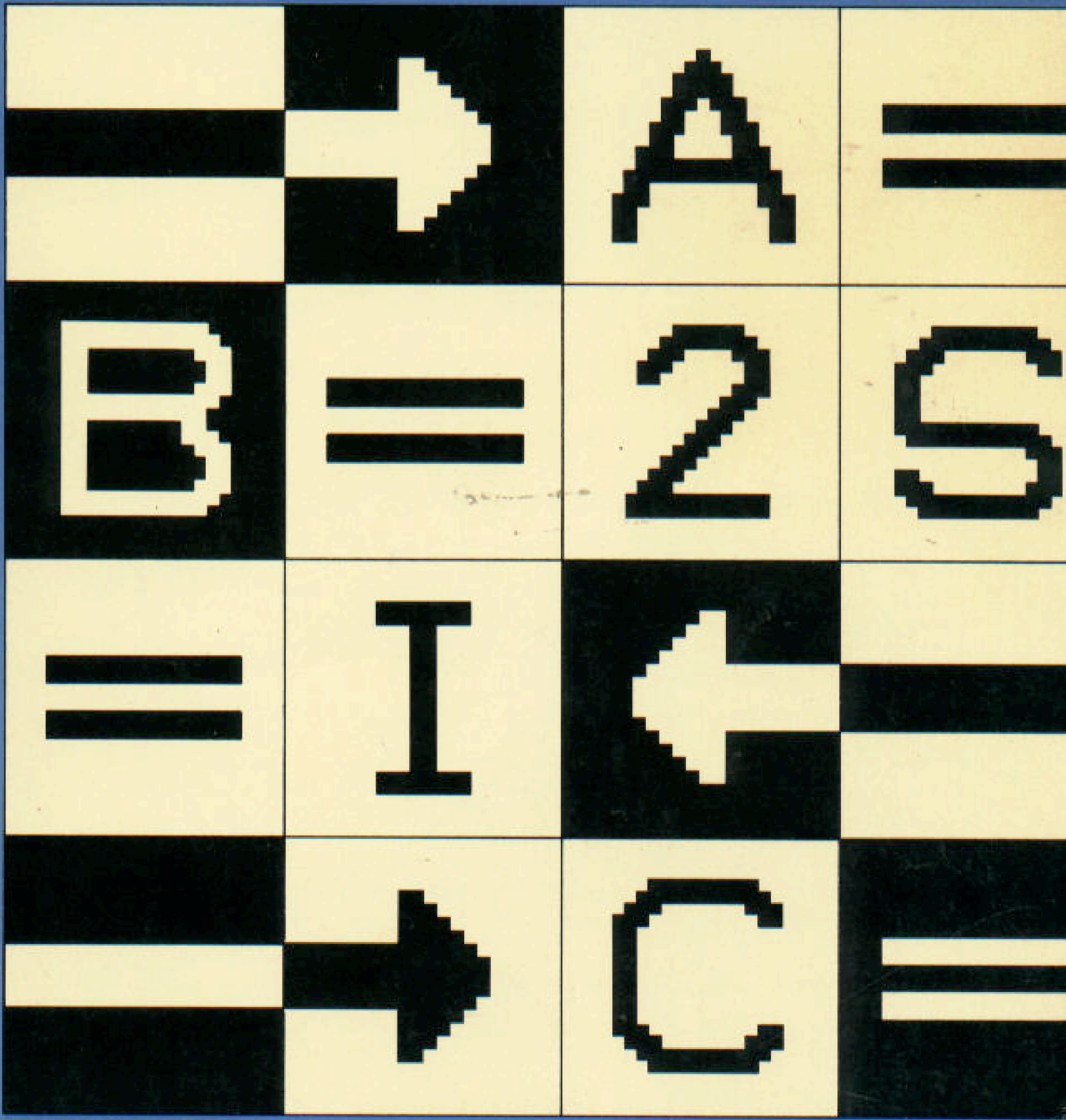
Texas Instruments TI-99/4A Computer

Beginner's BASIC

Step-by-step hands-on approach to learning the fun and power of programming in the TI BASIC language.



IMPORTANT KEYBOARD INFORMATION

The TI-99/4A Computer has a standard typewriter keyboard with both upper-case (large capital) and lower-case (small capital) characters. The **SHIFT** key is used, just as on standard typewriters, to type an upper-case symbol. Except for the letter keys, each key's upper-case (SHIFTed) character is printed above the lower-case character on the key face.

Depressing **ALPHA LOCK** locks all of the alphabetical keys into their upper-case mode. Number and punctuation keys are not affected. Press **ALPHA LOCK** again to return to normal keyboard operation.

TI BASIC, except in specific instances, accepts both upper- and lower-case characters as input. However, when you LIST a program, the screen displays all reserved words, variable names, and subprogram names as large capitals.

The TI-99/4A Computer also has several *function* keys that perform important operations. These are accessed by holding down the **FCTN** key while pressing the appropriate number or letter key. Most of the functions (**AID, CLEAR, QUIT,** etc.) are identified on the strip overlay packed with your computer. Note also that certain symbols are printed on the *front* of several letter keys. These functions, such as $\boxed{\leftarrow}$, $\boxed{\rightarrow}$, $\boxed{\uparrow}$, and $\boxed{\downarrow}$, and some punctuation symbols are also accessed by holding down the **FCTN** key while pressing the symbol key. The following chart identifies these functions.

FUNCTION KEYS

Name	*Press*	*Name*	*Press*
AID	**FCTN 7**	Left arrow	**FCTN S**
CLEAR	**FCTN 4**	Right arrow	**FCTN D**
DELete	**FCTN 1**	Down arrow	**FCTN X**
INSert	**FCTN 2**	Up arrow	**FCTN E**
QUIT	**FCTN =**	**PROC'D**	**FCTN 6**
REDO	**FCTN 8**	**BEGIN**	**FCTN 5**
ERASE	**FCTN 3**	**BACK**	**FCTN 9**

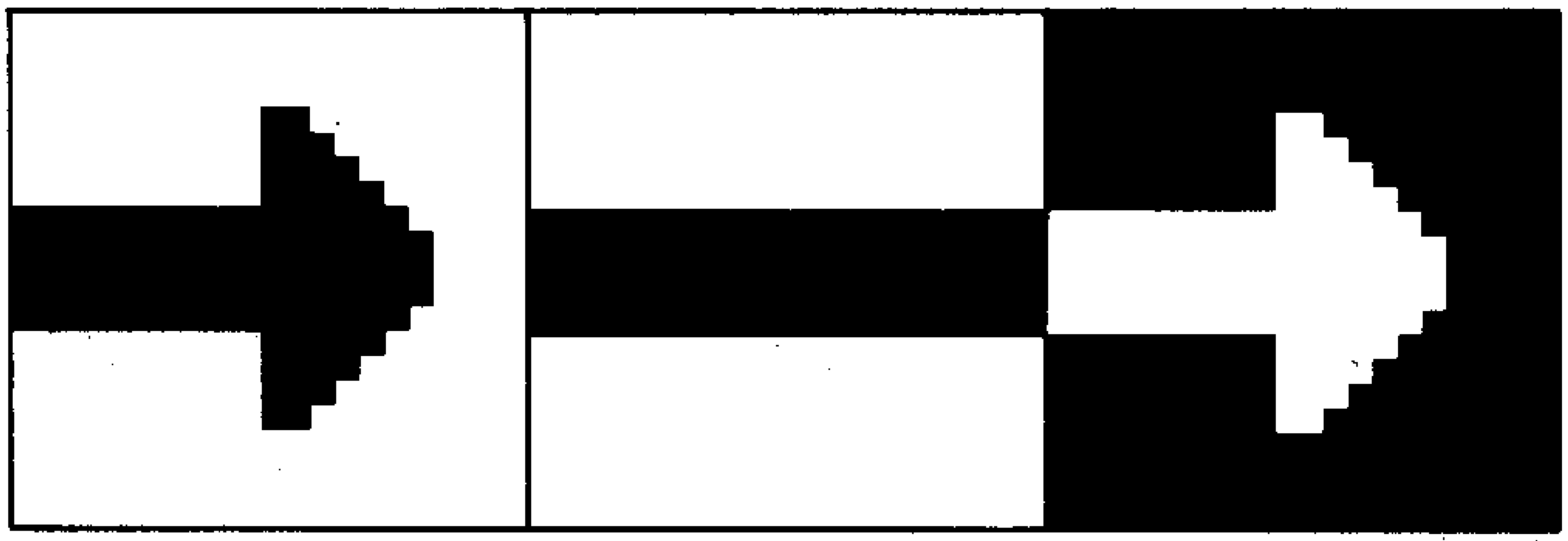
IMPORTANT NOTICE REGARDING PROGRAMS AND BOOK MATERIALS

LCB 4180

Beginner's BASIC

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.

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.



Note: The instructions and sample programs in this book are designed for use with the Texas Instruments TI-99/4A Computer. The information included will be generally useful with other computers incorporating BASIC programming language conforming to the American National Standard for Minimal BASIC. However, the program instructions included here – especially those for graphics and sound – will apply specifically to the TI computer.

This book was developed by

Don Inman
Ramon Zamora
Bob Albrecht

in cooperation with Texas Instruments Incorporated and the Staff of the Texas Instruments Learning Center:

Jacquelyn Quiram
Bob O'Dell

Artwork and layout were coordinated and executed by

Schenck Design Associates, Inc.

ISBN 0-89512-028-3
Library of Congress Catalog Number: 79-65510

Table of Contents

INTRODUCTION *Page*

You and Computer Programming 5
TI BASIC 5
About this Book 6

CHAPTER 1: THE IMMEDIATE MODE

The PRINT Statement 8
The CALL CLEAR Statement 10
Error Messages 11
Error Correction 12
The LET Statement 13
The CALL SOUND Statement 17
Graphics (CALL VCHAR and CALL HCHAR) 20
Summary 25

CHAPTER 2: SIMPLE PROGRAMMING

A Printing Program 26
Program Structure 27
Commands – NEW, RUN, LIST 28
A Numerical Program 29
Editing the Program 31
- Adding Program Lines 31
- Removing Program Lines 32
- The INPUT Statement 33
- String Variables 35

The GO TO Statement 38
A GO TO Loop with CALL SOUND 39
A GO TO Loop with CALL COLOR 40
Error Messages 45
Summary 47

CHAPTER 3: MORE PROGRAMMING POWER

The FOR-NEXT Statement 48
- A GO TO Loop 49
- A FOR-NEXT Loop 49

CALL COLOR with a FOR-NEXT Loop 52
"Nested" FOR-NEXT Loops 53
- Error Conditions with FOR-NEXT 56

Plain and Fancy Printing 57
- Spacing with Commas 57
- Spacing with Semicolons 60
- Spacing with Colons 62
- The TAB Function 63

Arithmetic Power 66
- Order of Operations 67
- Using Parentheses 67
- Scientific Notation 69

The INT Function 69
Summary 72

CHAPTER 4: FUN AND SIMULATIONS *Page*

The RND Function 73
The RANDOMIZE Statement 74
Other Random Number Ranges 75
A Two-Dice Simulation 78
Error Conditions with RND 80
Randomized Character Placement 80
The IF-THEN Statement 83
Error Conditions with IF-THEN 85
Games and Music 85
A Number-Guessing Program 86
A Tone-Guessing Program 88
Color Up! 89
Random Notes 91
A Musical Interlude 92
The CALL KEY Routine 93
Summary 94

CHAPTER 5: COMPUTER GRAPHICS

Blocks of Color 96
The CALL SCREEN Statement 97
Patterns 98
Rectangles and Squares 98
"Holes" 103
Animation 104
Flashing Letters 104
Flashing Color Squares 105
Moving Color Squares 107
The CALL CHAR Statement 108
A Block Figure with CALL CHAR 114
Mr. Bojangles 118
The Giant 121
Summary 123

Appendix A: Musical Tone Frequencies 124
Appendix B: Character Codes 125
Appendix C: Color Codes 126
Appendix D: Mathematical Operations 127
Index 142

Introduction

You and Computer Programming

This book is your guide to an adventure — learning to program your Texas Instruments computer. Even if you have never worked with a computer before, you can use this book to teach yourself, your family, and your friends how to use and enjoy your computer.

Computers are rapidly becoming an everyday part of our lives. We're very familiar with bank statements compiled and printed by computers; we watch computerized scoreboards at sports events — we even see computer-aided instruction in our children's classrooms! Almost everything we eat, wear, and use has been handled at some point in the manufacturing process by computer-controlled machinery.

By learning to program and interact with your computer, you'll be joining this technological revolution of the Computer Age. You'll understand more about how computers work, what they can (and can't) do, and why they are becoming so widely used. Best of all, you'll be able to apply the power of your computer to the areas that appeal to you — your business and finance, your hobbies, your family's needs and interests.

And perhaps we'd better warn you — many people become fascinated with computer programming as an exciting and entertaining hobby! So don't be surprised when you — and all the family — find yourselves wanting to learn about and use your computer more and more as time goes on.

What is computer programming? Nothing mysterious! Programming is simply communicating with a computer — telling it what to do and when to do it. To program your computer you'll only need to learn two things: the language your computer understands, and the way you talk to it. No lengthy training periods or super-sophisticated skills are required.

TI BASIC

In order to communicate with any computer, you'll need to learn its language. The language we'll be exploring here is a form of BASIC (short for *B*eginners *A*ll-purpose *S*ymbolic *I*nstruction *C*ode). BASIC was developed by John Kemeny and Thomas Kurtz at Dartmouth College during the middle 1960's. Although BASIC is only one of many computer languages, it is one of the most popular in use today. It's easy to learn and simple to use; yet it's powerful enough to do almost anything you'd want to do with computers.

Some BASIC features may vary slightly from one type of computer to another. The similarities, however, far outweigh the differences; in fact, you can think of these different forms of BASIC as dialects of the same language. The dialect used by the TI computer is called TI BASIC.

As you read this book and try out the examples on your own computer, you'll notice one striking fact about BASIC: it's very much like English! You'll see words like PRINT, GO TO, RUN, and END. The meanings of these words in BASIC are almost identical to the definitions you already know and understand. This fact is what makes BASIC so easy to learn and fun to use.

Introduction

You and Computer Programming

This book is your guide to an adventure – learning to program your Texas Instruments computer. Even if you have never worked with a computer before, you can use this book to teach yourself, your family, and your friends how to use and enjoy your computer.

Computers are rapidly becoming an everyday part of our lives. We're very familiar with bank statements compiled and printed by computers; we watch computerized scoreboards at sports events – we even see computer-aided instruction in our children's classrooms! Almost everything we eat, wear, and use has been handled at some point in the manufacturing process by computer-controlled machinery.

By learning to program and interact with your computer, you'll be joining this technological revolution of the Computer Age. You'll understand more about how computers work, what they can (and can't) do, and why they are becoming so widely used. Best of all, you'll be able to apply the power of your computer to the areas that appeal to you – your business and finance, your hobbies, your family's needs and interests.

And perhaps we'd better warn you – many people become fascinated with computer programming as an exciting and entertaining hobby! So don't be surprised when you – and all the family – find yourselves wanting to learn about and use your computer more and more as time goes on.

What is computer programming? Nothing mysterious! Programming is simply communicating with a computer – telling it what to do and when to do it. To program your computer you'll only need to learn two things: the language your computer understands, and the way you talk to it. No lengthy training periods or super-sophisticated skills are required.

TI-BASIC

In order to communicate with any computer, you'll need to learn its language. The language we'll be exploring here is a form of BASIC (short for *B*eginners *A*ll-purpose *S*ymbolic *I*nstruction *C*ode). BASIC was developed by John Kemeny and Thomas Kurtz at Dartmouth College during the middle 1960's. Although BASIC is only one of many computer languages, it is one of the most popular in use today. It's easy to learn and simple to use; yet it's powerful enough to do almost anything you'd want to do with computers.

Some BASIC features may vary slightly from one type of computer to another. The similarities, however, far outweigh the differences; in fact, you can think of these different forms of BASIC as dialects of the same language. The dialect used by the TI computer is called TI BASIC.

As you read this book and try out the examples on your own computer, you'll notice one striking fact about BASIC: it's very much like English! You'll see words like PRINT, GO TO, RUN, and END. The meanings of these words in BASIC are almost identical to the definitions you already know and understand. This fact is what makes BASIC so easy to learn and fun to use.

Introduction

Now, how do you talk to the computer? Well, take a look at the keyboard. You see there the letters of the alphabet, numbers, punctuation marks, and other special symbols, many of which you've seen before on typewriter keyboards. Everything you'll need to use to communicate with your computer is right there on the keyboard. You "type" your instructions, and the computer "hears" them. It's essentially as easy as that!

About This Book

This book will guide you step by step through the process of learning TI BASIC. While the book is not a complete textbook on BASIC programming, the material included here will give you a good foundation for the continued development of your programming skills. (Once you are familiar and comfortable with BASIC, you'll be ready for the more advanced material found in the *User's Reference Guide.*) Throughout the book, each explanation of a statement or command is followed by one or more examples for you to try out. Also, you can – and should – experiment with other examples of your own, to help you become thoroughly acquainted with the capabilities of your computer. You'll find some special sections marked **EXPERIMENT!** throughout the book. These are for you to try out on your own.

In the first chapter we'll explore some of the BASIC statements that can be performed in what's called the *Immediate Mode* (that is, directly from the keyboard). Do you want to add 3 and 5, or create sounds, or make designs (computer graphics) on the screen? You can do all of these in the Immediate Mode, and you'll find out how to do them in Chapter 1.

Chapters 2 and 3 take you on into programming. You'll learn how to "structure" a program, issue "commands" your computer can follow, perform mathematics, use graphics and sound more effectively, and create loops (program segments that repeat themselves).

Then, in Chapters 4 and 5, you'll get further into some of the exciting things a computer can do. Did you know that your computer can play games? Make music? Draw colorful designs on the television screen? It can, and you can teach it how!

At the end of the book are several appendices of reference information you can use as you develop your own programs. Of special interest to those who want and need to use the computer as a powerful computational device, *Appendix D* outlines the mathematical operations and functions of the computer. You'll also find a convenient alphabetized index of topics to help you look up features you want to review.

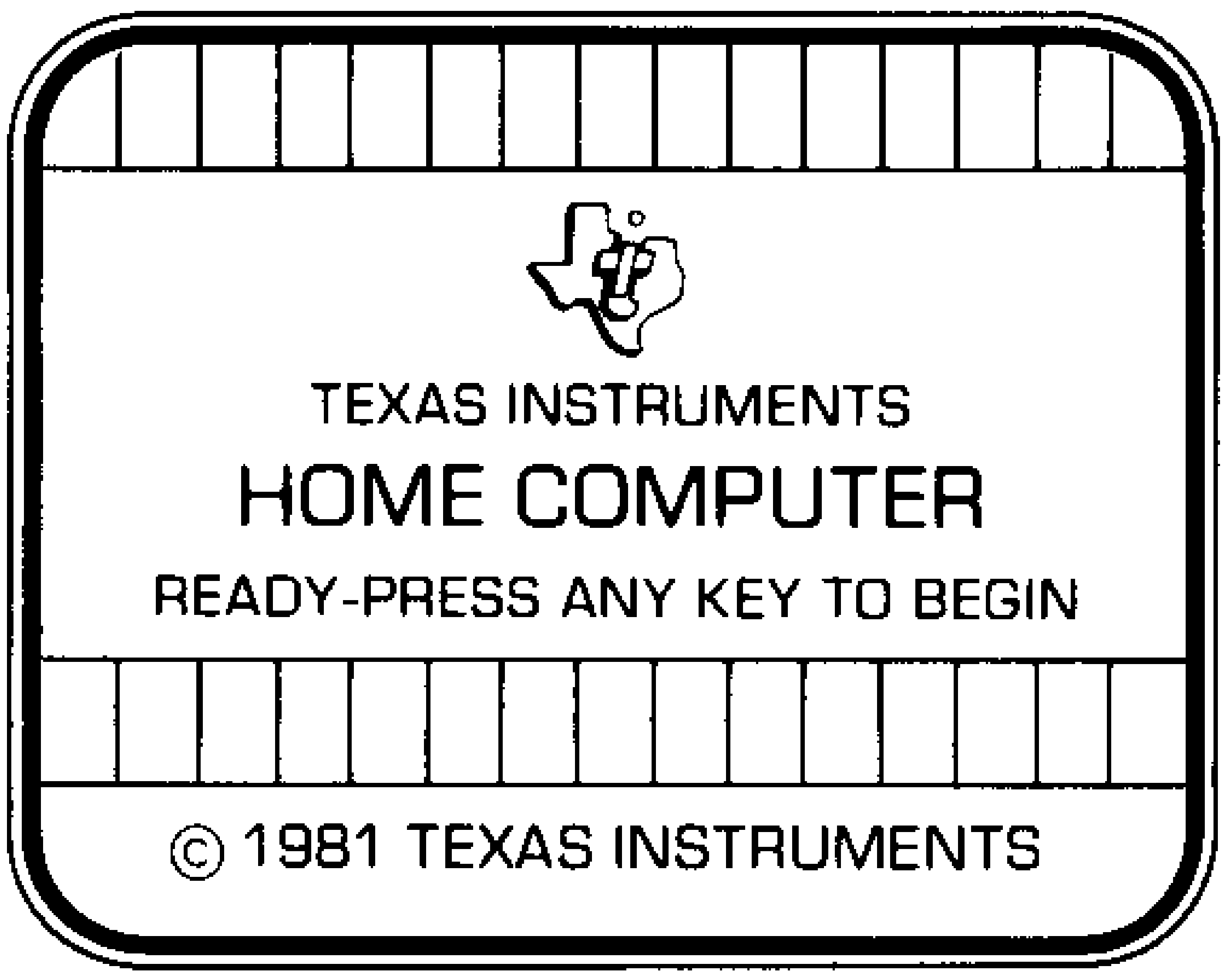
Now that you know what's ahead, let's waste no more words – let's get started in the Immediate Mode.

1 CHAPTER ONE — The Immediate Mode

In its Immediate Mode, your computer "immediately" performs each BASIC statement you've typed in, as soon as you press **ENTER**. Because you can see an instant response on the screen, the Immediate Mode is a good way to introduce and explore certain BASIC language statements.

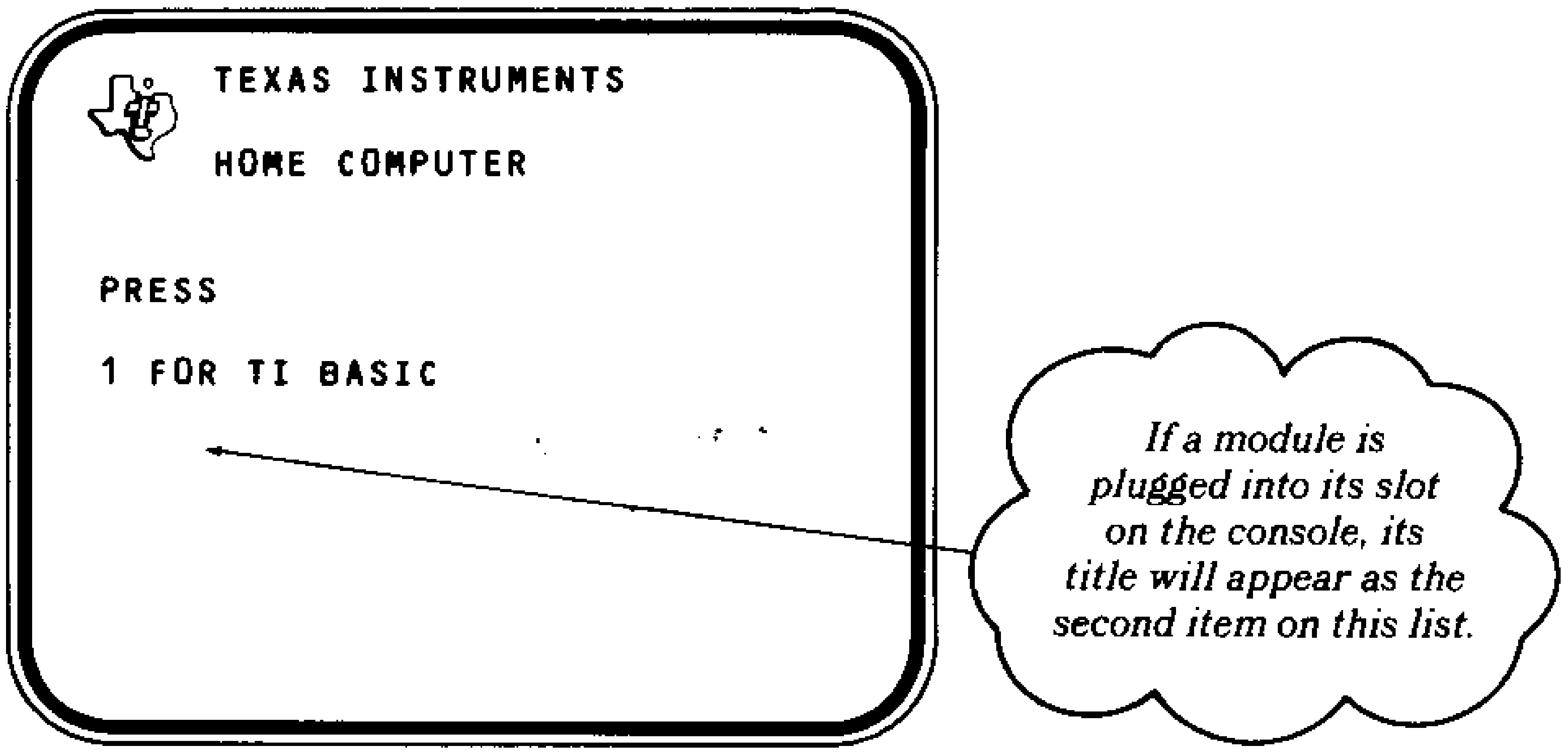
Before you begin learning BASIC, take a few minutes to review the operation of the keyboard. You'll find a complete "key tour" in the *User's Reference Guide.*

When you are ready, turn on your computer. The display screen should look like this:



```
TEXAS INSTRUMENTS
HOME COMPUTER
READY-PRESS ANY KEY TO BEGIN

© 1981 TEXAS INSTRUMENTS
```

Press any key on the keyboard. The display will then show the master selection list.



```
TEXAS INSTRUMENTS
HOME COMPUTER

PRESS

1 FOR TI BASIC
```

If a module is plugged into its slot on the console, its title will appear as the second item on this list.

(*Note:* When you're ready to leave TI BASIC, just type the word **BYE** and press the **ENTER** key. The computer will then return to the main title screen.)

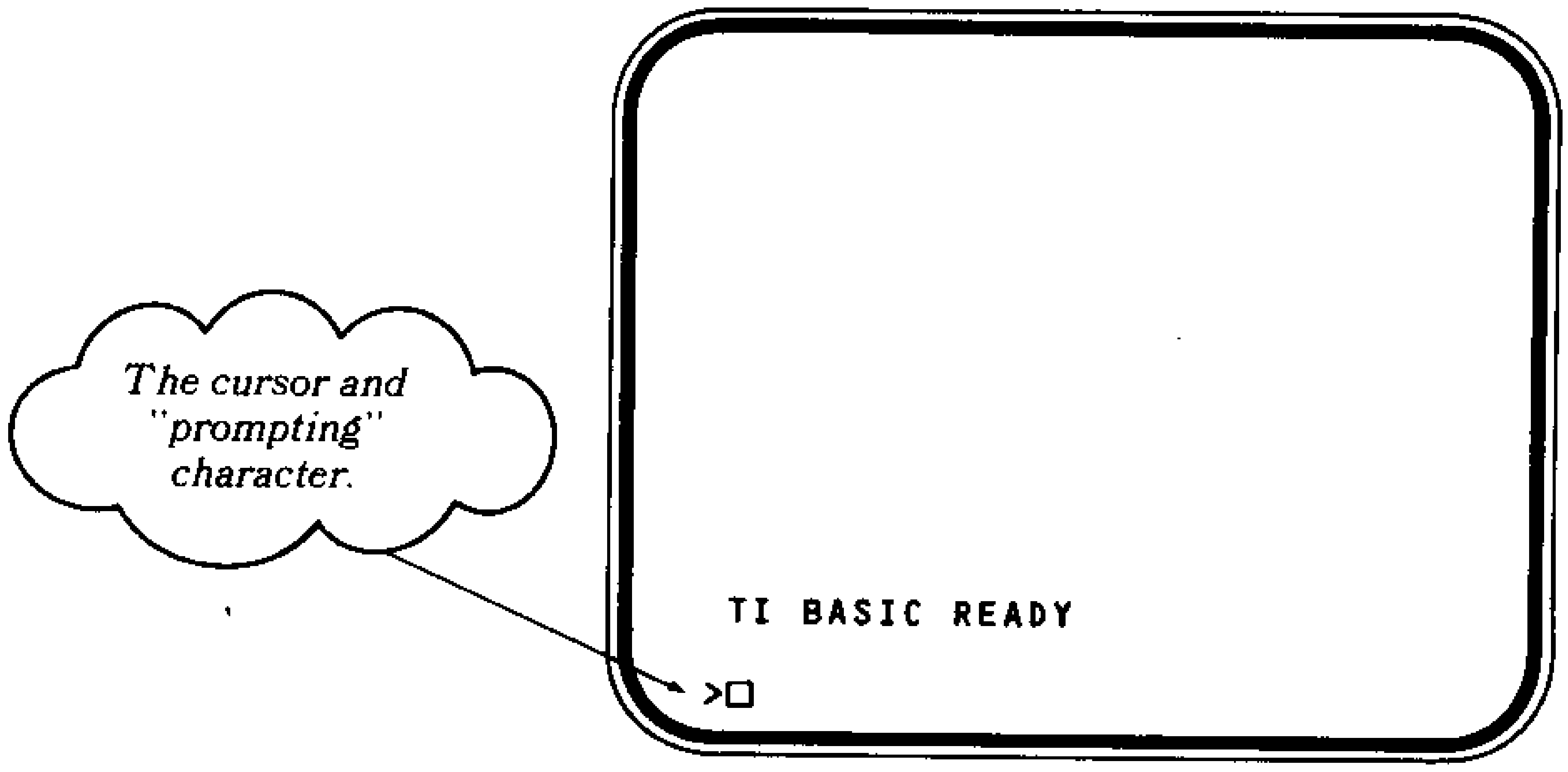
The examples shown in this book are printed in upper-case (large capital) letters. If you want to reproduce the examples exactly as you see them here, press down the **ALPHA LOCK** key. However, in most cases the computer accepts either upper-case or lower-case letters.

