\_\_ Zip: \_\_\_\_\_

(LCB-4190) \_\_\_\_\_ copies @ \$19.95 each .....

State and local sales tax (If applicable)\* .....

Shipping method\*\* .....

Additional postage enclosed\*\* .....

Total payment amount enclosed .....

\*State and local sales taxes required by every state except AK, DE, HI, MT, NH, OR.

\*\*For mailing other than Third or Fourth Class.

Price effective October 1, 1982 subject to change without notice.

BB-79



**Beginner's BASIC** is a step-by-step guide that takes you from the "ground up" into an adventure — the adventure of communicating with a computer in a simple, yet powerful language. Quickly and easily you'll find yourself in control of a system that can calculate, make decisions, educate, and entertain. Even if this is the first time you've seen a computer, you'll be able to follow this easy-to-understand, hands-on approach.

**It Starts at the Beginning.** Getting started, do's and don'ts, all of the very first things you need to know. Simple but exciting first experiences with the computer make it fun from the start.

**It Quickly Gives You Confidence.** Although the approach is step-by-step, it's fast and bright. Carefully thought-out useful applications are built into the learning. In a short while you'll know all of the key elements of the BASIC computer language, as well as how to apply it in building powerful programs that carry out exactly what you want your computer to do. On completion of this book, you're ready to explore more advanced techniques explained in your *User's Reference Guide* — or in the many other published materials available on BASIC programming.

**A Learn-by-Doing Approach** that shows you how to

- Command the computer in "Immediate Mode"
- Write simple programs and explore features that let you edit, calculate and make decisions
- Create graphic designs in 16 colors on the computer screen
- Build musical tones, patterns, and chords — in three voices with a range of more than four octaves each
- Develop more advanced programs that "put it all together" — computation, decision making, graphics and sound — to do the things that are most important to you!

**An Important Part of Today's Literacy.** Your Texas Instruments computer, along with this book, offers an excellent framework for learning about the world of computers. You'll know first hand what computers do — and what they can't do. You'll be familiar with the BASIC computer language — one of the most universally applied languages in the world. You'll be a computer "literate" — better able to discuss, understand, and decide on the role of computers in your life. In today's world "computer literacy" can give you an important edge as you encounter computer applications in education, entertainment, and business.