Also, see *Important Keyboard Information* on the inside front cover of this book for details about the function keys (**CLEAR,** *left-arrow* key, *right-arrow* key, etc.).

1

Press the **1** key to select TI BASIC.

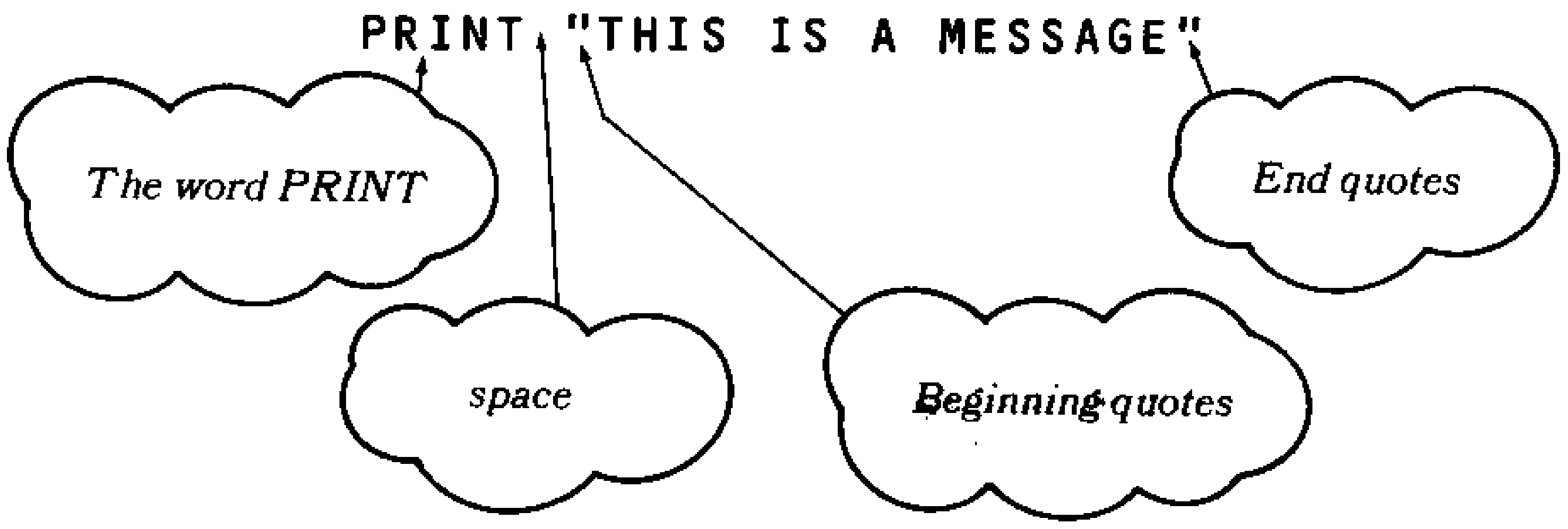
The display now shows that the computer is ready for you to begin.



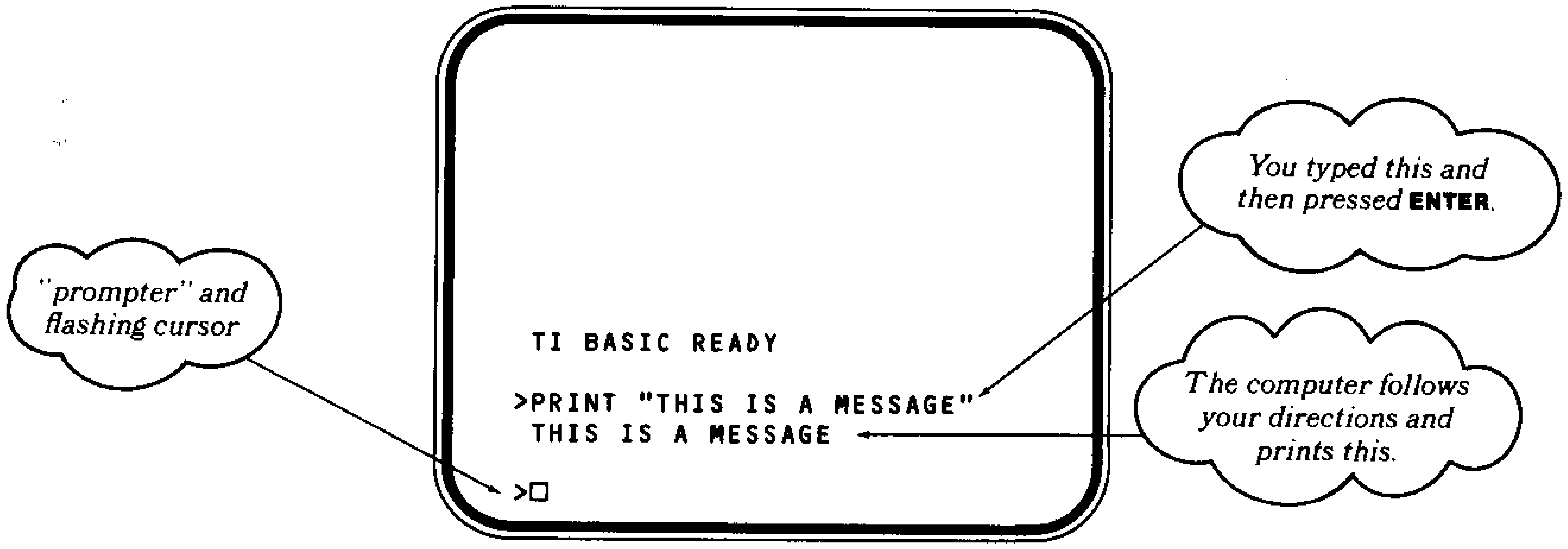
The flashing rectangle is called the *cursor*. It tells you that the computer is ready for you to use. Whenever you see the cursor, you know that it's your turn to do something. The prompting symbol marks the beginning of each line you type.

The PRINT Statement

The PRINT statement means exactly what it says. You merely type the word PRINT, followed by a message enclosed in quotation marks, and the computer prints the message when you press **ENTER**.



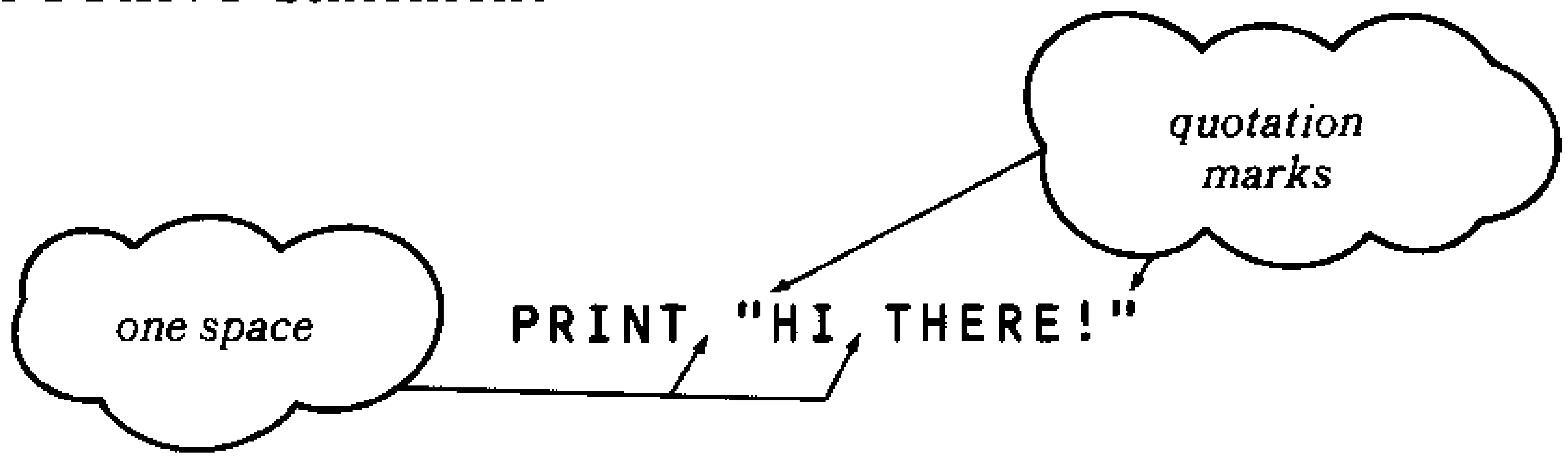
Remember to press the **ENTER** key after the ending quotation marks! This is the computer's cue to perform what you have requested.


You typed this and then pressed ENTER.
"prompter" and flashing cursor
The computer follows your directions and prints this.

```
TI BASIC READY

>PRINT "THIS IS A MESSAGE"
 THIS IS A MESSAGE

>□
```

Let's try another PRINT statement.

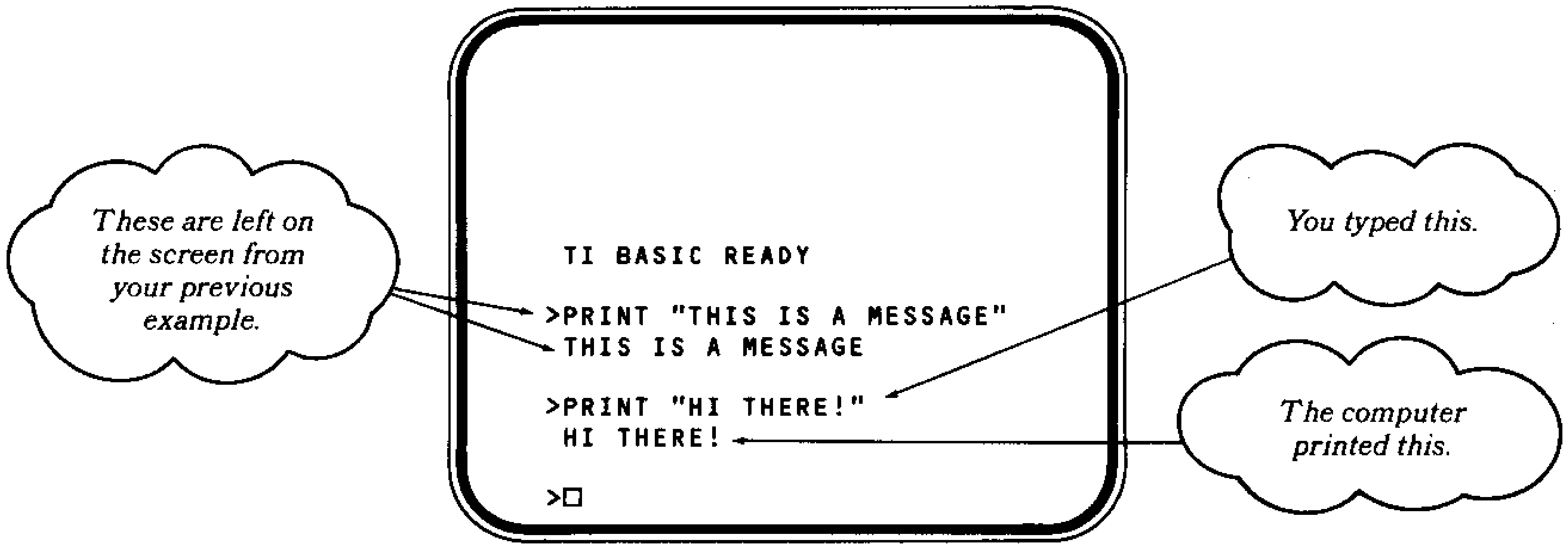
Type this:



```
PRINT "HI THERE!"
```

(*Note:* If you accidentally press a wrong letter or symbol key, just use the *left arrow* key to move the cursor back to the incorrect symbol. Then retype.)

Now press **ENTER**, and the computer will do just what you told it to do:

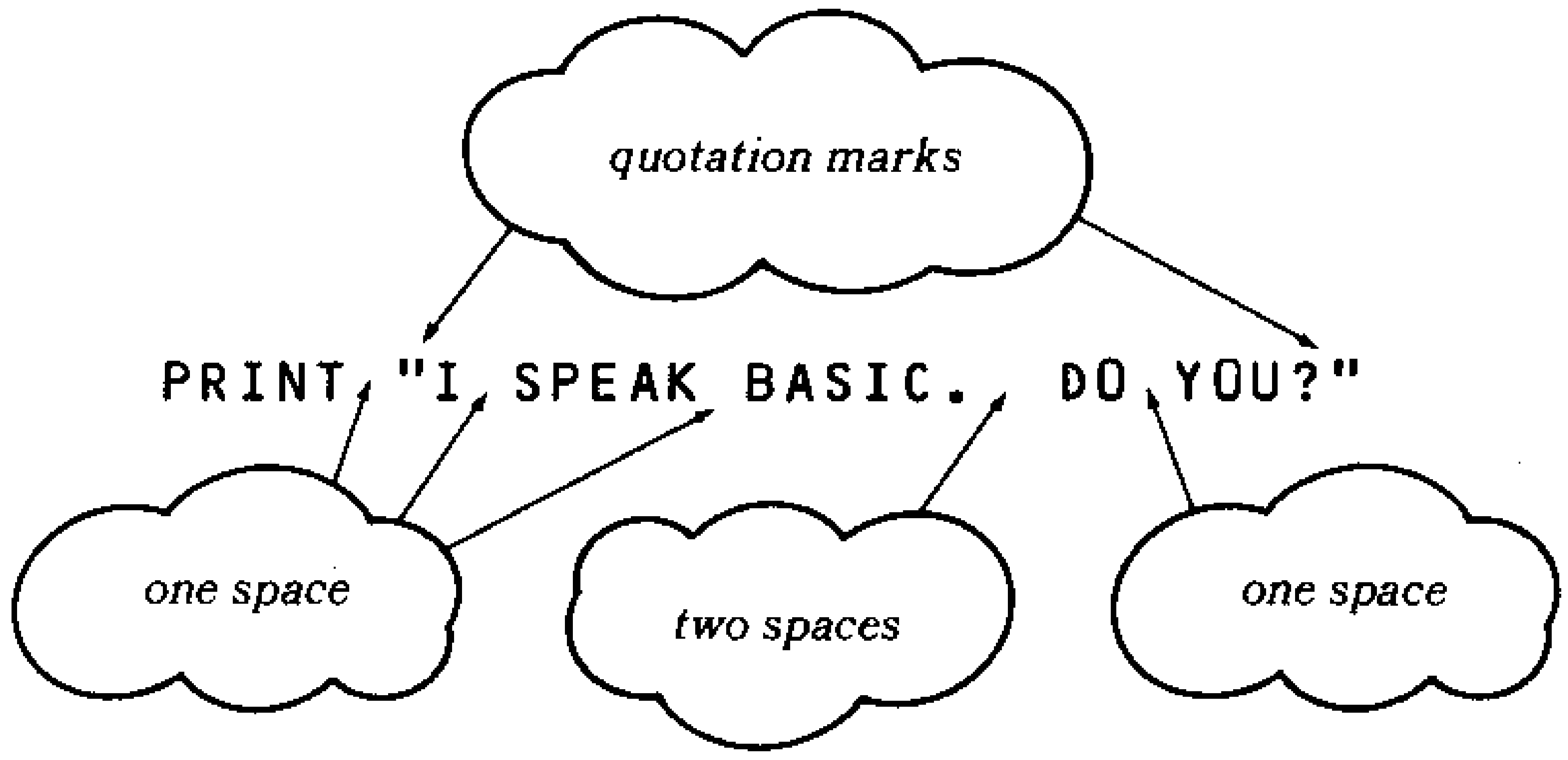


```
TI BASIC READY

>PRINT "THIS IS A MESSAGE"
THIS IS A MESSAGE

>PRINT "HI THERE!"
 HI THERE!

>□
```

These are left on the screen from your previous example.
You typed this.
The computer printed this.

Did you notice the way the lines moved up on the screen when you pressed **ENTER** and again when the computer finished printing its line? This procedure is called *scrolling*. The cursor tells you it's your turn now and shows you where the next line will begin.

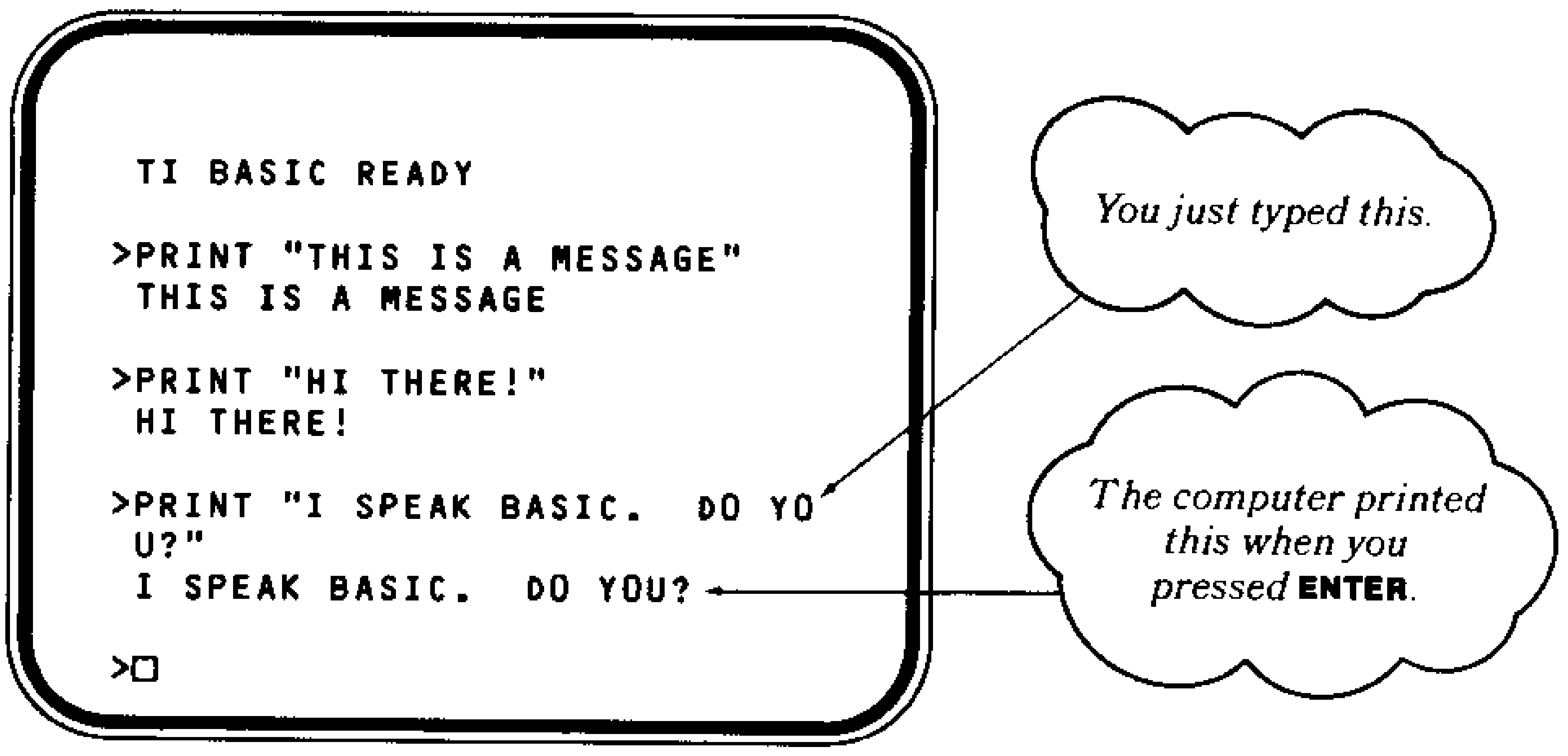
Let's try another example. Type these words, but don't press **ENTER** just yet:



```
PRINT "I SPEAK BASIC.  DO YOU?"
```

(When you run out of room on a line, just keep typing — the computer will automatically "scroll" to the next line.)

Now, look at the screen and check what you've typed. If there are any errors, just use the *left arrow* key until the cursor has reached the error. Then retype the line correctly from that point on. (This is only one correction procedure – you'll learn others as you go along in the book.) When everything is correct, press the **ENTER** key. You'll then see:

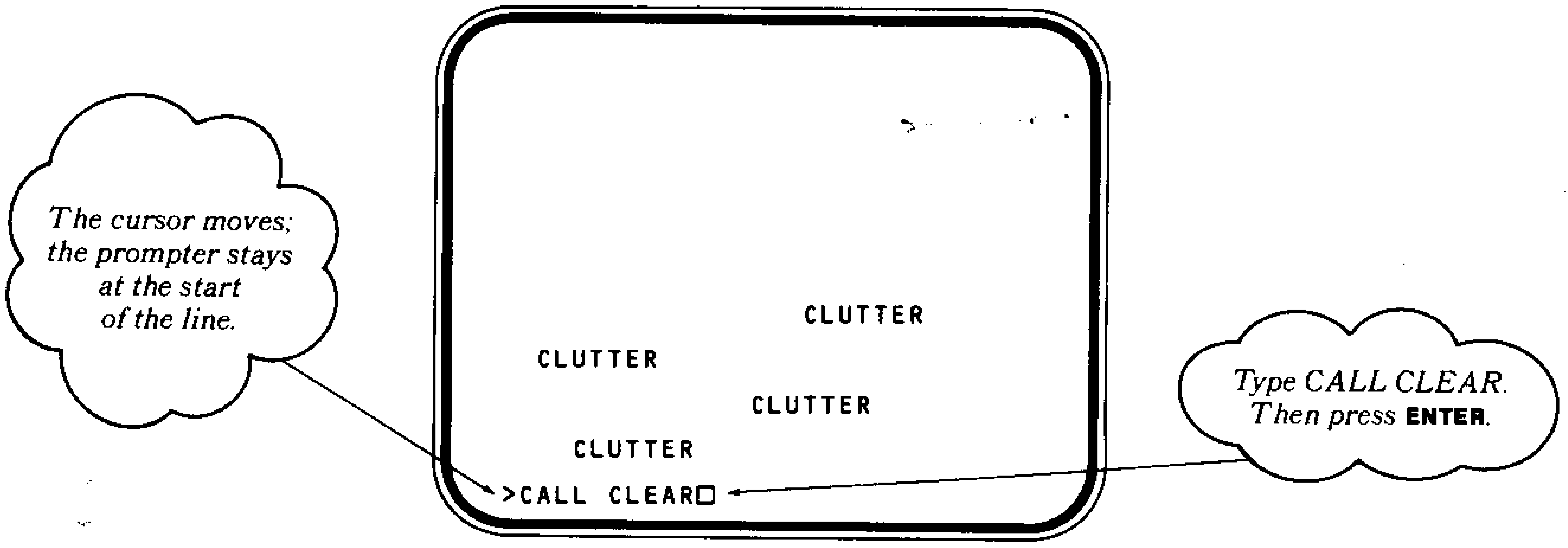


```
TI BASIC READY

>PRINT "THIS IS A MESSAGE"
 THIS IS A MESSAGE

>PRINT "HI THERE!"
 HI THERE!

>PRINT "I SPEAK BASIC.  DO YO
 U?"
 I SPEAK BASIC.  DO YOU?

>□
```

If you want to try some other PRINT statements on your own, go right ahead. Each time you press **ENTER**, you'll see the lines on the screen scroll upwards. The top lines will finally begin to disappear as the screen's capacity (24 lines) is reached.

The CALL CLEAR Statement

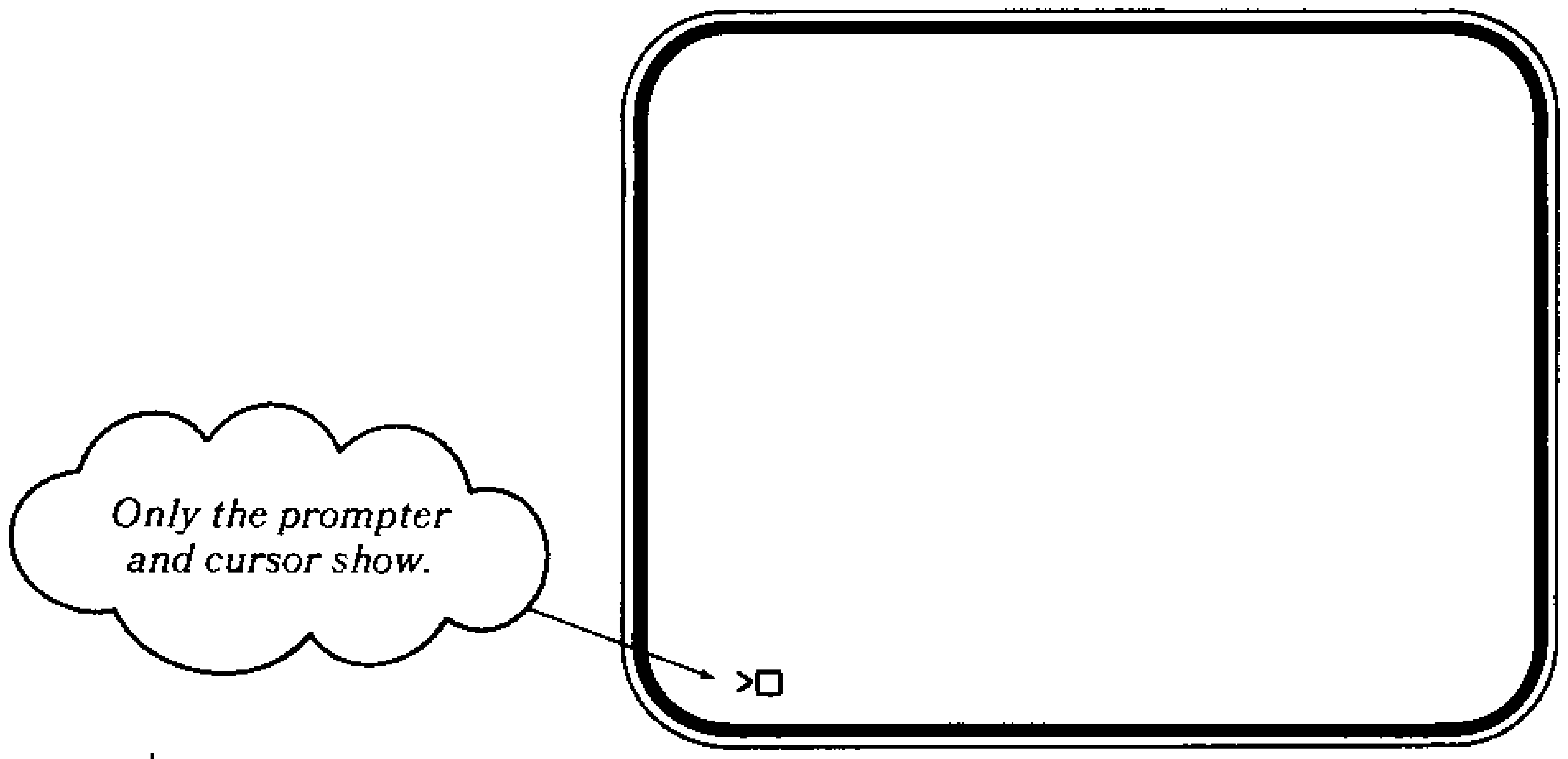
You've probably noticed that your video display has begun to look rather cluttered. If you want to clear the screen for a less distracting appearance, you can use the words CALL CLEAR.



```
                    CLUTTER
   CLUTTER
                 CLUTTER
     CLUTTER
>CALL CLEAR□
```

The cursor moves; the prompter stays at the start of the line.
Type CALL CLEAR. Then press **ENTER**.

CALL CLEAR wipes the slate clean for your next request, and your display will look like this:



Note: As you work through this book, you'll see several BASIC statements that begin with the word CALL. Your computer has been "taught" to do certain things by having some special-purpose programs built into it, and a CALL statement tells the computer to "call" the built-in program named in the statement.

Error Messages

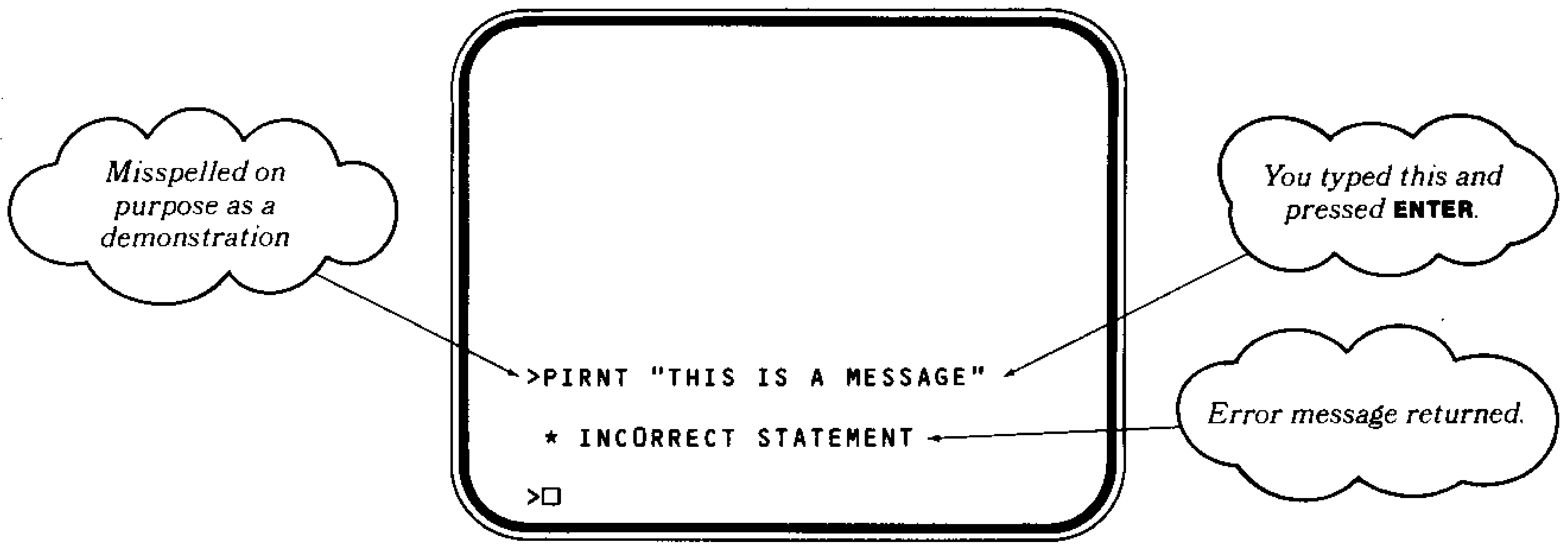
Every computer programmer makes mistakes, so don't hesitate to try experiments of your own as you go through the examples in this book. Errors will not hurt the computer. It quickly recognizes things it can't do and gives you an error message and a tone to tell you to try again. When mistakes happen, just identify the error and retype the instruction correctly.

Some of the most common errors are typing a wrong letter and omitting a necessary part of the statement. For example, here are a few mistakes your computer doesn't like in a PRINT statement:

1. A misspelling in the word PRINT.
2. A missing or extra quotation mark.
3. Extra spaces in the word PRINT.

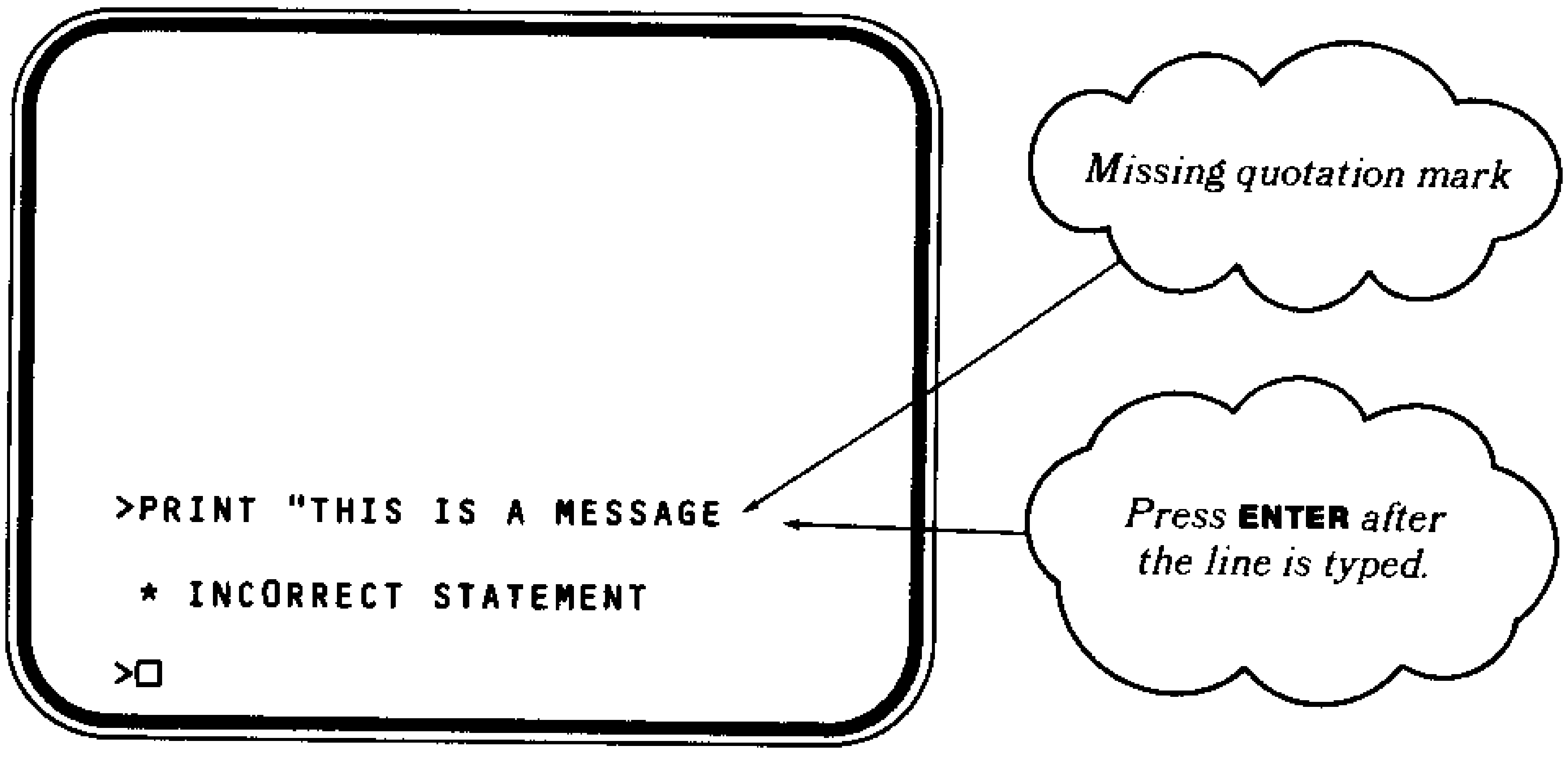
Let's experiment with some intentional errors to become more comfortable with error messages.

(1) Misspelling in the word PRINT


Misspelled on purpose as a demonstration
You typed this and pressed **ENTER**.
Error message returned.

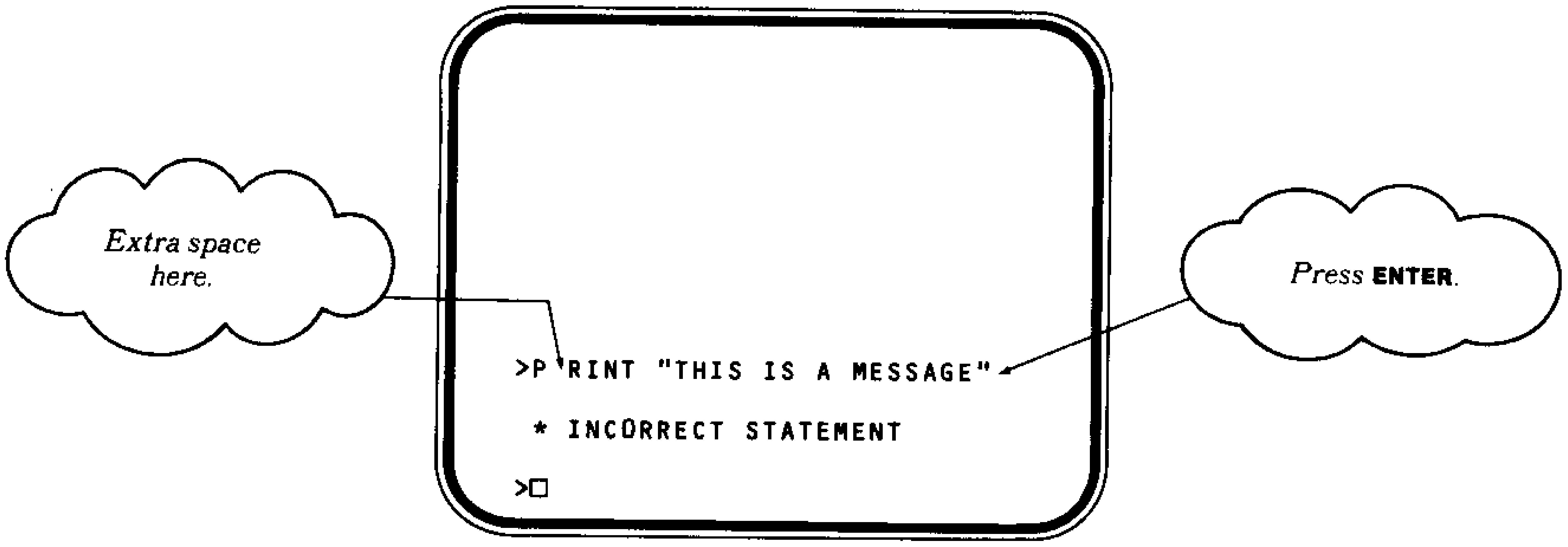
```
>PIRNT "THIS IS A MESSAGE"

 * INCORRECT STATEMENT

>□
```

(2) Missing or extra quotation marks



```
>PRINT "THIS IS A MESSAGE

 * INCORRECT STATEMENT

>□
```

(3) Extra spaces in the word PRINT



```
>P RINT "THIS IS A MESSAGE"

 * INCORRECT STATEMENT

>□
```

Experiment!

Try a few more messages with the PRINT statement, introducing intentional errors so that you will become familiar with the error messages. (We'll discuss other error messages at appropriate places throughout the book.).

Error Correction

There are several ways to correct typographical errors *before* you have pressed **ENTER**.

1. You can press **ERASE** to erase what you've typed on the line.
2. If you spot the mistake just after you've made it, use the *left arrow* key to move the cursor back to the error, retype the line from that point on, and then press **ENTER.** (Note that the characters are not erased as you backspace over them.)

3. If you've finished typing a line and you find a mistake near the beginning of the line, use the *left arrow* key as above, retype the letter or word, use the *right arrow* key to move the cursor back to the end of the line, and then press **ENTER**. Note that the *right arrow* key *does not* erase as it moves the cursor. If you need to erase a character or word, use the **SPACE BAR** to advance the cursor over the character.

OR

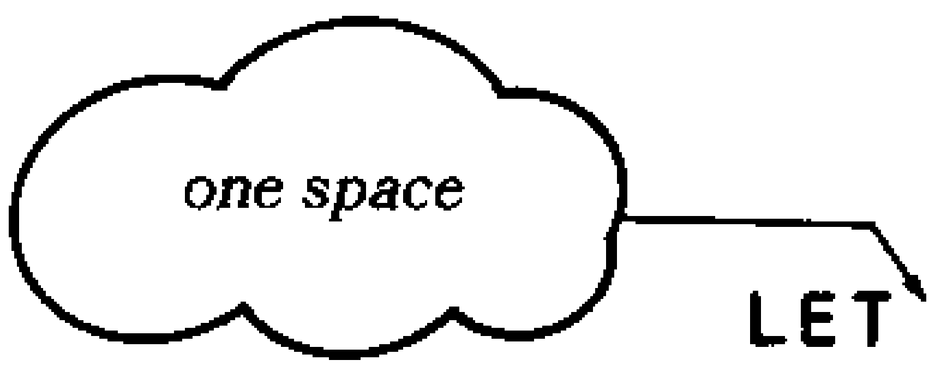
You can just disregard the error and press **ENTER** anyway. The computer may give you an error message, but it's very forgiving. Simply retype your line – correctly, this time – and press **ENTER** again.

The LET Statement

The LET statement is used to assign a value to a *variable.* Variables are "names" given to numbers or to phrases containing both numbers and letters (and certain other characters). Although there are two types of variables, in this section we'll consider only those variables that give names to numbers. These are called numeric variables. A numeric variable is just a name given to a numeric value.

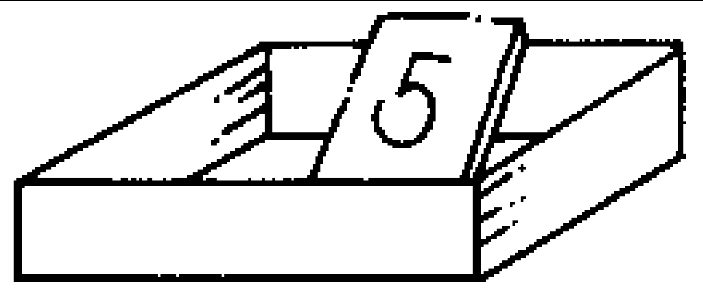
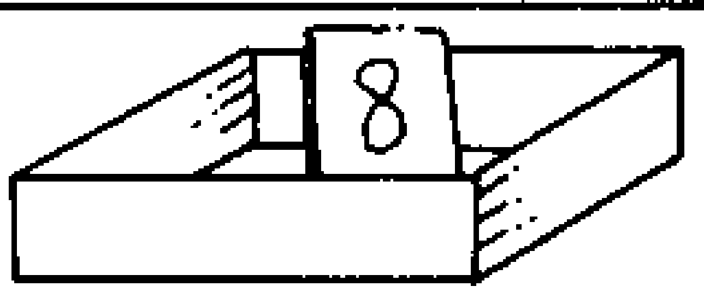
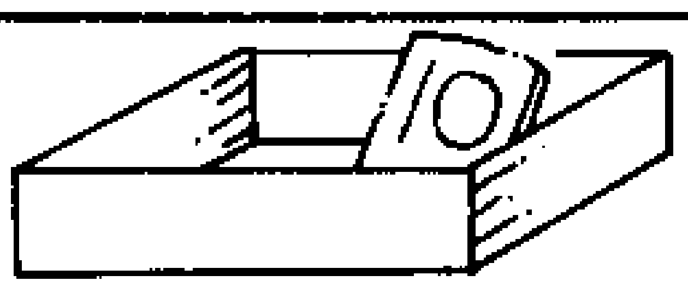
In the LET statement the word LET is followed by the variable (the name), then an equals sign, and finally the numeric value you're assigning to the variable. Variables can be up to 15 characters long, but they are generally kept fairly short for convenience.

Let's try a few examples. Type in the following lines, pressing **ENTER** at the end of each line:

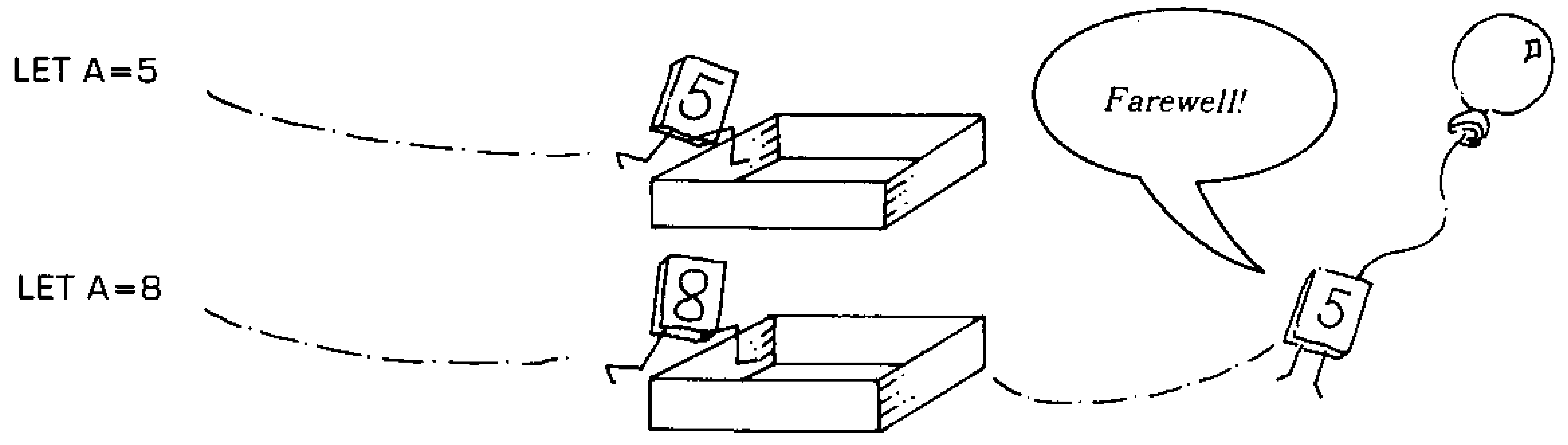


```
LET A=5
LET A2=8
LET ALPHA=10
```

You can think of variables as labeled boxes that hold assigned values.

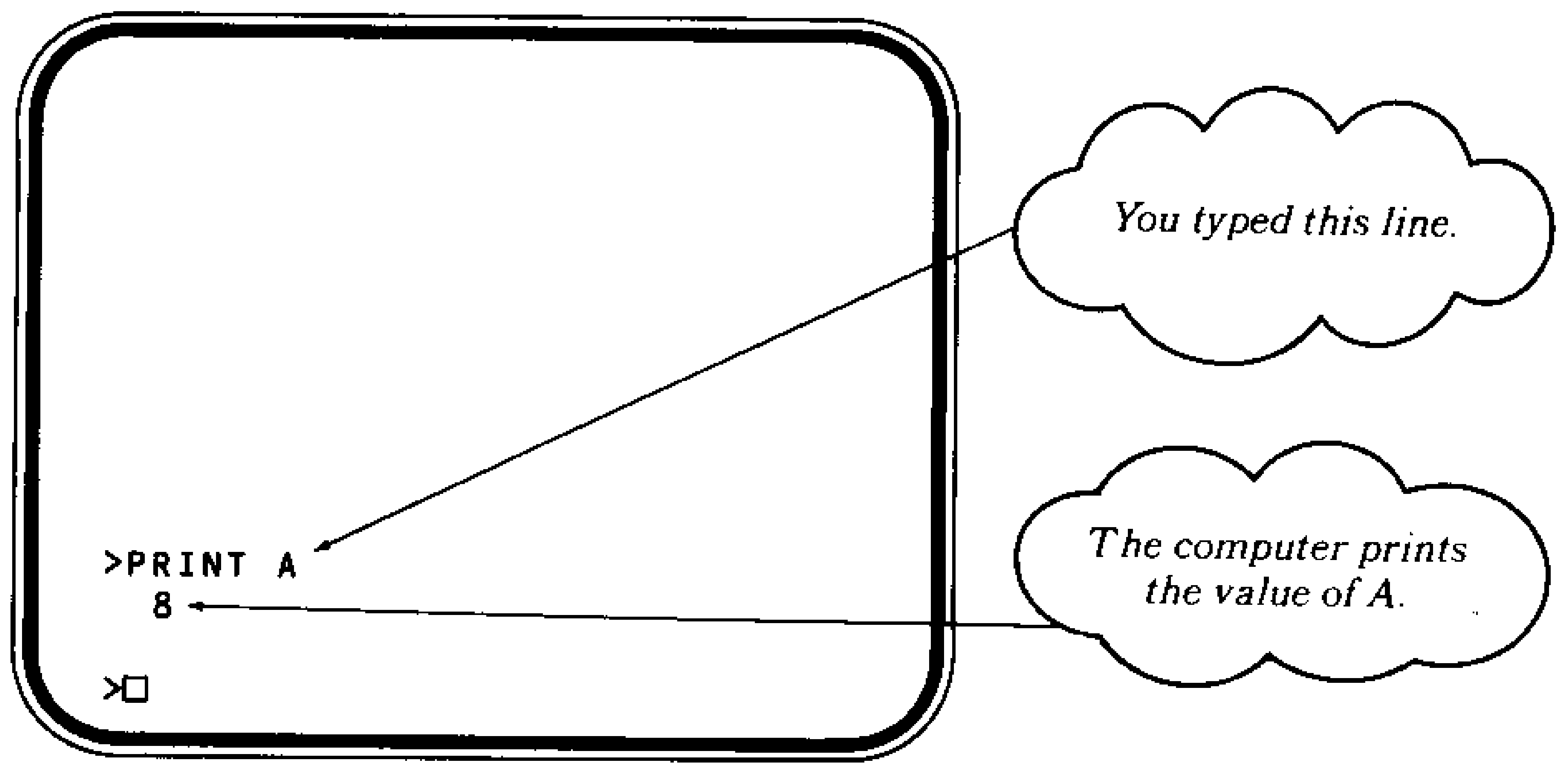
LET A=5	 A=5
LET A2=8	 A2=8
LET ALPHA=10	 ALPHA=10

Only one value at a time may be assigned to a given variable, but you can change a value easily. Type these successive LET statements, pressing **ENTER** after each line.



The value of A is no longer 5. The 5 has been replaced by the value 8.

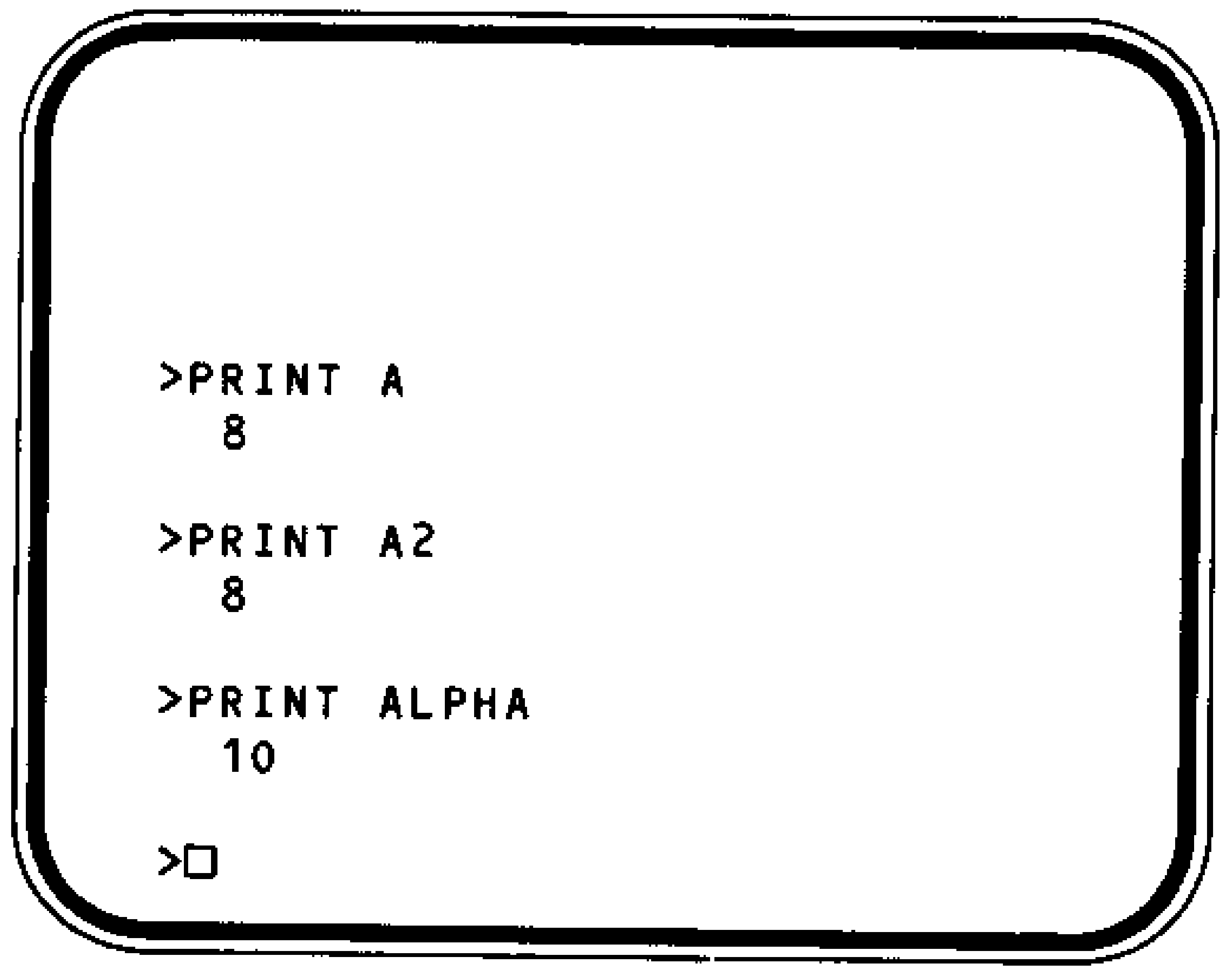
Now let's use PRINT statements to check the values we've entered. Clear the screen; then type PRINT A and press **ENTER**.



```
>PRINT A
  8

>□
```

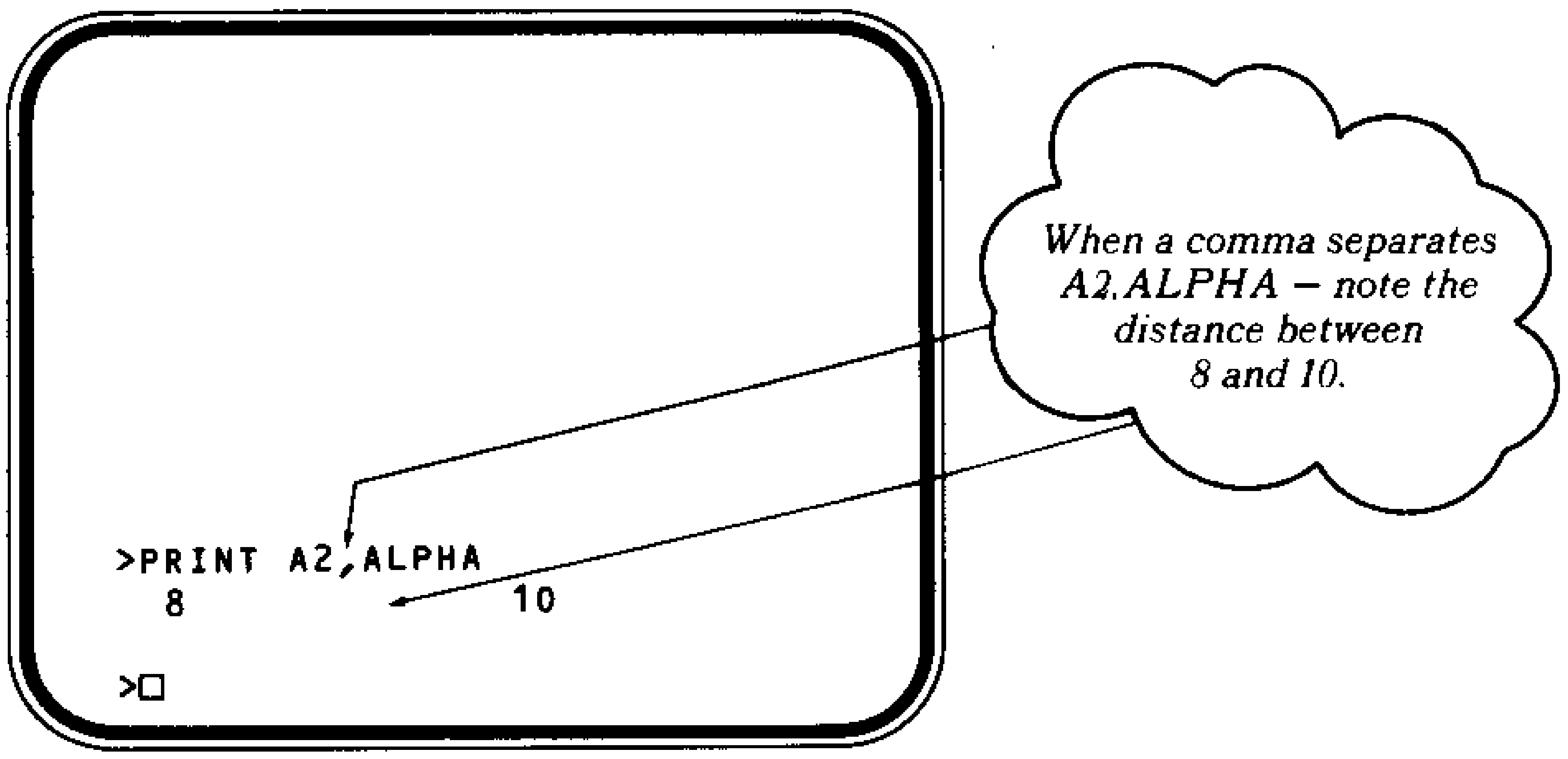
Did you notice that this PRINT statement is different from the PRINT statements we explored earlier? We didn't put quotation marks around the A. That's because we didn't want to print the letter A; we only wanted to see the numeric value assigned to A.

Now, check for the values of A2 and Alpha. (Remember! Press the **ENTER** key at the end of each line, even though it isn't shown in the illustration below.)



```
>PRINT A
  8

>PRINT A2
  8

>PRINT ALPHA
  10

>□
```

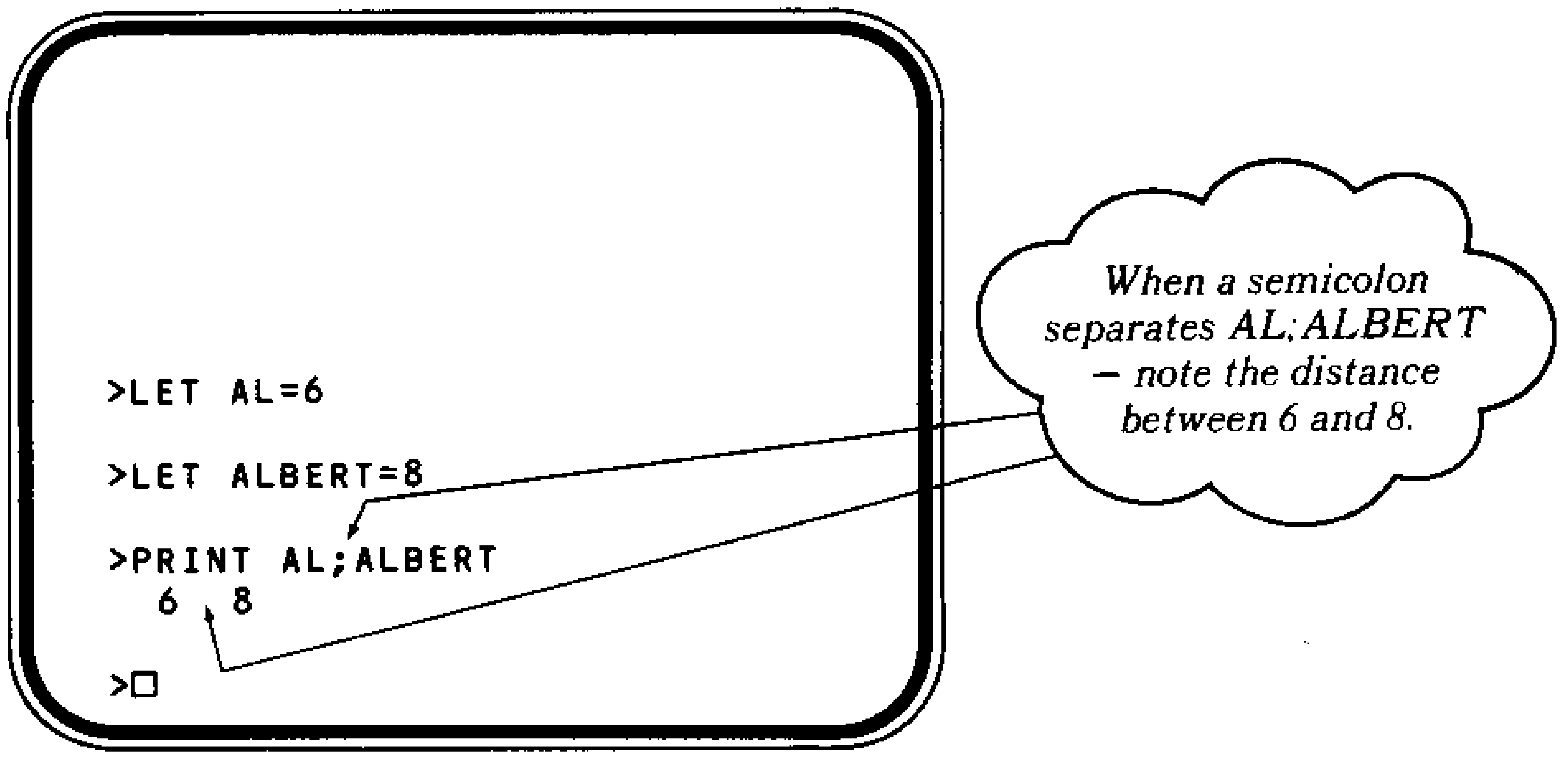
A single PRINT statement can also be used to print two or more things. Clear the screen, and try these examples:



```
>PRINT A2,ALPHA
  8                 10

>□
```

When a comma separates A2,ALPHA – note the distance between 8 and 10.

Now, try these:

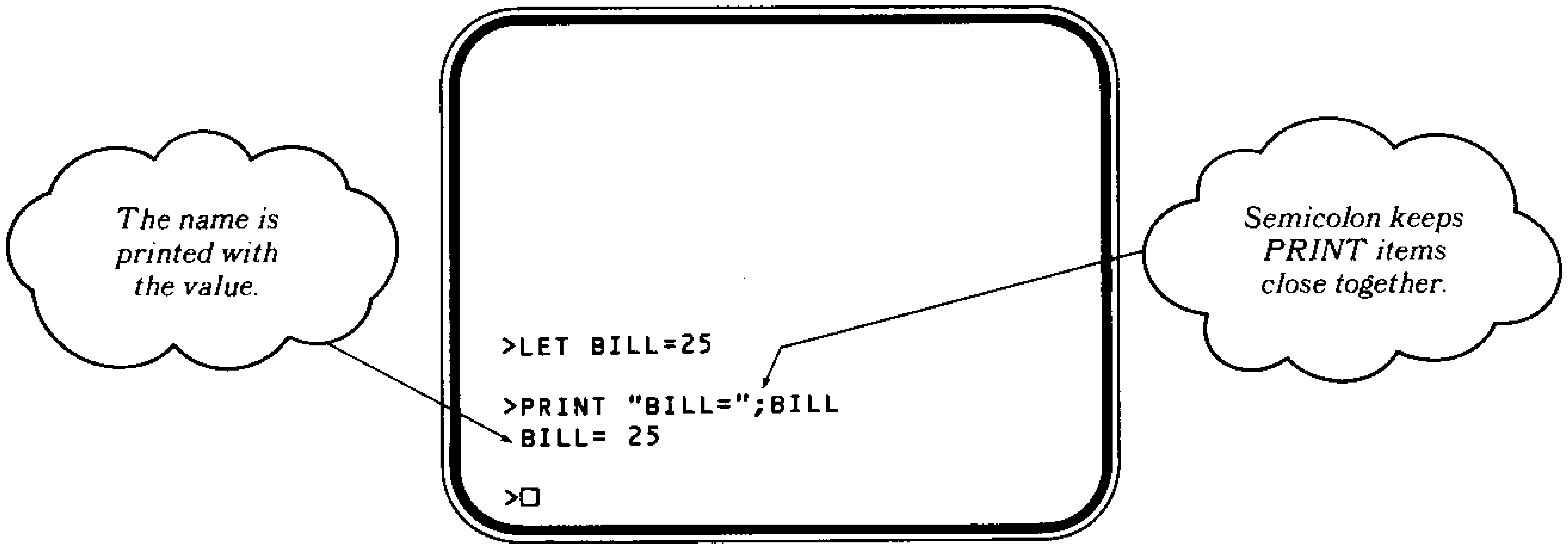


```
>LET AL=6

>LET ALBERT=8

>PRINT AL;ALBERT
  6  8

>□
```

When a semicolon separates AL;ALBERT – note the distance between 6 and 8.

The computer divides the display screen into two horizontal zones. When you use a comma (,) between two (or more) variables in a print statement, you are telling the computer to print the values in different zones. On the other hand, the semicolon (;) instructs the computer to print the numbers close together.

If you want to print the variable's name along with its value, you can. Remember our old friends, the quotation marks? Here's where we use them again:



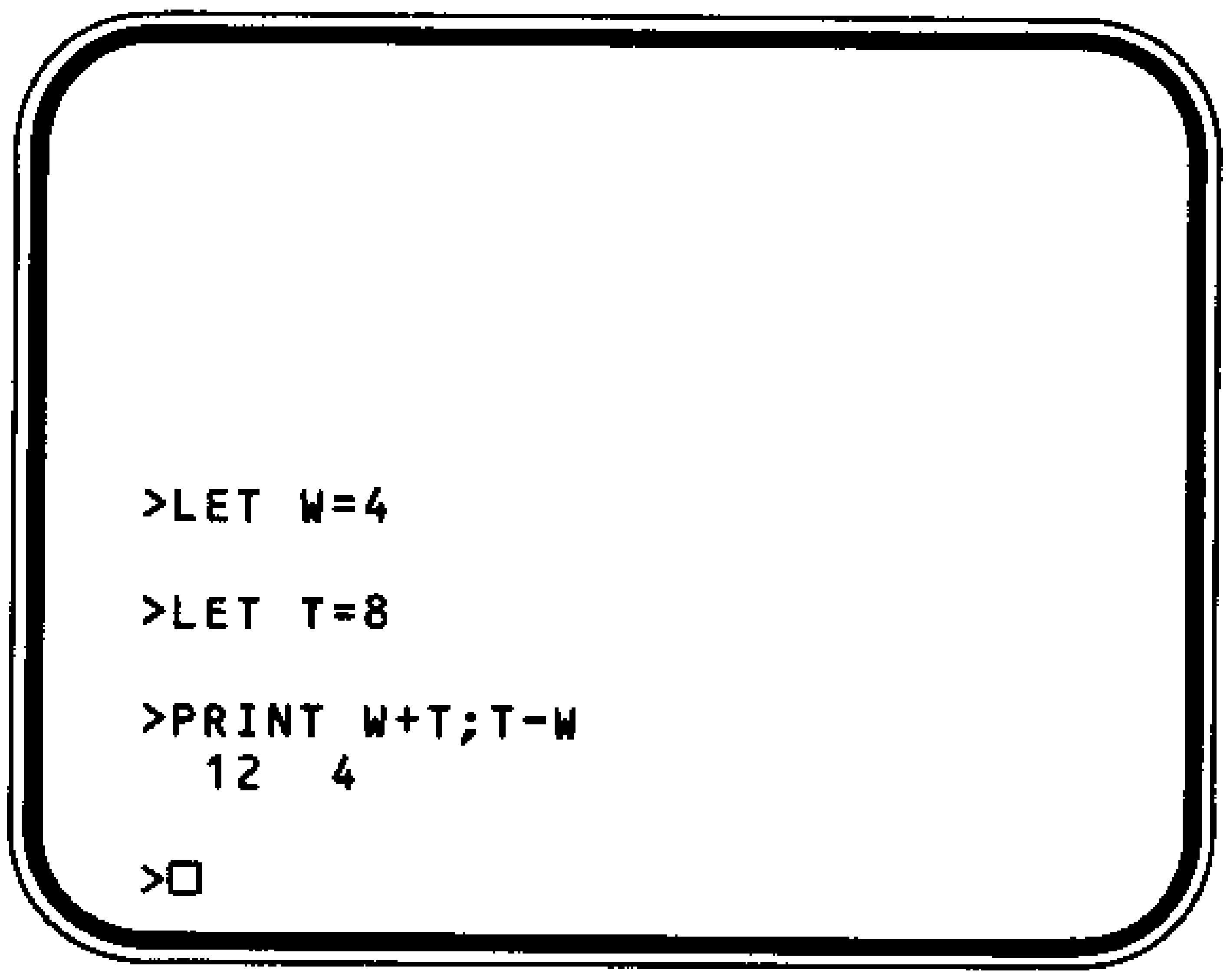
```
>LET BILL=25

>PRINT "BILL=";BILL
BILL= 25

>□
```

The name is printed with the value.
Semicolon keeps PRINT items close together.

(Did you remember to press **ENTER** at the end of each line?)

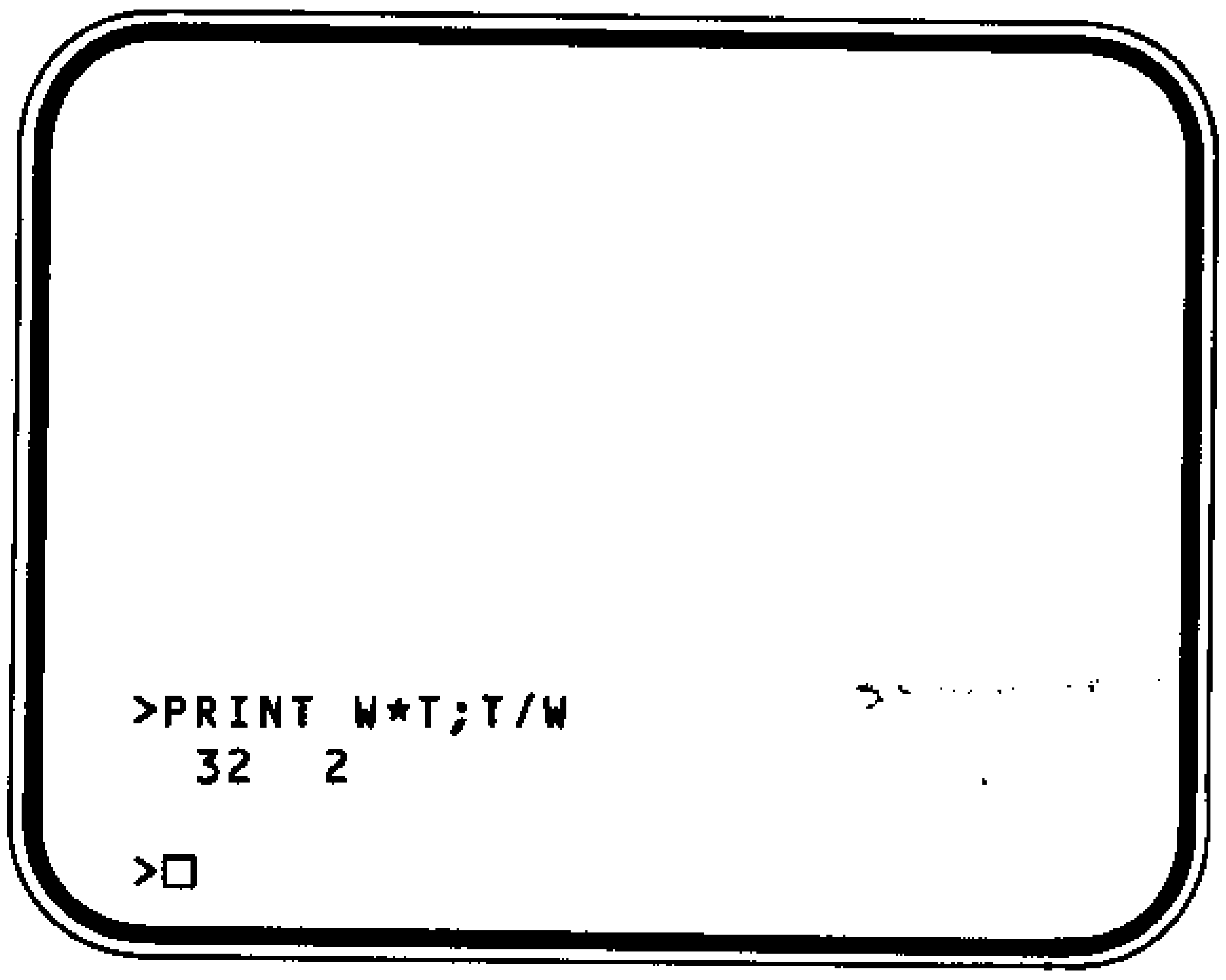
Now that you've learned to assign values to variables, what can you do with this new skill? Let's find out. First, use the CALL CLEAR statement to clear the screen.

After variables have been assigned values by LET statements, the PRINT statement may be used to perform arithmetic operations on the variables and to display the results.



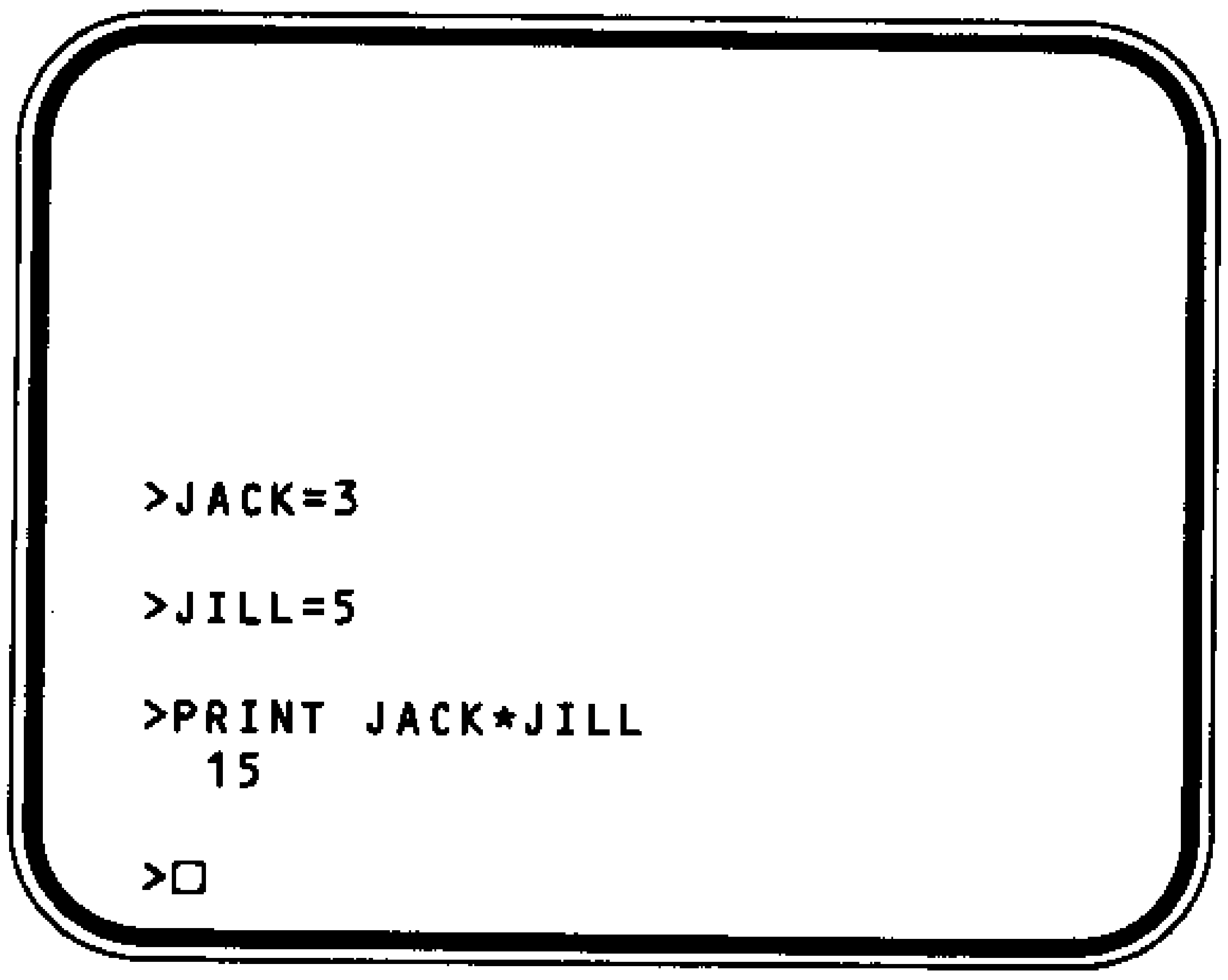
```
>LET W=4

>LET T=8

>PRINT W+T;T-W
  12  4

>□
```

You can also perform multiplication and division by using an asterisk (*) to multiply and a slash mark (/) to divide. For example,



```
>PRINT W*T;T/W
  32  2

>□
```

Note: In TI BASIC, the LET statement is not the only way to assign a numeric value to a variable. Your computer will also accept the assignment without the word LET:



```
>JACK=3

>JILL=5

>PRINT JACK*JILL
  15

>□
```

In other words, the word LET is optional in TI BASIC; your computer will accept the assignment either way.

Experiment!

Try other variable names and numeric values, and experiment with using the comma and semicolon to separate variables in the PRINT statement. Try adding, subtracting, multiplying, and dividing these variables in PRINT statements. Discover what mistakes will cause error messages.

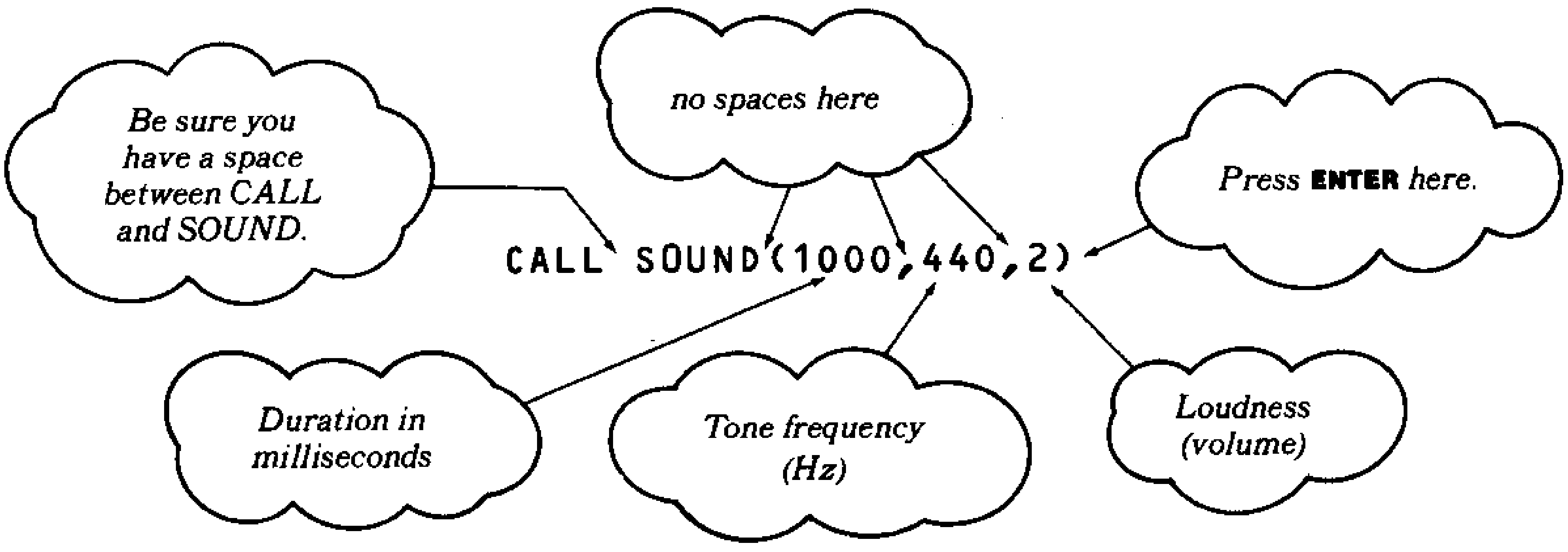
The CALL SOUND Statement

Here is another of the CALL statements. (Remember the CALL CLEAR statement we discussed earlier? We hope you've been using it occasionally to "erase" the display screen.)

Using the CALL SOUND statement, you can produce sounds over a range of frequencies from 110 to more than 44,000 Hertz. One Hertz (abbreviated Hz) is equal to one cycle per second. Thus the sounds you generate with your computer can vary from 110 cycles per second (A below low C on a piano keyboard) to over 44,000 (well above human hearing limits).

You can also control the *duration* and the *volume* of the sound. The time the sound lasts (duration) ranges from 1 to 4250 milliseconds. One thousand (1000) milliseconds equal one second, so the duration range could be stated as being from 0.001 to 4.250 seconds. Volume selections are scaled from 0 to 30. Zero and one produce the same sound level and are the loudest. Thirty produces the quietest tone.

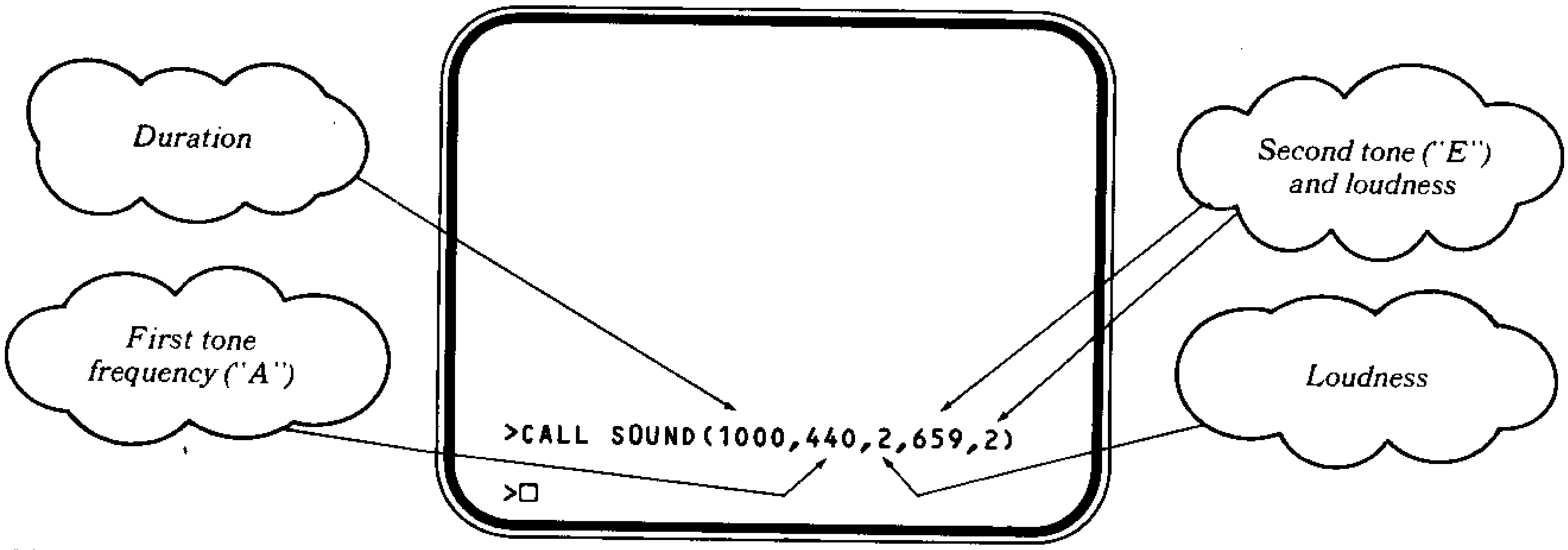
This example shows how to use the CALL SOUND statement:



Notice that the three values that control the sound are enclosed in parentheses following the words CALL SOUND. This example will produce a note of 440 Hz (A above middle C) with a duration of 1000 milliseconds (one second) and a volume of 2 (quite loud!).

Try the example now to hear the tone quality of your computer.

You can play more than one tone in a single CALL SOUND statement. Let's add a second note and see how this enhances the sound.

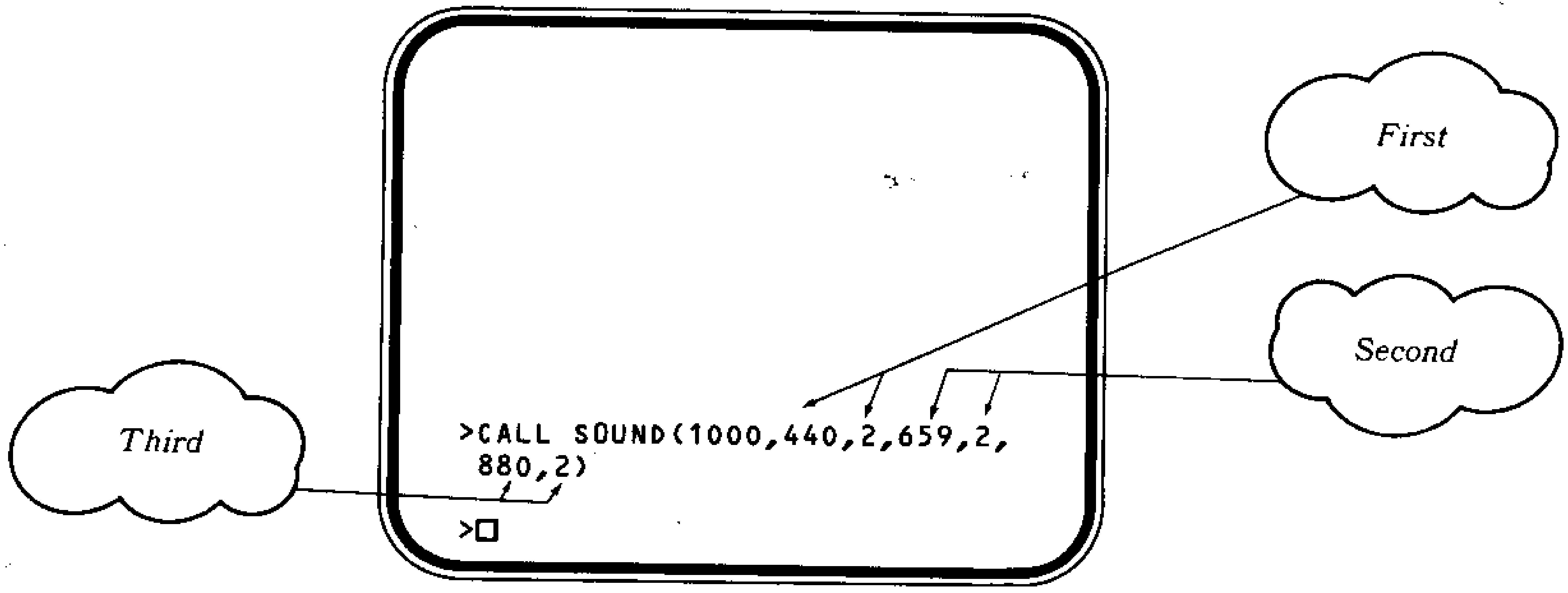


```
>CALL SOUND(1000,440,2,659,2)
>□
```

Note: Because the statement above contains exactly 28 characters (letters, spaces, and symbols), the cursor will move down to the next line as soon as you type the close parenthesis symbol. Be sure that you remember to press **ENTER**! (Notice that the prompting symbol stays at the beginning of your line.)

You only had to type the duration value (the number code that determines how long the sounds last) one time – at the beginning of the CALL SOUND instruction enclosed in parentheses. Both of the sounds *must* last for the same length of time. On the other hand, you *can* vary the loudness values. What would happen if you typed 5, instead of 2, for the second note's loudness? Try it!

Next, try a three-note chord:



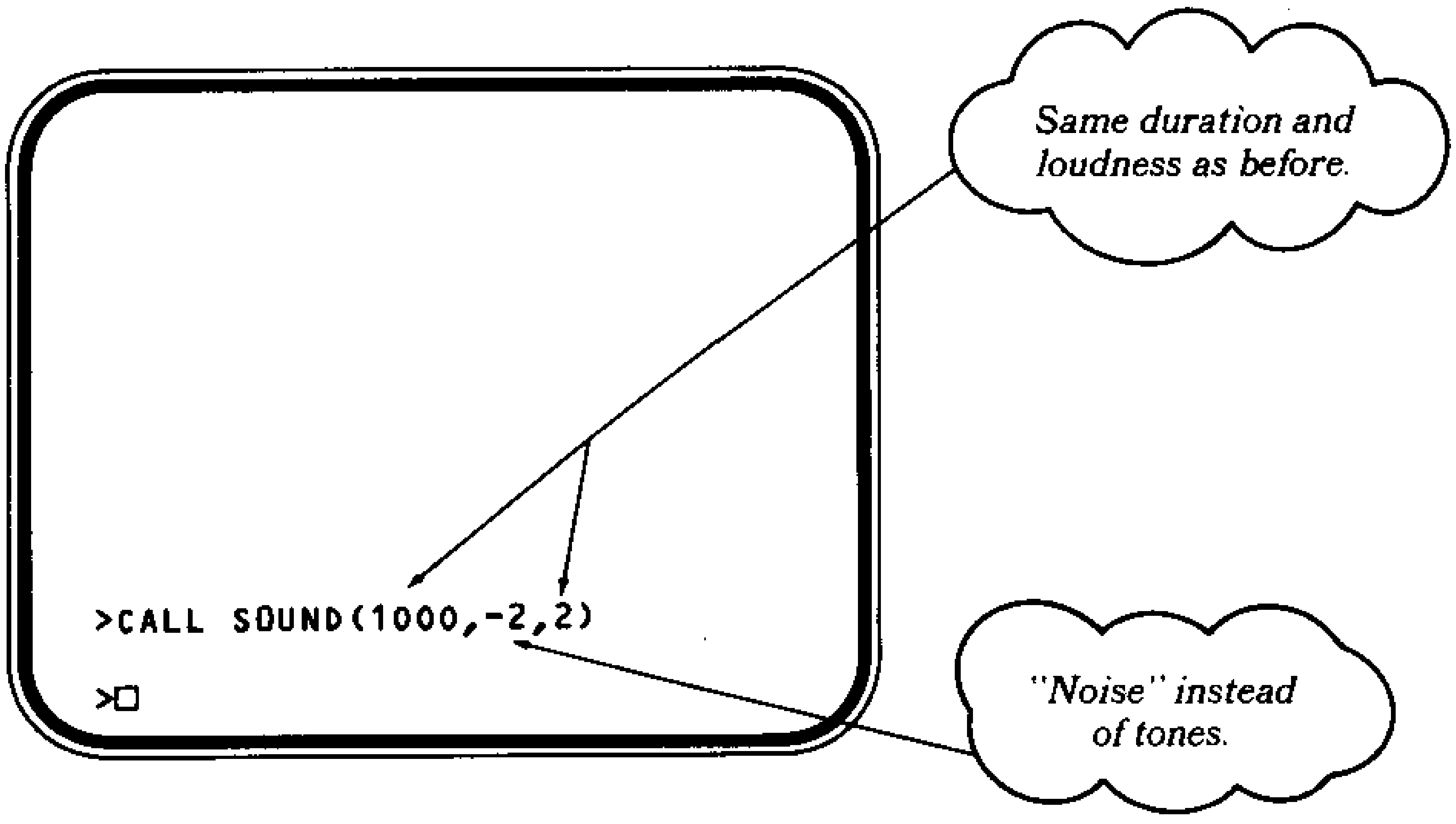
```
>CALL SOUND(1000,440,2,659,2,
 880,2)
>□
```

(Part of this CALL SOUND statement extends to the second line, since TI BASIC uses only 28 printing positions per line. This gives large, clear, readable text on the screen.)

You can produce up to three tones and one "noise" simultaneously over a given time duration. Noise is rather hard to define in words; it's best for you to experiment and hear for yourself. Remember, one person's "noise" may be another person's "music"!

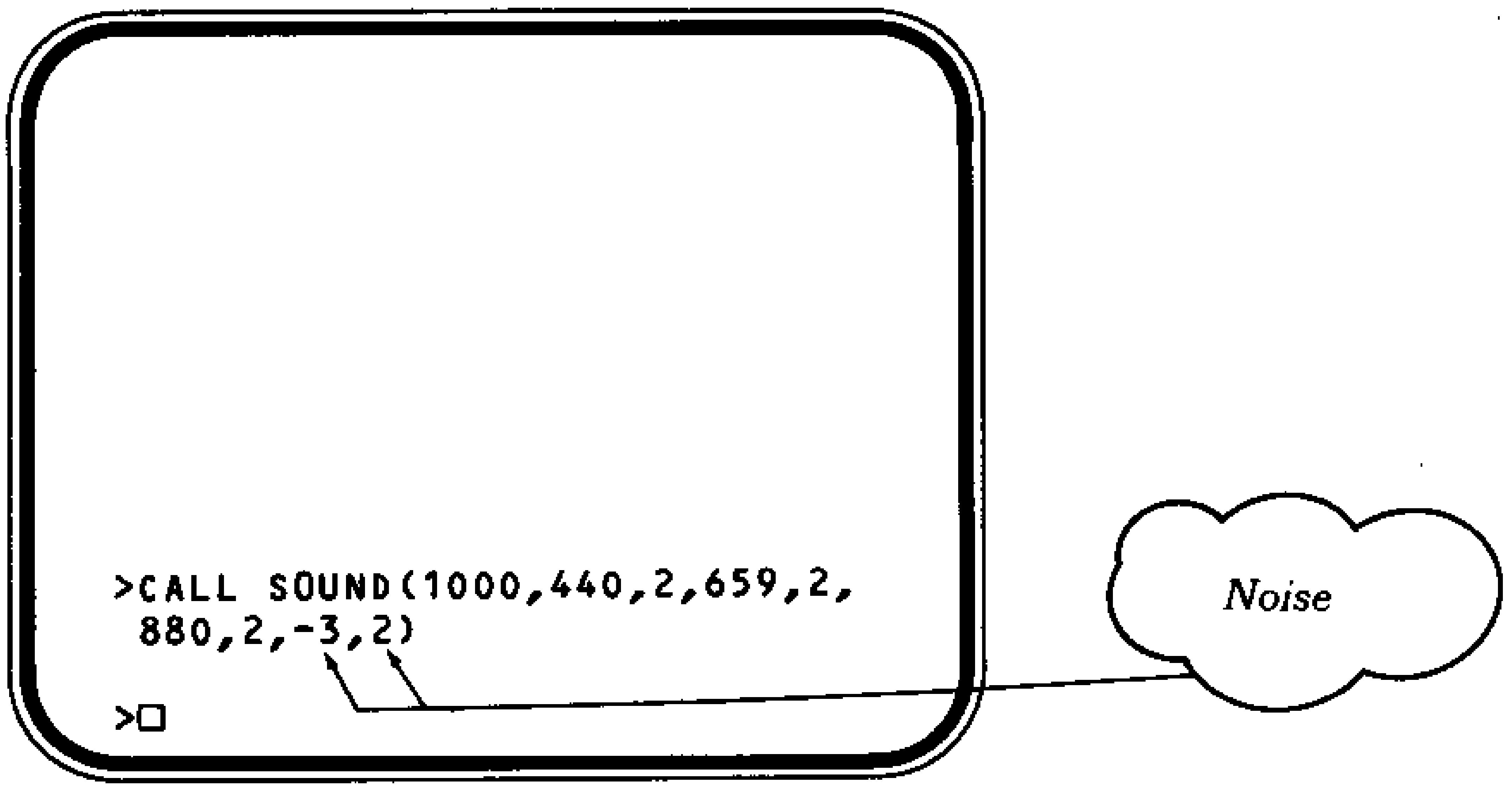
To produce noise instead of tones, replace the tone frequency with a negative integer from -1 to -8.

Try these examples:



```
>CALL SOUND(1000,-2,2)
>□
```

Same duration and loudness as before.
"Noise" instead of tones.



```
>CALL SOUND(1000,440,2,659,2,
 880,2,-3,2)
>□
```

You can also use variables, rather than actual values, in the CALL SOUND statement. For example, let's use these variables:

T = time (duration)
V = volume (loudness)
C = 262 (Middle C on the piano)
E = 330 (E)
G = 392 (G)

So type in the following LET statements:

```
LET T=1000
LET V=1
LET C=262
LET E=330
LET G=392
```

Now you're ready for the CALL SOUND statement. Type:

```
CALL SOUND(T,C,V,E,V,G,V)
```

and press **ENTER**.

Experiment!

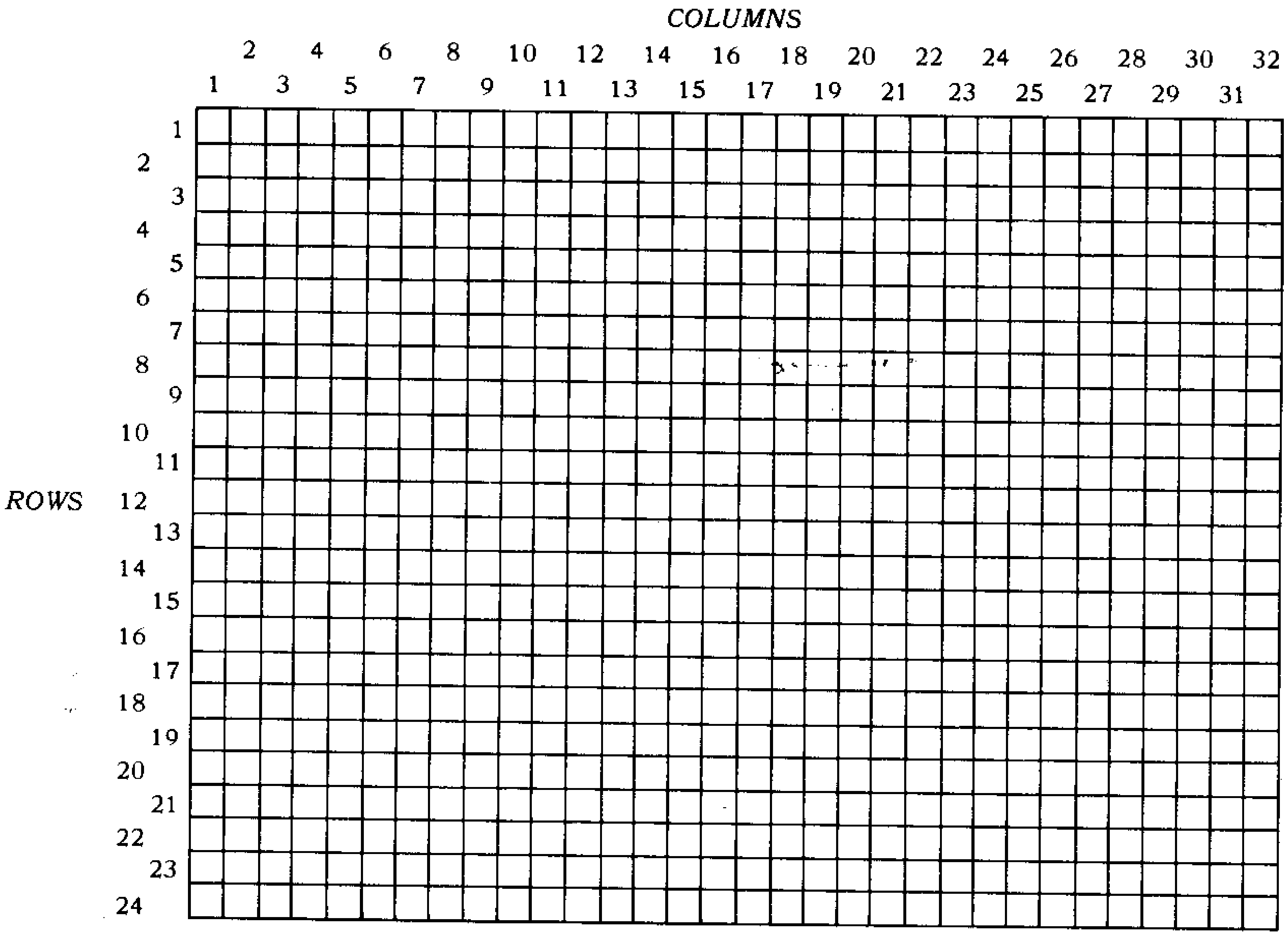
Experiment with other values for duration, tone, volume, and noise within the required range of values for each. (A list of musical note frequencies is included in *Appendix A*.) You'll soon be able to create imaginative sound effects for use in your future programs. The Immediate Mode is quite helpful for this type of experimentation.

Graphics (CALL VCHAR and CALL HCHAR)

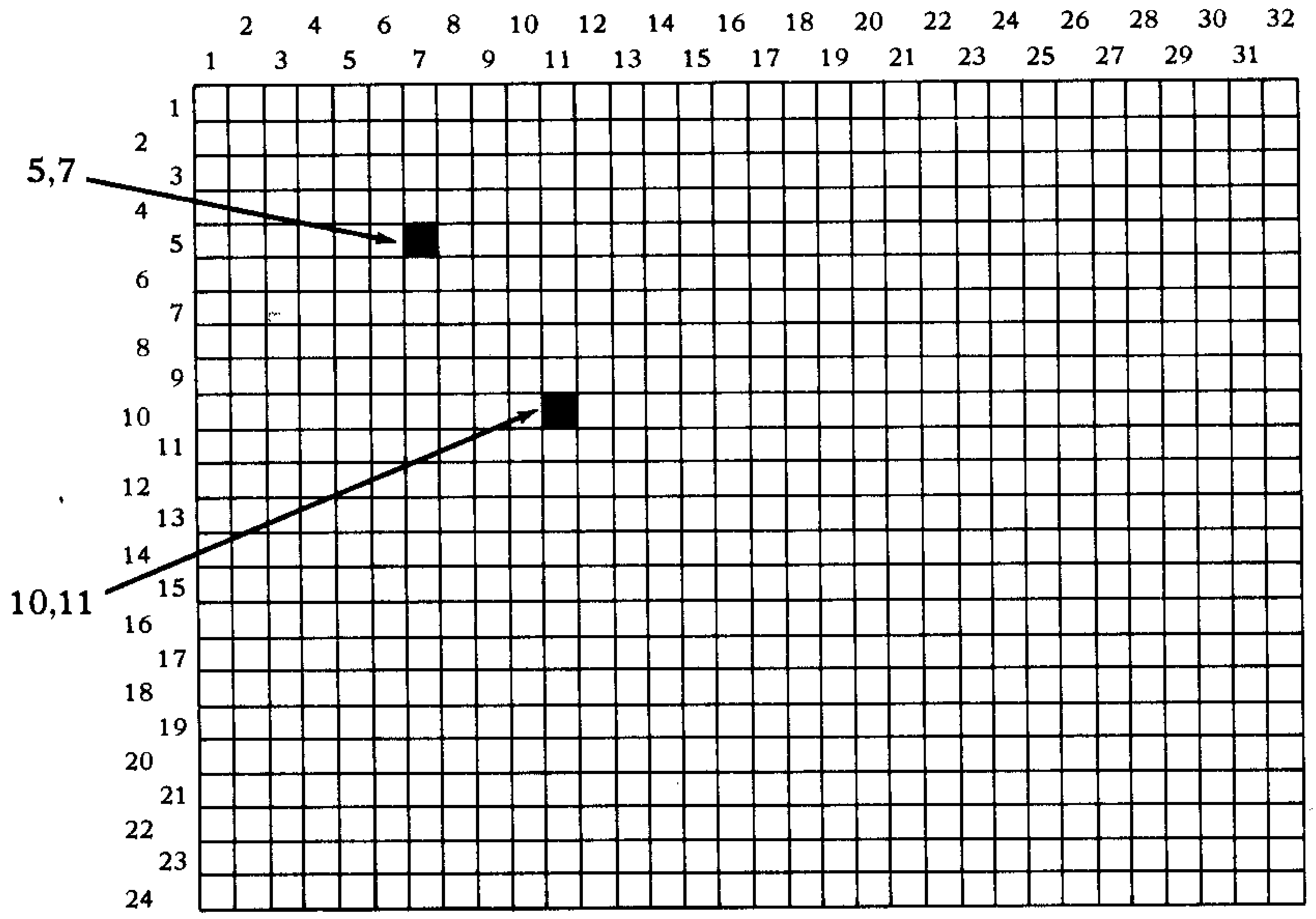
One of the most exciting things you can do with your computer is to create colorful designs right on the screen. With your computer's graphic capability, you can make a design, draw a picture, create a gameboard, and so on.

In this chapter, we'll introduce you to two simple, yet important, graphics statements. CALL VCHAR and CALL HCHAR are used to position a character or draw a line of characters on the screen. Later chapters will show you how to choose and combine colors and how to use graphics statements in programs.

Earlier we mentioned that TI BASIC uses 28 printing positions on each line. For graphics, however, the computer allows 32 character positions on each line. Think of the screen as a "grid" of square blocks made up of 32 columns and 24 rows.

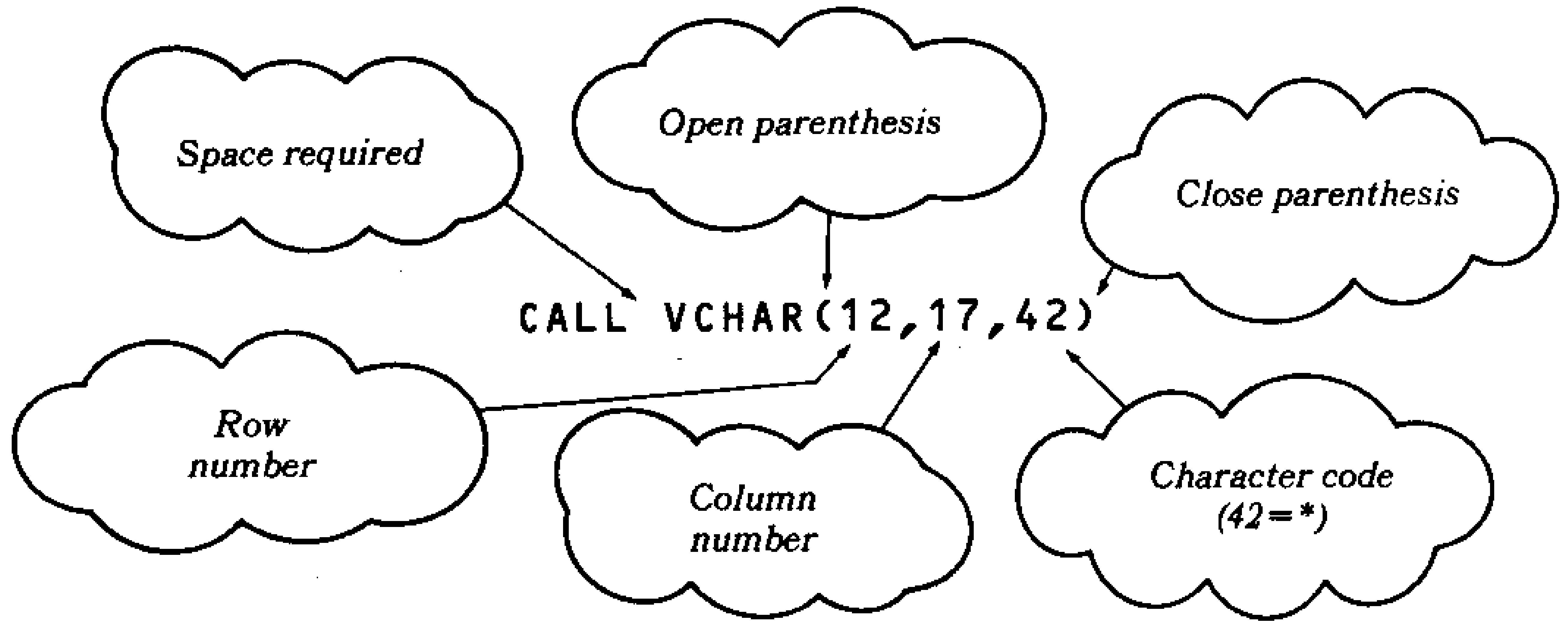


Each square on the grid is identified by two values (called *coordinates*) – a row number and a column number. For example, the coordinates 5,7 mean the fifth row and the seventh column, and the coordinates 10,11 mean the tenth row and the eleventh column.



The first thing we want to try is to place a character in a particular square on the screen. For the time being let's consider that a character is any one of the 26 letters of the alphabet, the numbers 0 through 9, and certain other symbols, like the asterisk (*), the plus and minus signs (+ and –), and the slash (/). (Later on, in Chapter 5, you'll learn more about how to define other characters for graphics.) Each character is assigned an identifying numeric value of its own, and the values for the full character set are given in *Appendix B.*

By using either CALL VCHAR or CALL HCHAR, naming the two coordinates (row and column), and identifying a character by its numeric value, you can place the character in any spot you choose. Here's the form used for these two statements:



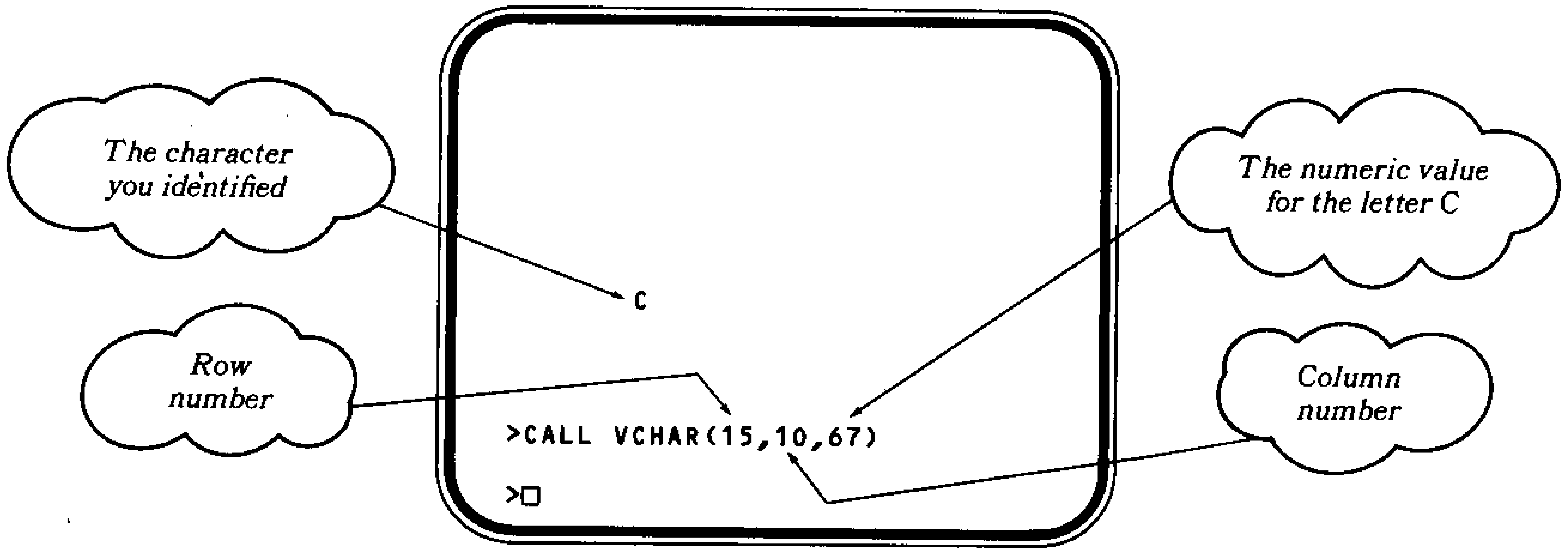
Try this example, and you'll see an asterisk (*) appear near the center of the screen.

Let's try a few more examples. First, clear the screen by typing CALL CLEAR and pressing **ENTER**. Now type:

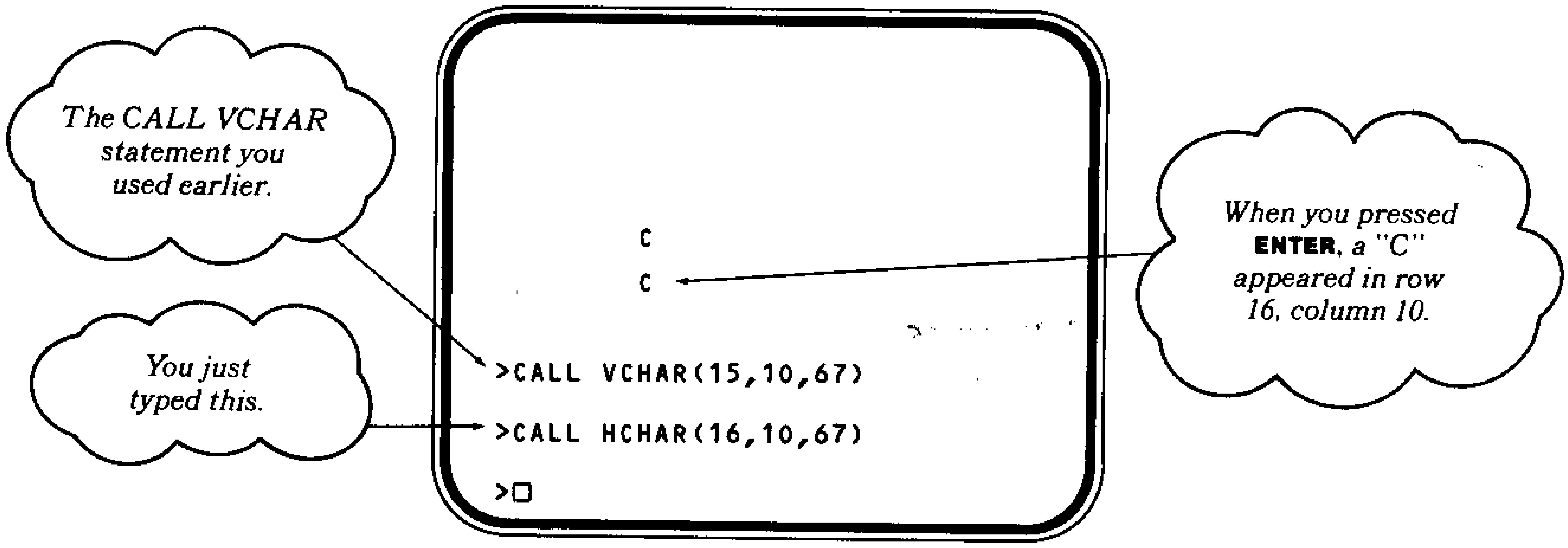
```
CALL VCHAR(15,10,67)
```

Check your typing and then press **ENTER**.

(Don't forget the parentheses in the statement – they're important!)


The character you identified
The numeric value for the letter C
Row number
Column number
C
>CALL VCHAR(15,10,67)
>□

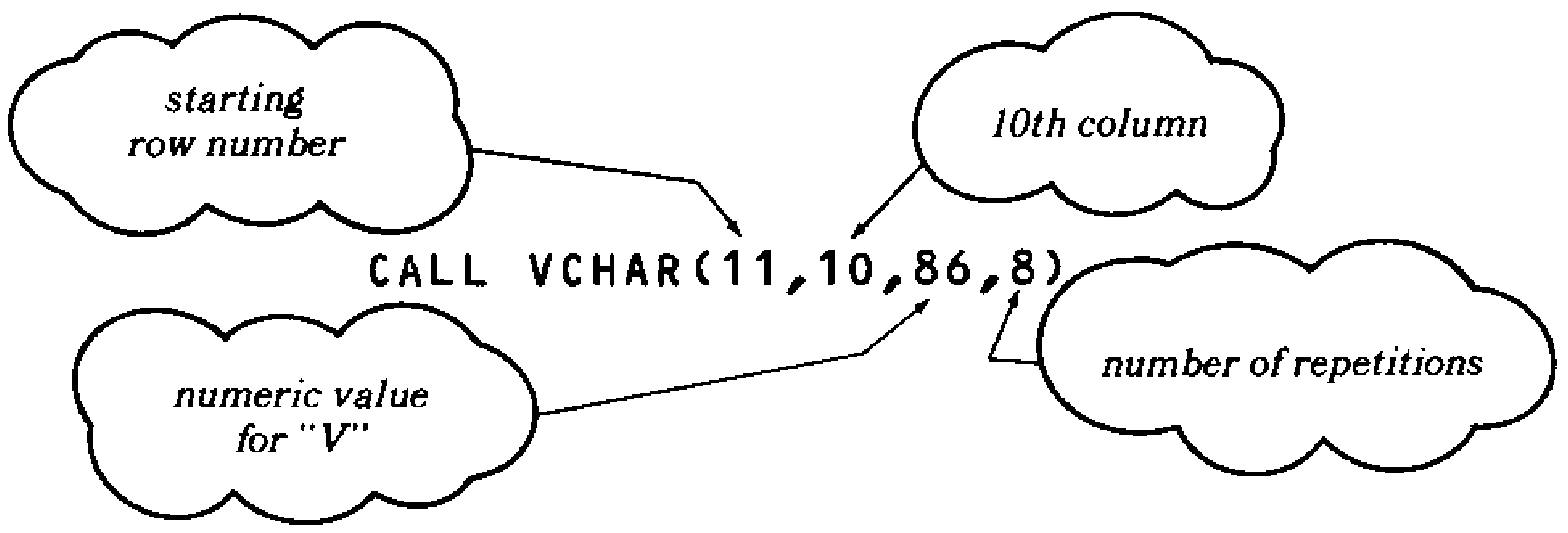
Now try the CALL HCHAR statement.


The CALL VCHAR statement you used earlier.
When you pressed ENTER, a "C" appeared in row 16, column 10.
You just typed this.
C
C
>CALL VCHAR(15,10,67)
>CALL HCHAR(16,10,67)
>□

The order for entering the row number, the column number, and the character's numeric value is the same for both CALL VCHAR and CALL HCHAR, and they both do the same thing *when you are positioning a single character on the screen.*

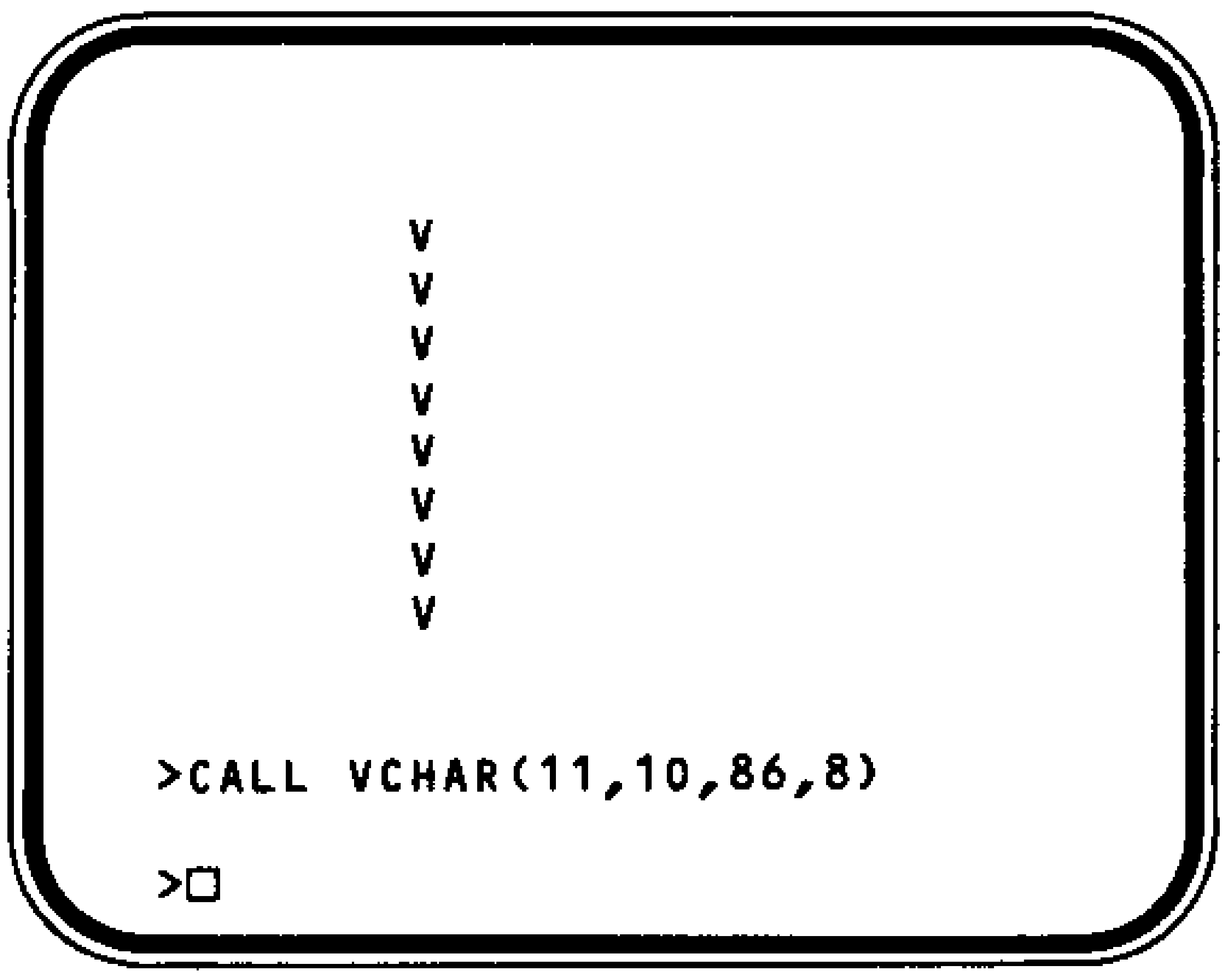
If you want to draw a line of characters, however, you'll find that there is a distinct difference between the functions of the two statements. CALL VCHAR causes a *vertical* column of characters to appear, while CALL HCHAR draws a *horizontal* row of characters. To draw a line with either statement, we must add a fourth numeric value to the statement: the number of repetitions we want. This number controls the "length" of the line.

Clear the screen (type CALL CLEAR and press **ENTER**), and let's try a vertical line. Type this:



```
CALL VCHAR(11,10,86,8)
```

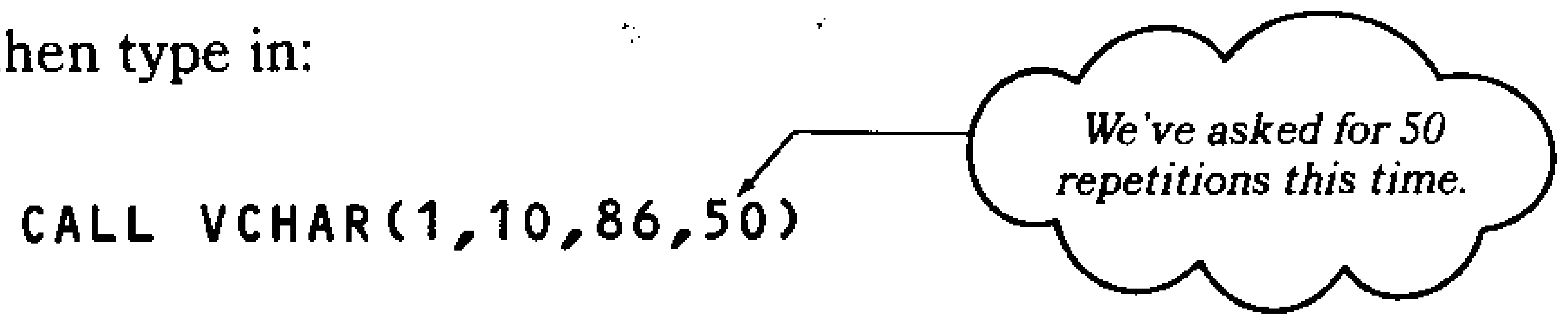
Check for errors, and then press **ENTER**. The screen will look like this:



```
        V
        V
        V
        V
        V
        V
        V
        V


>CALL VCHAR(11,10,86,8)

>□
```

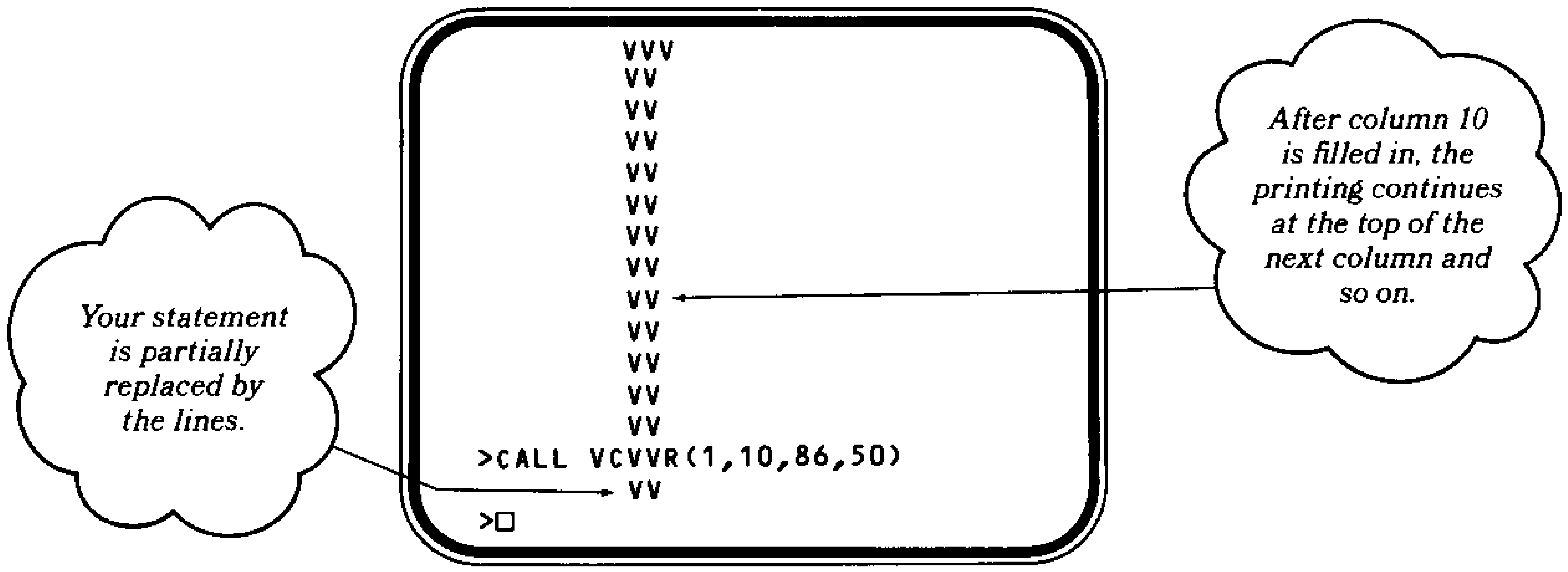
As we mentioned earlier, there are 24 horizontal rows of character blocks on the "grid" of the screen. Therefore, you can only draw a vertical line (column) that is 24 characters long. What will happen, then, if you enter a repeat value greater than 24? Let's try it.

Clear the screen and then type in:



```
CALL VCHAR(1,10,86,50)
```

When you press **ENTER**, the screen should show:



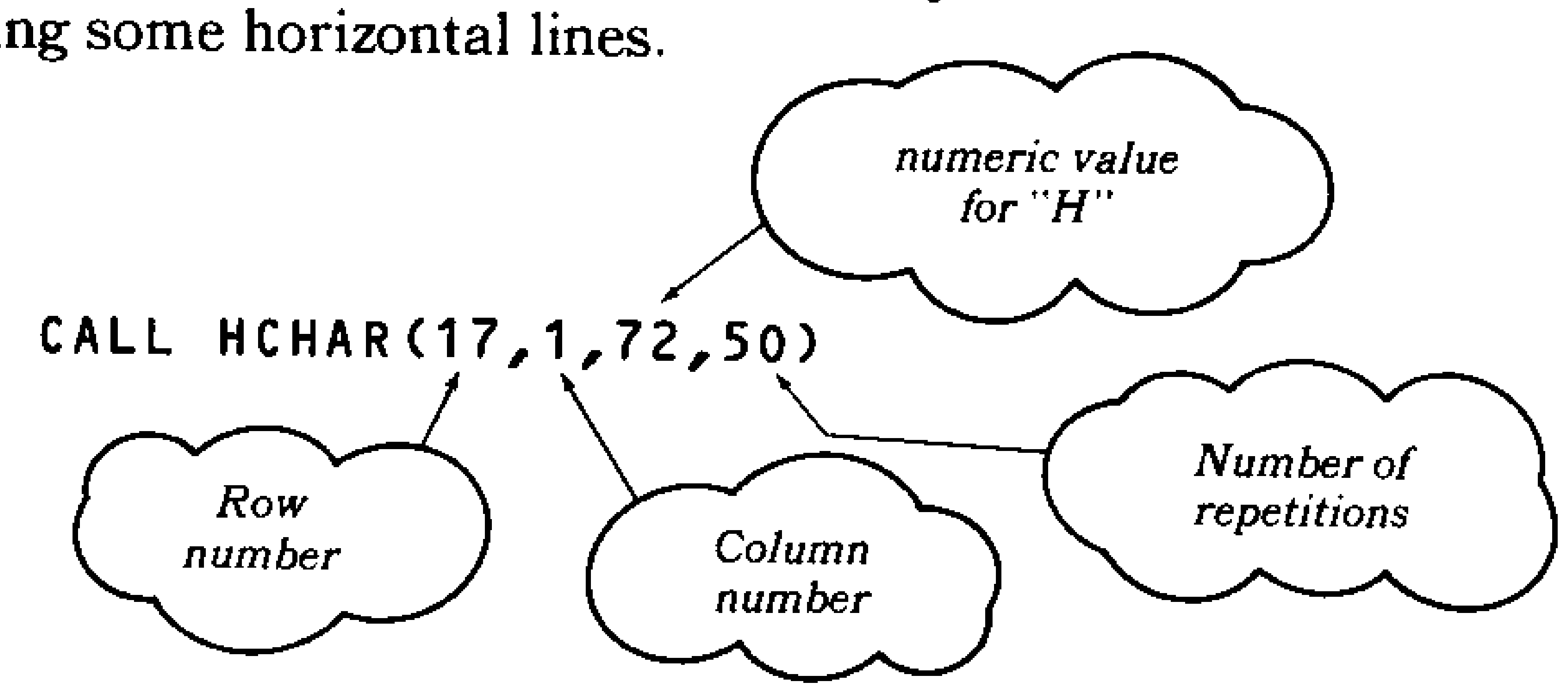
```
        VVV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
        VV
>CALL VCVVR(1,10,86,50)
        VV
>□
```

Your statement is partially replaced by the lines.
After column 10 is filled in, the printing continues at the top of the next column and so on.

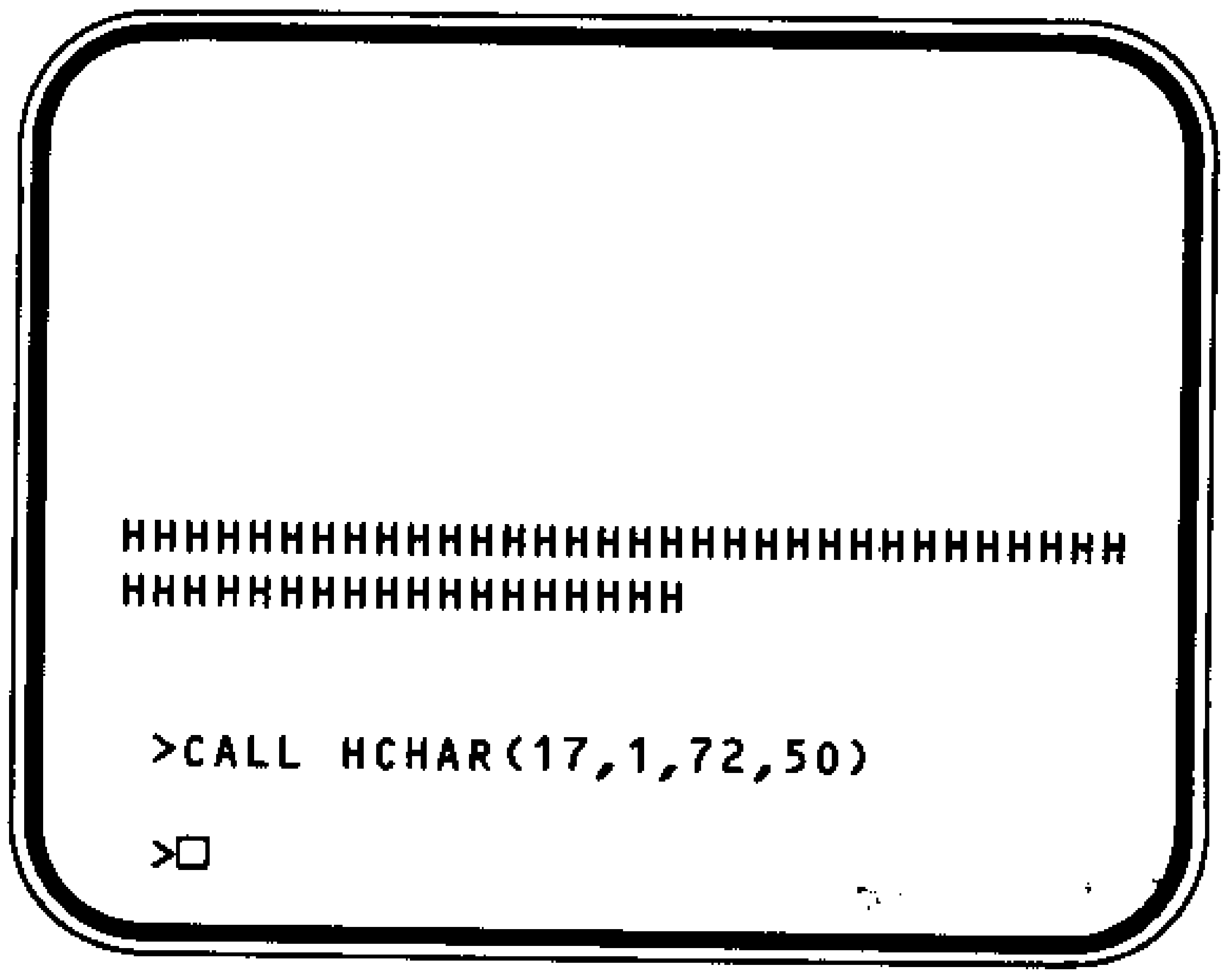
(*Note:* Graphics *in the Immediate Mode only* are affected by the scrolling of the screen. That's why you don't actually see all 50 of the V's above – some have already scrolled off the top of the screen.)

We also mentioned earlier that there are 32 vertical columns; therefore, it would seem that we could draw a horizontal line 32 characters long. However, some display screens may "clip off" the first two and last two columns (columns 1 and 2, 31 and 32). The only way to know what your screen shows is to experiment. So let's clear the screen and try drawing some horizontal lines.

Type in:



```
CALL HCHAR(17,1,72,50)
```



```
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHHHHHHHHH

 >CALL HCHAR(17,1,72,50)
 >□
```

Again the printing filled one line (horizontal, this time) and then started over on the next line. Count the H's. If you see only 28 in the full line, columns 1 and 2, 31 and 32 do not show on your screen, and you should use only columns 3 through 30 to avoid losing part of your graphic design.

So far, we've entered actual numeric values in our statements. However, you can use the LET statement to assign numeric values to variables and then use the variables in the CALL VCHAR and CALL HCHAR statements. Try this:

```
LET A=5
LET B=12
LET C=67
CALL CLEAR
CALL VCHAR(A,B,C)
```

Where did the "C" appear on the screen?

Experiment!

For a big finale let's fill the screen with asterisks (numeric code 42). Type these lines, pressing **ENTER** at the end of each line.

```
CALL CLEAR
CALL HCHAR(1,1,42,768)
```

24 rows × 32 columns = 768 positions.

Continue to experiment on your own, trying different characters (see *Appendix B*) and positions. For example, can you fill the screen with your first-name initial?

SUMMARY OF CHAPTER 1

This concludes our "tour" in the Immediate Mode, and you've been introduced to these BASIC statements:

PRINT
CALL CLEAR
LET
CALL SOUND
CALL VCHAR
CALL HCHAR

This chapter has given you a glimpse of TI BASIC and your computer's capabilities. Now, you're ready to get into the real fun – learning to program your computer.

2 CHAPTER TWO Simple Programming

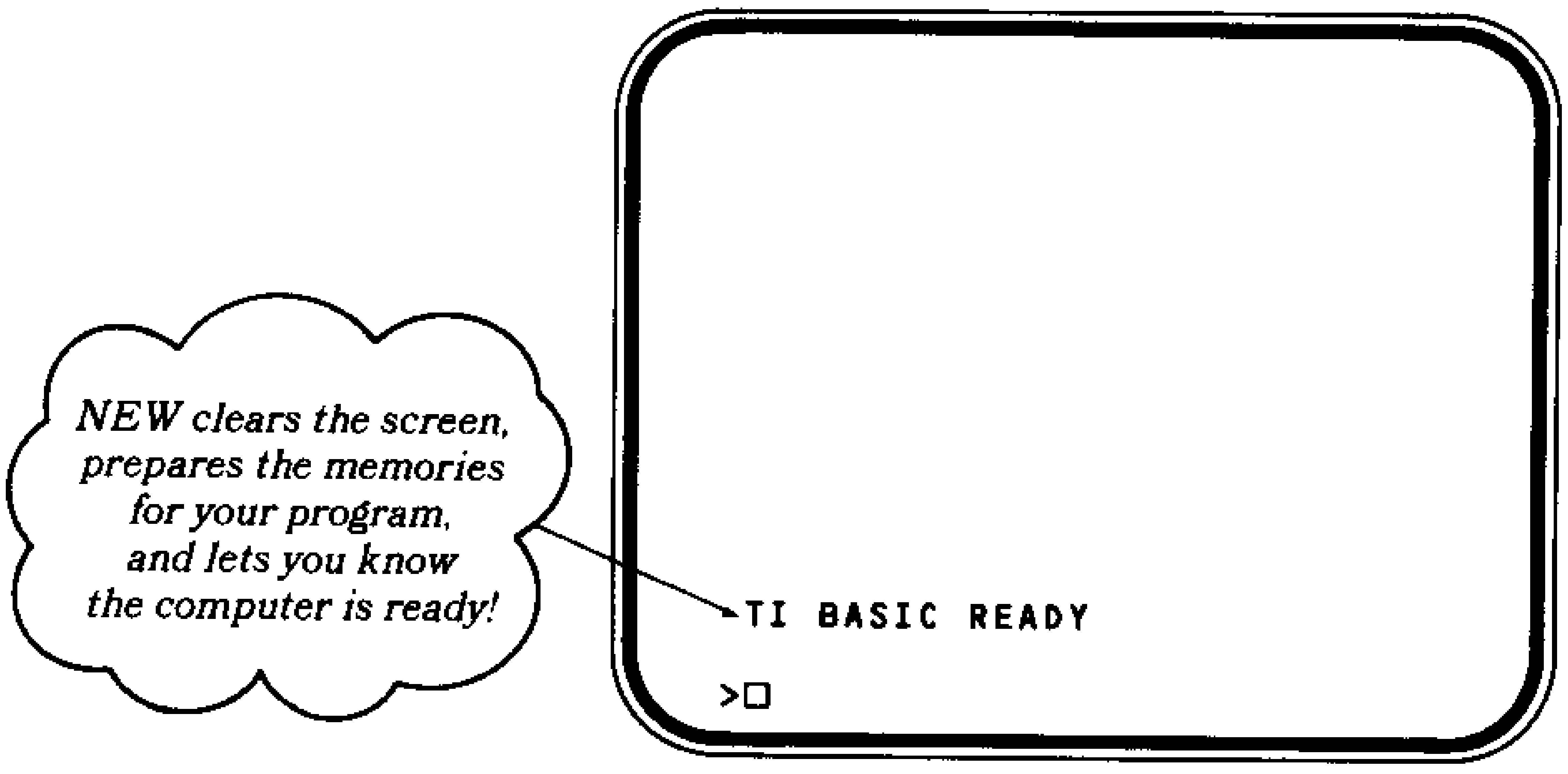
In Chapter 1, you used Immediate Mode statements to instruct the computer to do one thing at a time. Each statement was performed *immediately* after you pressed the **ENTER** key.

You typed PRINT "HI THERE!" and pressed **ENTER**.
The computer printed HI THERE!

Now you're ready to discuss *programs,* sets of statements which are *not* done immediately. Instead, they are stored in the computer's memory, waiting for you to instruct the computer to perform them.

A Printing Program

Let's begin by using an old familiar friend, the PRINT statement, in a program. First type the word NEW and press **ENTER**.


NEW clears the screen, prepares the memories for your program, and lets you know the computer is ready!
TI BASIC READY
>□

Now type the following program, pressing **ENTER** at the end of each program line:

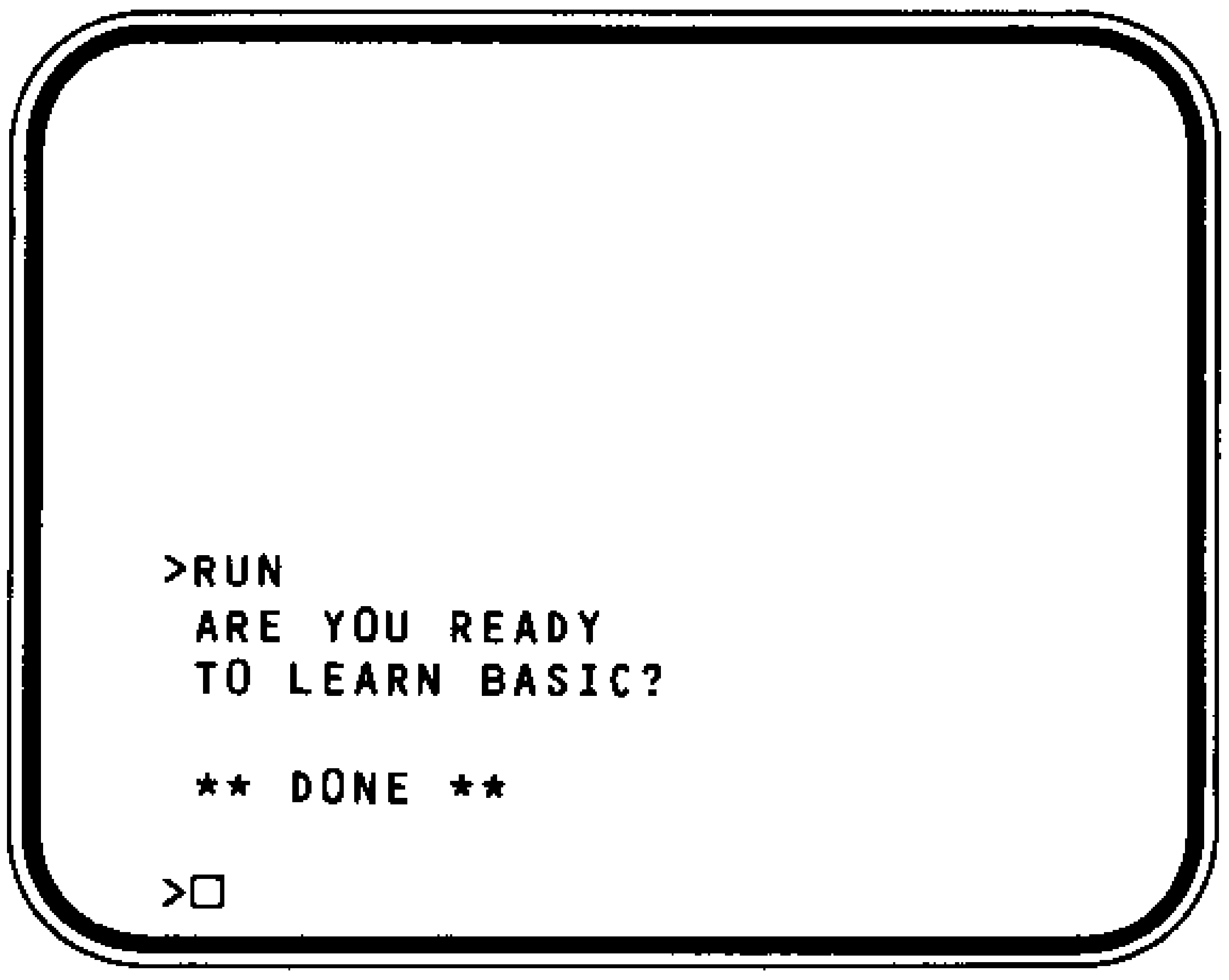
```
10 PRINT "ARE YOU READY"
20 PRINT "TO LEARN BASIC?"
30 END
```

(As you type the program, notice the small "prompting" character that appears just to the left of the printing area. This symbol marks the beginning of each program line you type.)

In computer terminology, you have just "entered" a program. Nothing to it! Check the program now to see if there are any typing mistakes. If there are, just retype the line correctly, *including the number at the beginning of the line,* right there at the bottom of the screen. Then press **ENTER**. The computer will automatically replace the old line with the new, correct one.

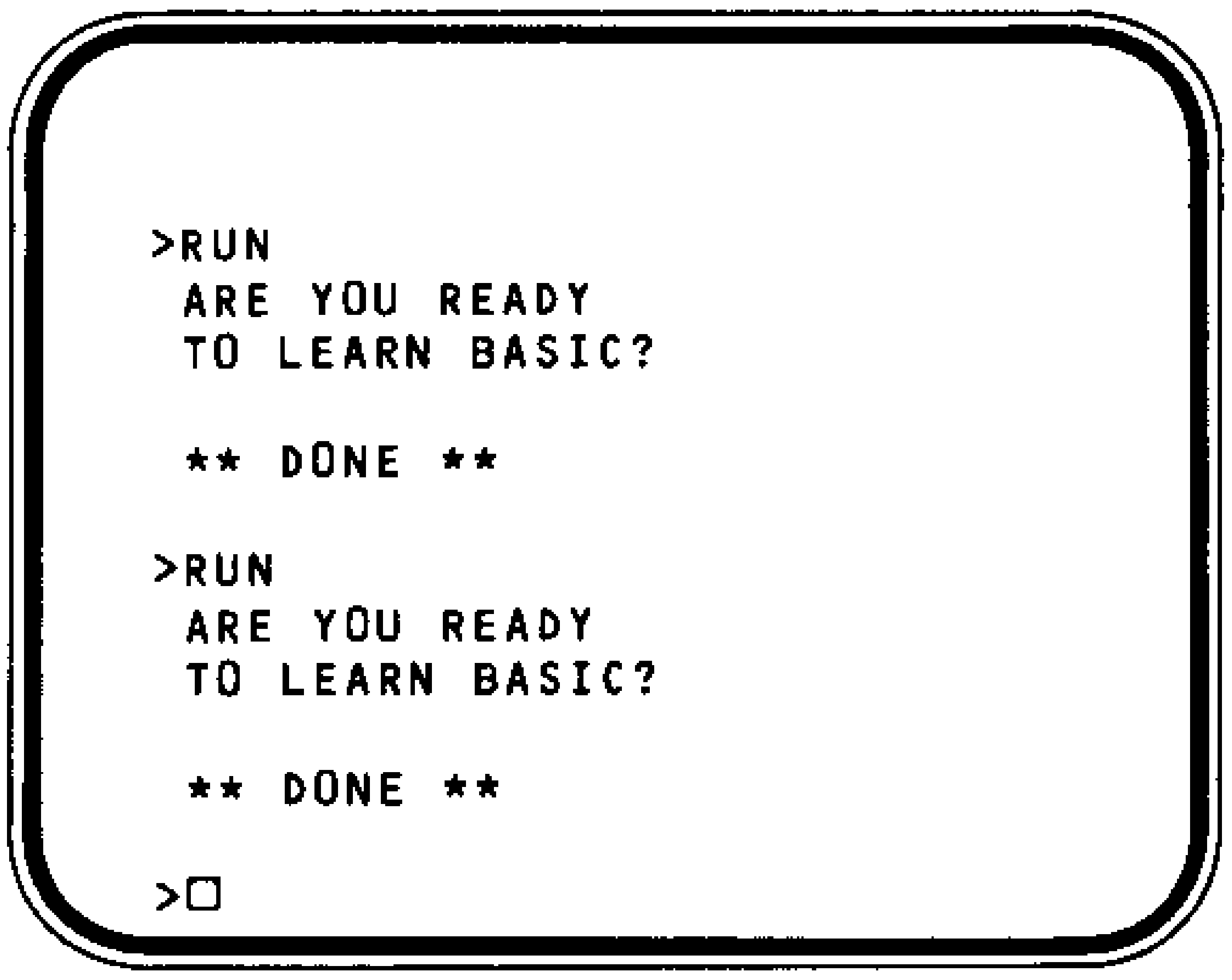
When you're ready to see the program in action, type CALL CLEAR and press **ENTER**. The screen will be cleared, but your program won't be erased – it's stored in the computer's memory!

Now type RUN and press **ENTER** again.



```
>RUN
 ARE YOU READY
 TO LEARN BASIC?

 ** DONE **

>□
```

Want to "run" the program again? Type RUN again and press **ENTER**.



```
>RUN
 ARE YOU READY
 TO LEARN BASIC?

 ** DONE **

>RUN
 ARE YOU READY
 TO LEARN BASIC?

 ** DONE **

>□
```

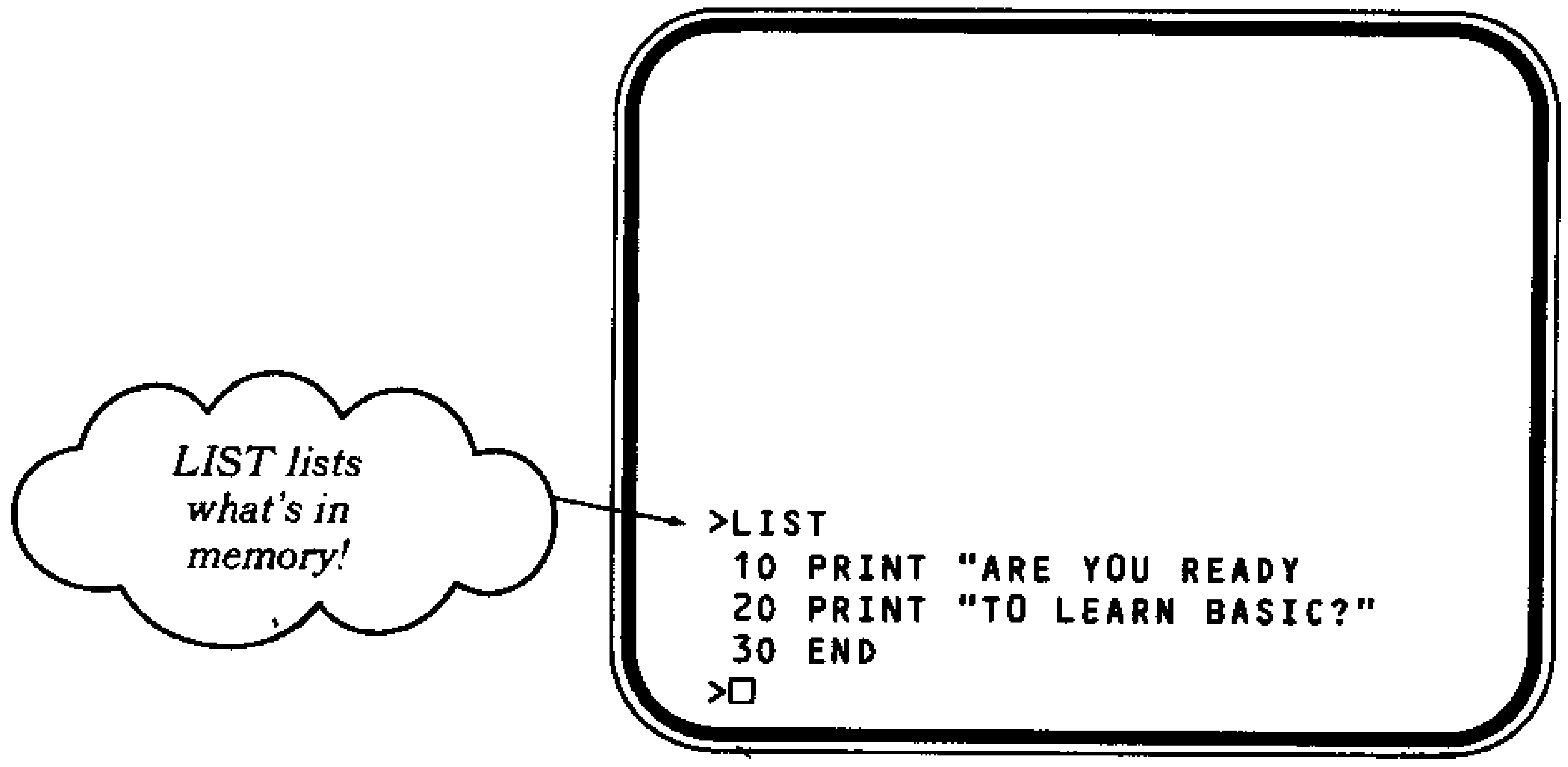
Each time you type RUN and press **ENTER**, the computer begins at the first statement and follows your instructions in order until it reaches the last statement. END means just what it says: the end, stop!

Did you notice that the display screen briefly turned green while the program was running? The screen always turns green while a program is being executed and then changes back to its normal blue color when the program is finished.

Program Structure

Now that you've had a bit of programming experience, let's review some of the things you did when you entered the program above. To refresh your memory, we'll get the program back on the screen.

First, type CALL CLEAR (*without* a line number) and press **ENTER** to clear the screen. Now type LIST and press **ENTER** again:

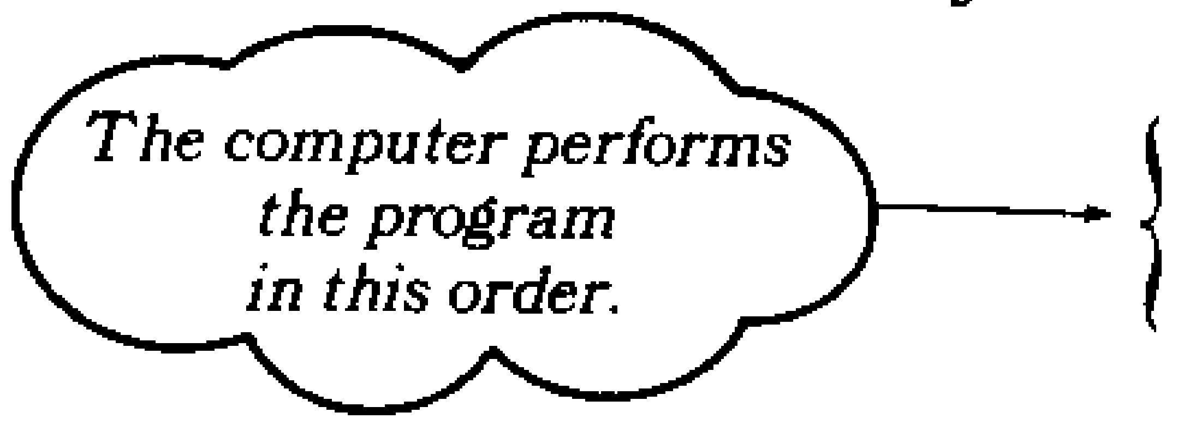


```
>LIST
 10 PRINT "ARE YOU READY
 20 PRINT "TO LEARN BASIC?"
 30 END
>□
```

The program above consists of three statements or "lines." Each statement begins with a *line number*, which serves two important functions:

1. It tells the computer *not* to perform the statement immediately, but to store it in memory when you press **ENTER**.
2. It establishes the *order* in which the statements will be done in the program.

As in the Immediate Mode, you pressed **ENTER** when you finished typing each program line. Pressing **ENTER** defines the end of the program line, just as the line number identifies the beginning of the line. It is also the computer's cue to store the line in its memory. *Pressing* **ENTER** *at the end of each program line is essential* – without it, your line will not be correctly stored by the computer.



```
10 PRINT "ARE YOU READY"
20 PRINT "TO LEARN BASIC?"
30 END
```

Also, you may be wondering why we numbered the lines in increments of ten (10,20,30,etc.). Well, we could just as easily have numbered them 1,2,3. By using increments of ten, however, or other spreads like 100,200,300, etc., we can go back and insert new lines if we want to expand the existing program, and we don't have to retype the whole program! (We'll cover this clever trick when we discuss editing a program.)

Commands—NEW, RUN, LIST

You've already used these commands, but you might like a little more definition of commands in general and these three in particular at this point.

Commands are different from *statements*. They are not part of the program, and they do not have line numbers. Instead, they instruct the computer to do specific tasks:

NEW –Instructs the computer to erase the program in its memory. (It also clears the screen, but *don't* confuse it with CALL CLEAR, which *only* clears the screen.)

RUN —Instructs the computer to perform (or "run") the program in its memory.

LIST —Instructs the computer to show (or "list") on the screen the program that is stored in its memory.

As you saw earlier, we use NEW *only* when we want to prepare the computer for storing a new program. Be careful in using NEW; when in doubt, use LIST first, so that you can see the current program before you erase it.

LIST is a powerful aid for correcting or changing a program. It lets you get the program right on the screen in front of you, where you can check for and correct any errors in your program.

And you already know what RUN will do! It's the magic word that makes it all happen.

A Numerical Program

In addition to its printing or "message" capabilities, your computer also has a great deal of "number power." You experimented with addition, subtraction, multiplication, and division in the Immediate Mode in Chapter 1. Now it's time to try a mathematical problem-solving program. Just to refresh your memory, review the keys that are used to perform the four basic mathematical operations:

SHIFT **+** for addition
SHIFT **–** for subtraction
SHIFT * for multiplication
/ for division

*Notice that +, –, and * are the upper symbols above =, /, and 8.*

As an example, we can easily construct a program that will convert kilograms to pounds (1 kilogram=2.2 pounds). The first thing we'll do is to clear the display and the computer's memories by typing NEW and pressing **ENTER**. We'll use the variables K (for kilograms) and P (for pounds) to help us remember which value is which, and we'll begin our program by assigning values to these variables.

Type:
```
10 LET K=50
20 LET P=2.2*K
```
Press **ENTER**.

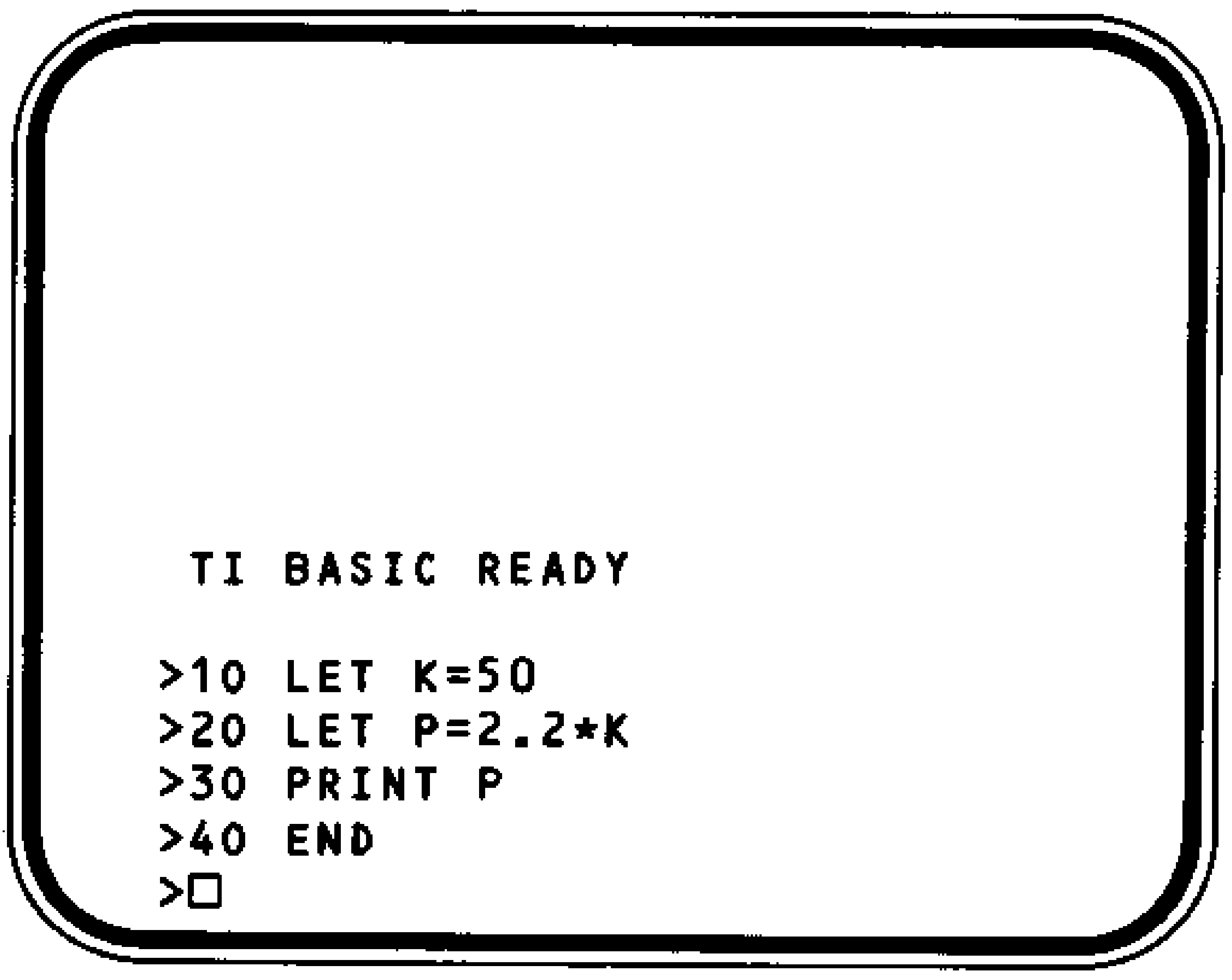
In this case, we are trying to find out how many pounds are equal to 50 kilograms, so we have defined K as 50. Notice that we have defined P as 2.2×K. If we stopped here and ran the program at this point, the computer would perform the conversion, but it wouldn't show us the answer! So type in:

```
30 PRINT P
```

and press **ENTER**. Now, have we told the computer everything it needs to do? We've told it the number of kilograms we want converted to pounds, we've told it how to make the conversion, and we've told it to show us the answer. Yes, that's all we need, so type:

```
40 END
```

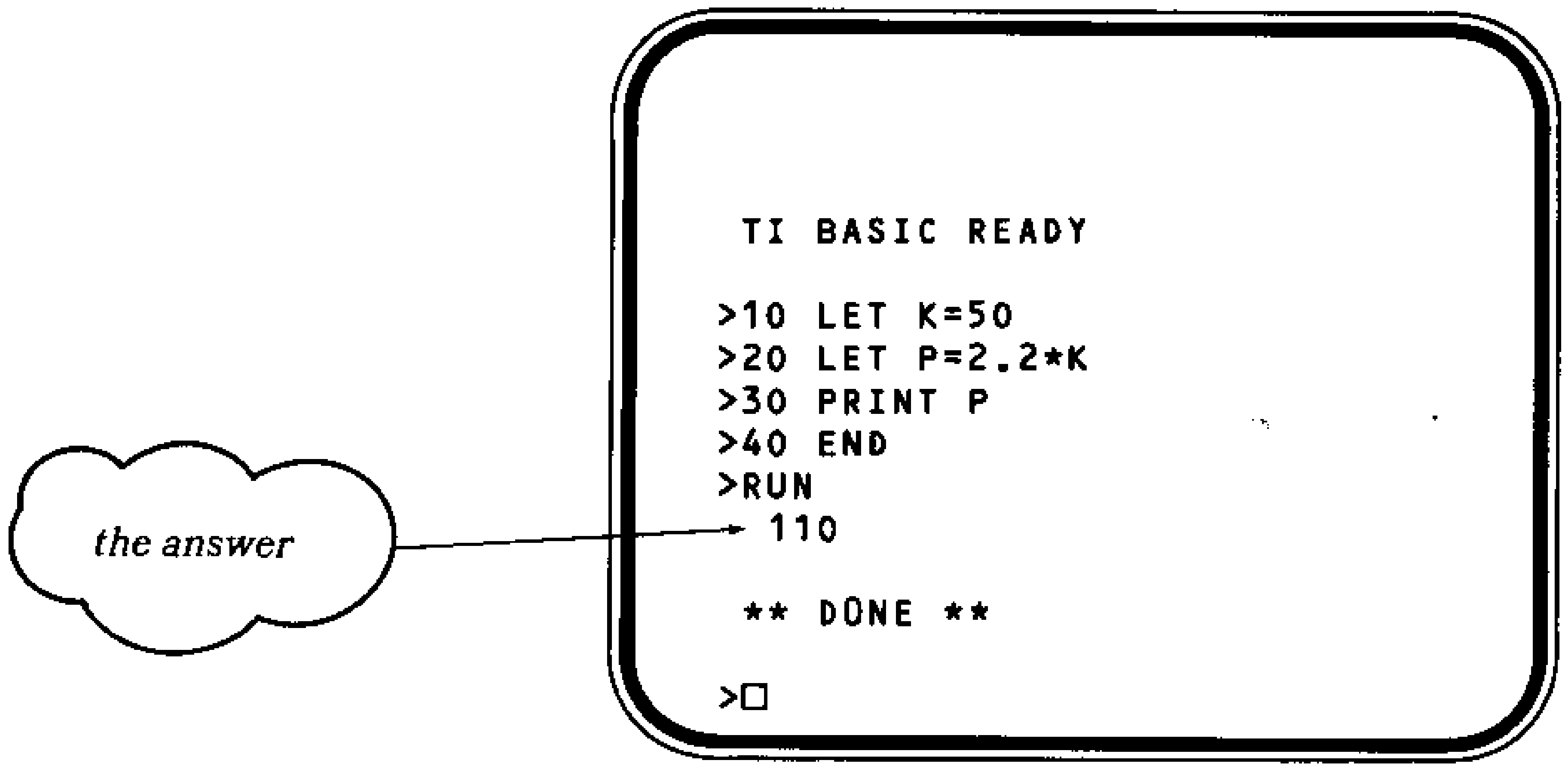
and press **ENTER**. Your program should look like this:



```
TI BASIC READY

>10 LET K=50
>20 LET P=2.2*K
>30 PRINT P
>40 END
>□
```

Before you run the program, let's mention two features of TI BASIC that may be slightly different from other versions of the language. First, a prompting character (to the left of the printing field on the screen) marks the start of every program line you type. You'll see its function more clearly when you begin to enter program lines that are longer than a single screen line. Second, the END statement in a program is optional in TI BASIC. Since it is a conventional part of BASIC, however, we'll use it in this example.

Now check the program for typographical errors. If there are any, retype the line correctly, including the line number, and press **ENTER**. When you're ready, type RUN and press **ENTER**.

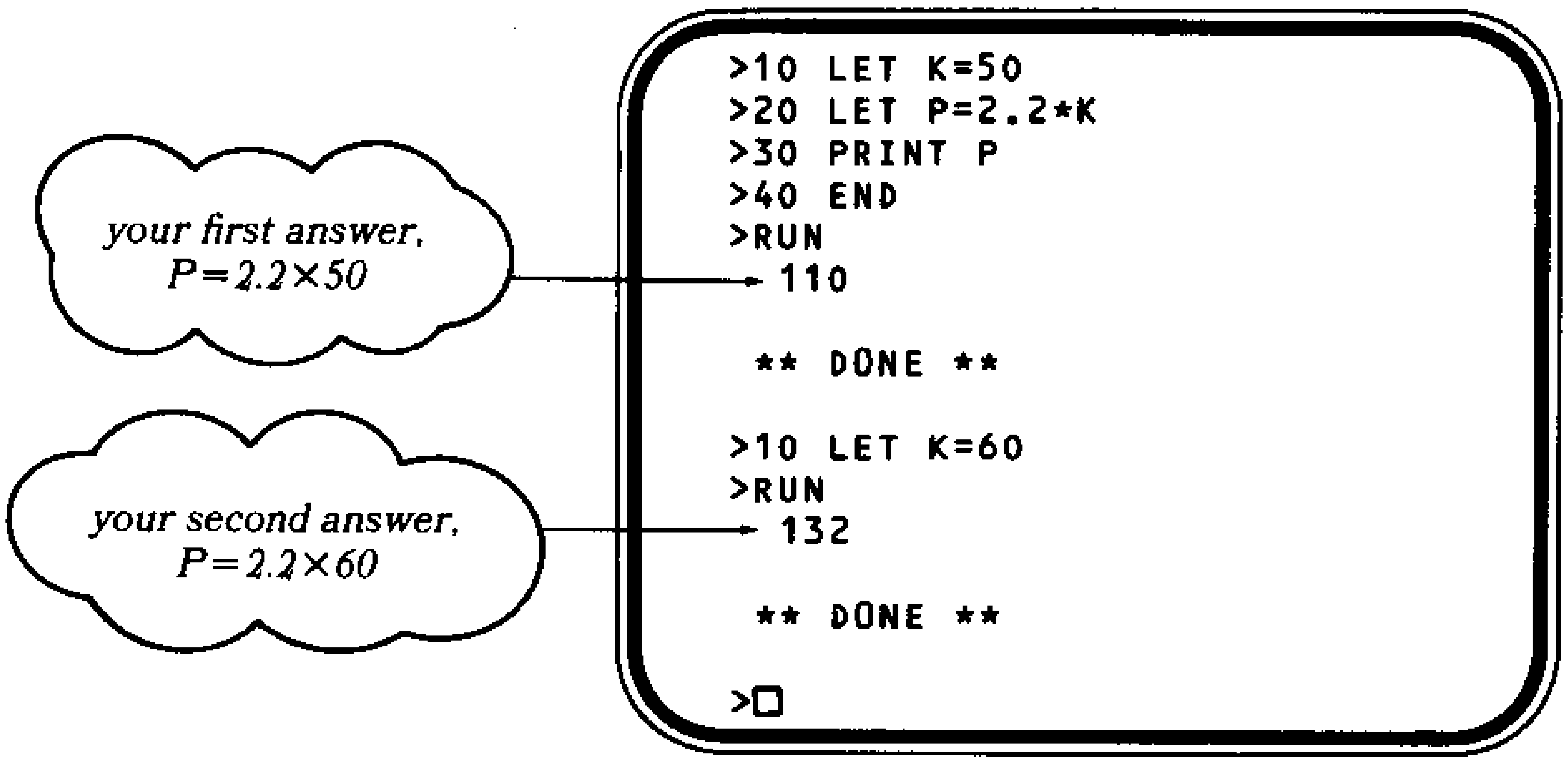


```
TI BASIC READY

>10 LET K=50
>20 LET P=2.2*K
>30 PRINT P
>40 END
>RUN
  110

 ** DONE **

>□
```

Your answer is on the screen: 50 kilograms is equal to 110 pounds. Suppose, however, that we want to find the number of pounds that are equivalent to 60 kilograms. Easy! We can do it by changing only one line – line 10. Type:

```
10 LET K=60
```

and press **ENTER**. Now type RUN and press **ENTER** again.



```
>10 LET K=50
>20 LET P=2.2*K
>30 PRINT P
>40 END
>RUN
 110

 ** DONE **

>10 LET K=60
>RUN
 132

 ** DONE **

>□
```

Editing the Program

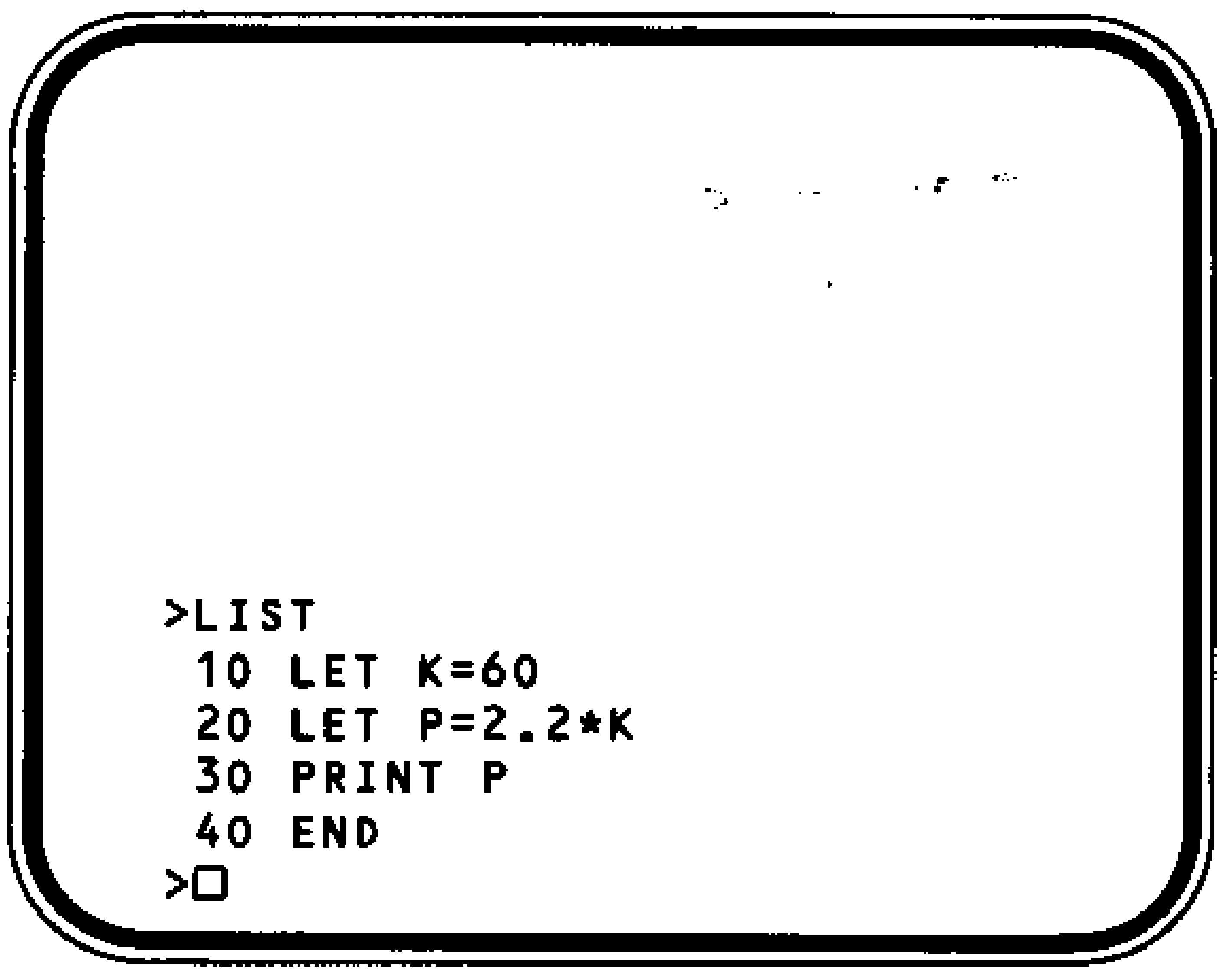
What you have just done is called "editing" a program. The ability to edit or change a program without retyping the whole thing is one you'll come to value highly as your programming skills grow. To give you an idea of the great flexibility editing adds to programming, let's experiment with a few more changes in the present program.

Adding Program Lines

We mentioned earlier that the reason we number program lines in increments of 10 (instead of 1,2,3, etc.) is to allow program lines to be added without retyping the whole program. Before we experiment with a few examples, let's clear the screen and recall our program.

Type:

```
CALL CLEAR
LIST
```



```
>LIST
 10 LET K=60
 20 LET P=2.2*K
 30 PRINT P
 40 END
>□
```

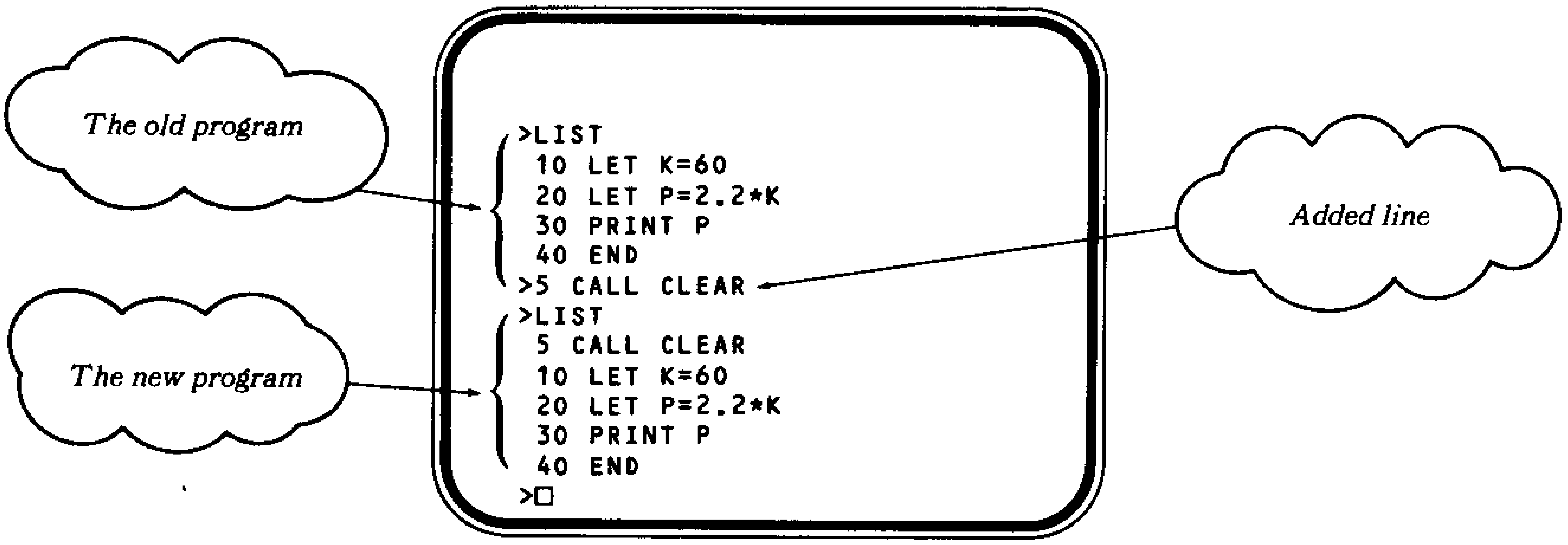
(Notice that the prompting character doesn't appear to the left of lines printed by the computer – only the lines you type are marked!)

We might want to add a CALL CLEAR statement to the program, so that we won't have to keep clearing the screen from the keyboard each time we "run" the program.

Type:

```
5 CALL CLEAR
```

Now list the program again to see the new line (type LIST and press **ENTER**).



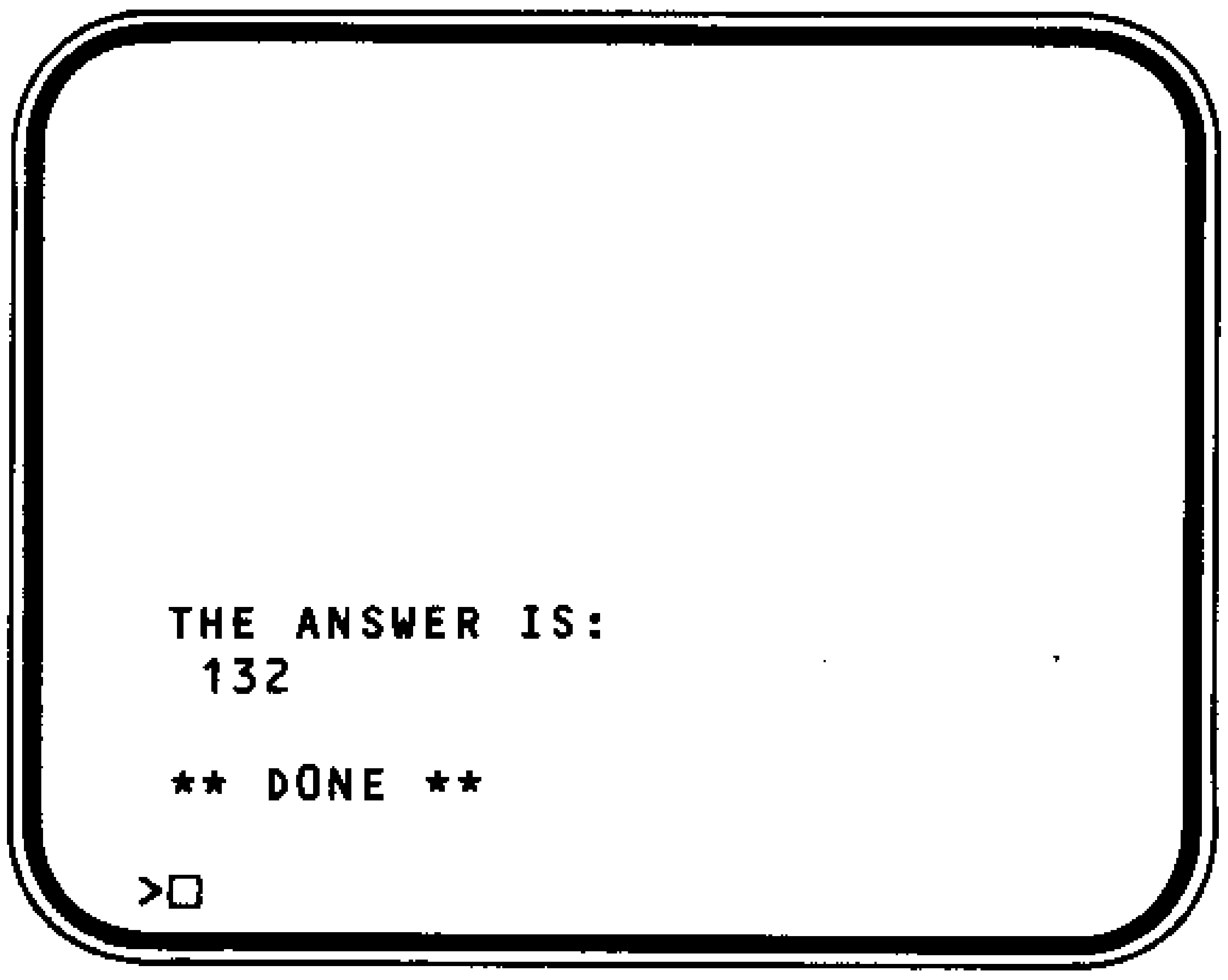
```
>LIST
 10 LET K=60
 20 LET P=2.2*K
 30 PRINT P
 40 END
>5 CALL CLEAR
>LIST
 5 CALL CLEAR
 10 LET K=60
 20 LET P=2.2*K
 30 PRINT P
 40 END
>□
```

Compare the two programs on the screen, and notice that the computer has automatically placed the new line in its proper order. Run the program again to see the effect of the added line.

Now let's add another line that will help to point out our answer. Type:

```
27 PRINT "THE ANSWER IS:"
```

and press **ENTER**. When you run the program again, you'll see this:



```
THE ANSWER IS:
 132

** DONE **

>□
```

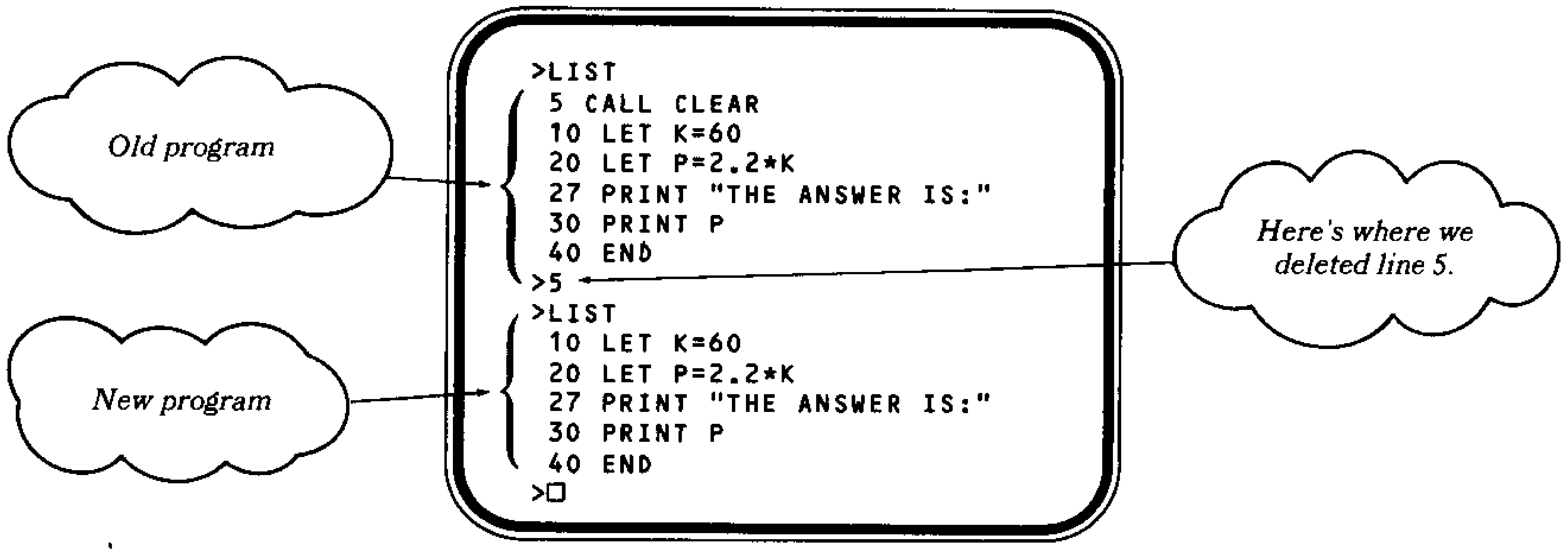
Removing Program Lines

Quite often it's necessary to remove a line or lines from a program. Deleting a program line is a very simple procedure.

The program we have stored right now doesn't really have any lines we want to delete. Just for practice, however, let's remove line 5.

First, clear the screen and list the program as it is now. Line 5 is the first line of the program, a CALL CLEAR statement. To remove it, simply type **5** and press **ENTER**.

Then list the program again. Presto! Line 5 is gone!



```
>LIST
 5 CALL CLEAR
 10 LET K=60
 20 LET P=2.2*K
 27 PRINT "THE ANSWER IS:"
 30 PRINT P
 40 END
>5
>LIST
 10 LET K=60
 20 LET P=2.2*K
 27 PRINT "THE ANSWER IS:"
 30 PRINT P
 40 END
>□
```

That's all there is to it. To remove a line, type the line number and press **ENTER**. The computer will then delete the line from program memory.

Since we really need line 5 in this program, let's reenter it. Type

```
        5 CALL CLEAR
```

and press **ENTER**.

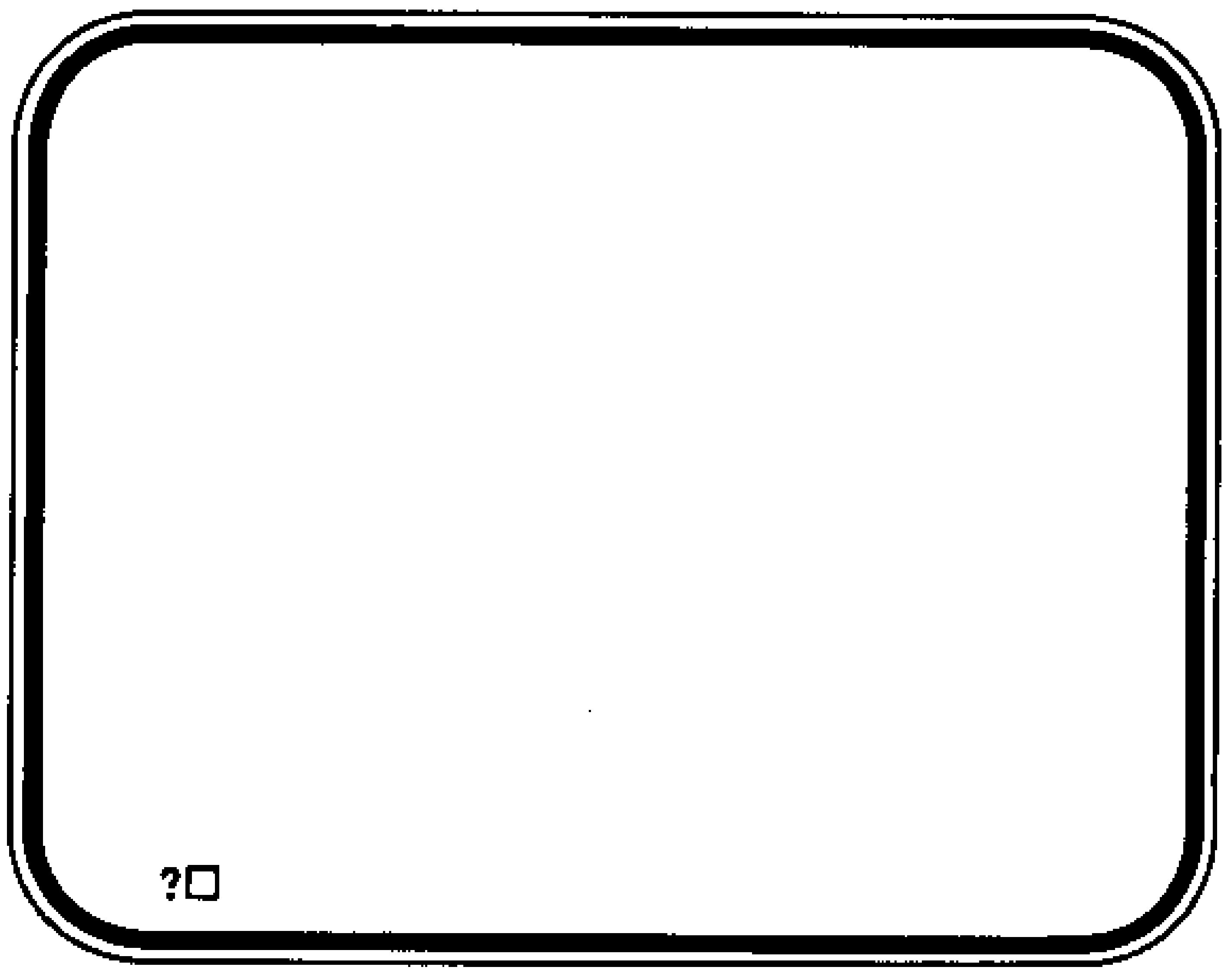
The INPUT Statement

You've already seen that you can easily change the value of K by simply retyping line 10 to assign a new value. But suppose you had many values for K, and you wanted to find the equivalent value of P for each one. It would get rather tiresome to retype line 10 each time.

There is a better way to edit line 10. An INPUT statement causes the computer to type a question mark and stop, waiting for you to type in a value and press **ENTER**. The value you enter is then assigned to the variable contained in the INPUT statement. For example, type

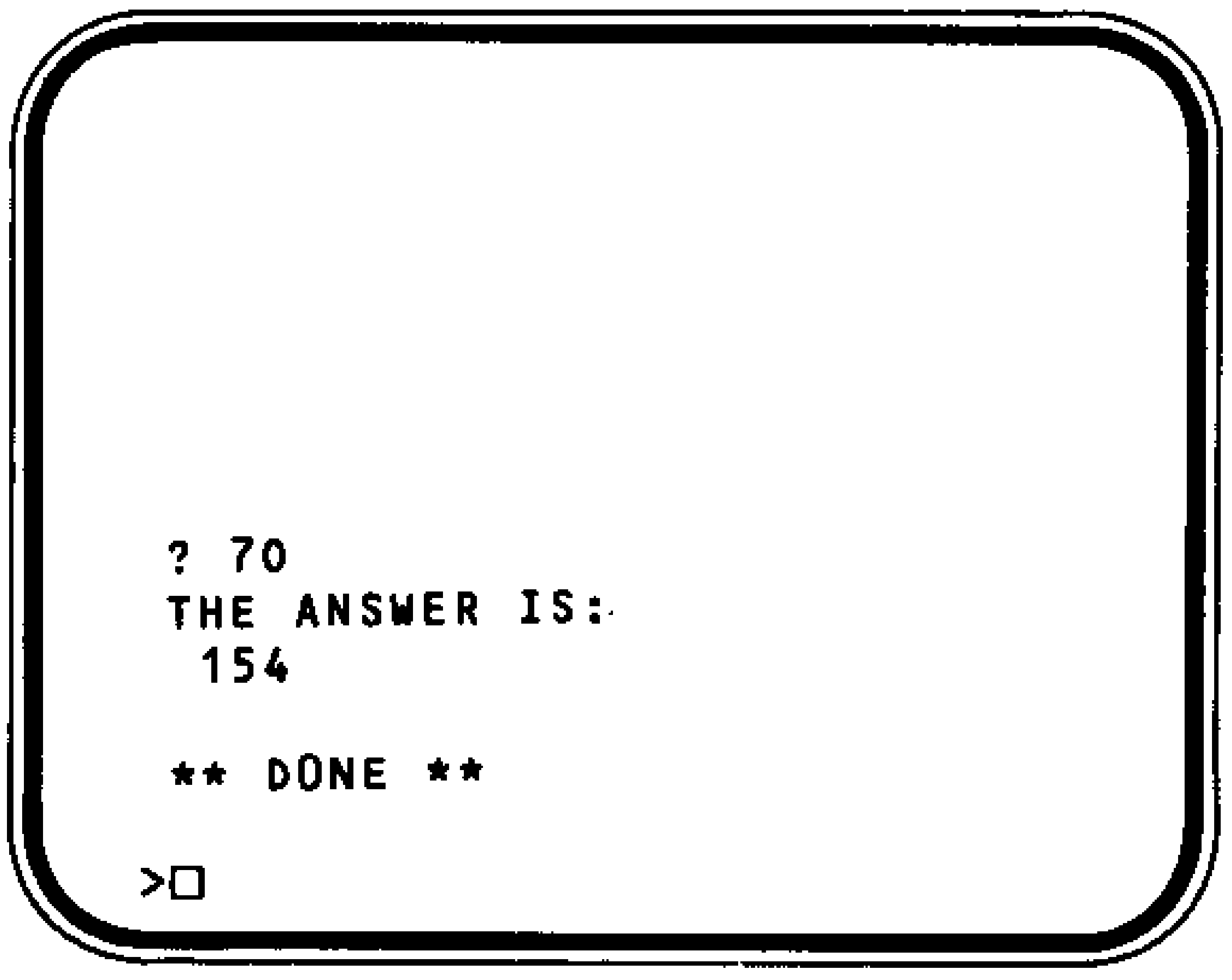
```
        10 INPUT K
```

and press **ENTER**. Now run the program again.



```
?□
```

The question mark and cursor show you that the computer is waiting for you to "input" a value for K. This time we'll let K=70, so type *70* and press **ENTER**. The computer prints your answer:



```
? 70
THE ANSWER IS:
 154

** DONE **

>□
```

Now you can run the program as many times as you like, changing the value of K each time the computer prints a question mark and stops. Try the program several times with different values for K.

The INPUT statement can also be used to print a "prompting" message (instead of a question mark) that helps you remember what value the computer is asking for. Change line 10 again by typing

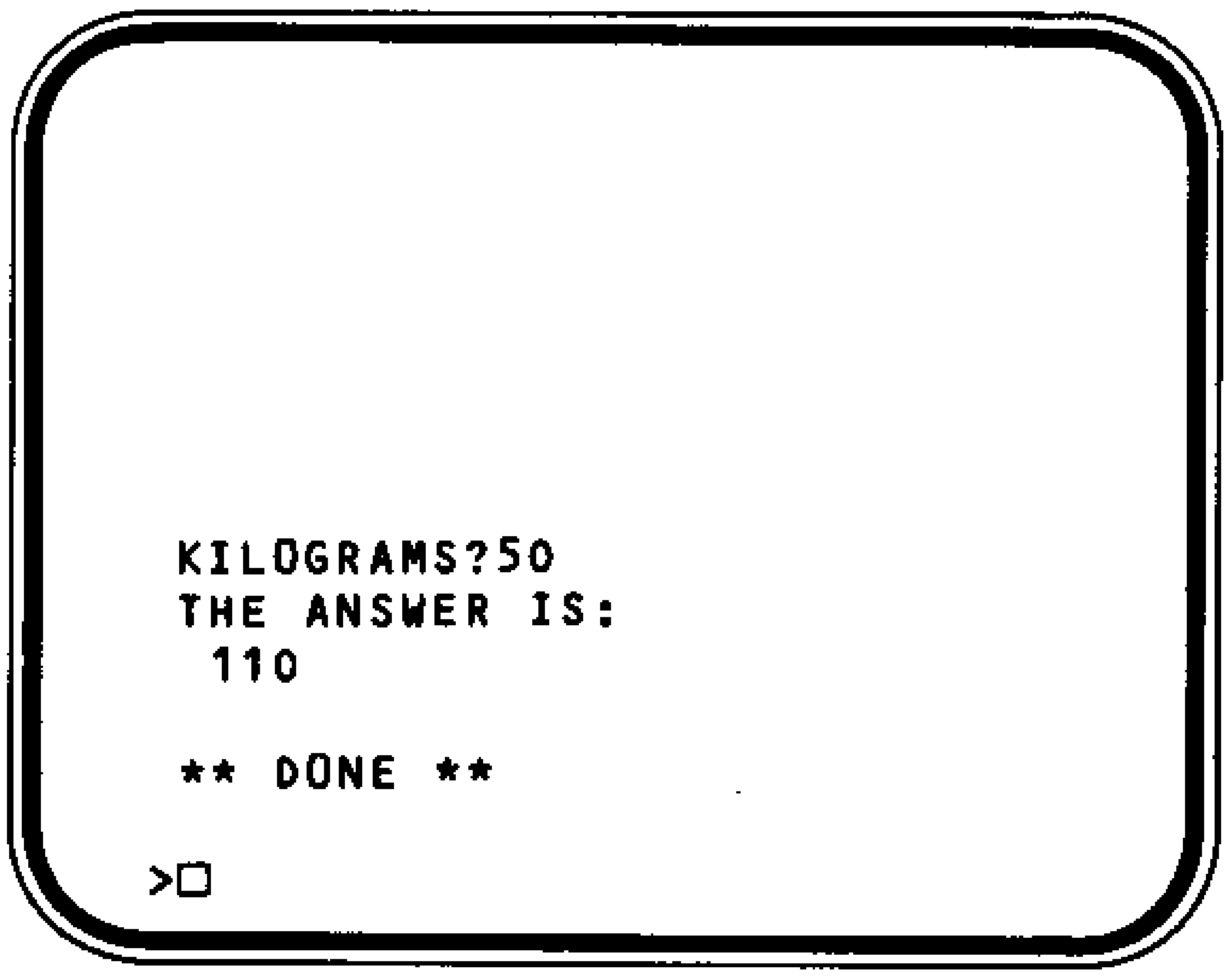
```
10 INPUT "KILOGRAMS?":K
```

colon

and pressing **ENTER**. Now run the program again. First the program asks:

```
KILOGRAMS?
```

Let's let K=50 this time. Type *50* and press **ENTER**.



```
KILOGRAMS?50
THE ANSWER IS:
 110

** DONE **

>□
```

By now, your program looks like this:

```
5 CALL CLEAR
10 INPUT "KILOGRAMS?":K
20 LET P=2.2*K
27 PRINT "THE ANSWER IS:"
30 PRINT P
40 END
```

If you'd like, you can list it on the screen at this time and review the changes you've made so far. When you're ready, we'll go on to look at one more change.

String Variables

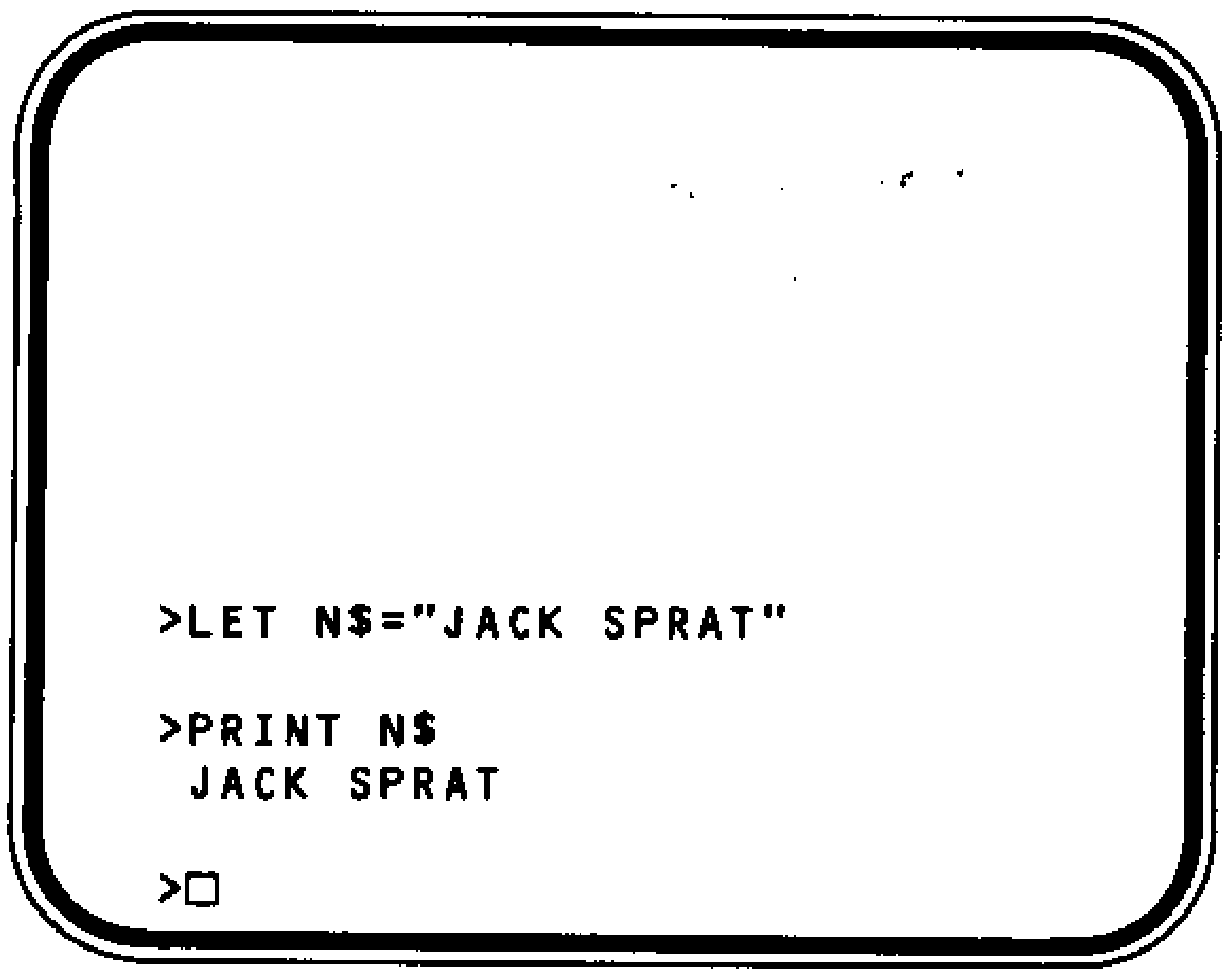
You already know what *numeric variables* are: numeric values assigned to names (variables), like "K=50." A string variable is a combination of characters (letters and numbers, or other symbols) assigned to a name. String variables differ from numeric variables in these ways:

1. The variable name *must* end with a $.
2. The alphanumeric characters in the "string" *must* be enclosed in quotation marks.
3. "Strings" of numbers cannot have arithmetic operations performed with or upon them.

Let's try a couple of examples in the Immediate Mode before changing the program. (Note that this does not interfere with the program stored in memory!)

Clear the screen (CALL CLEAR) and enter this:

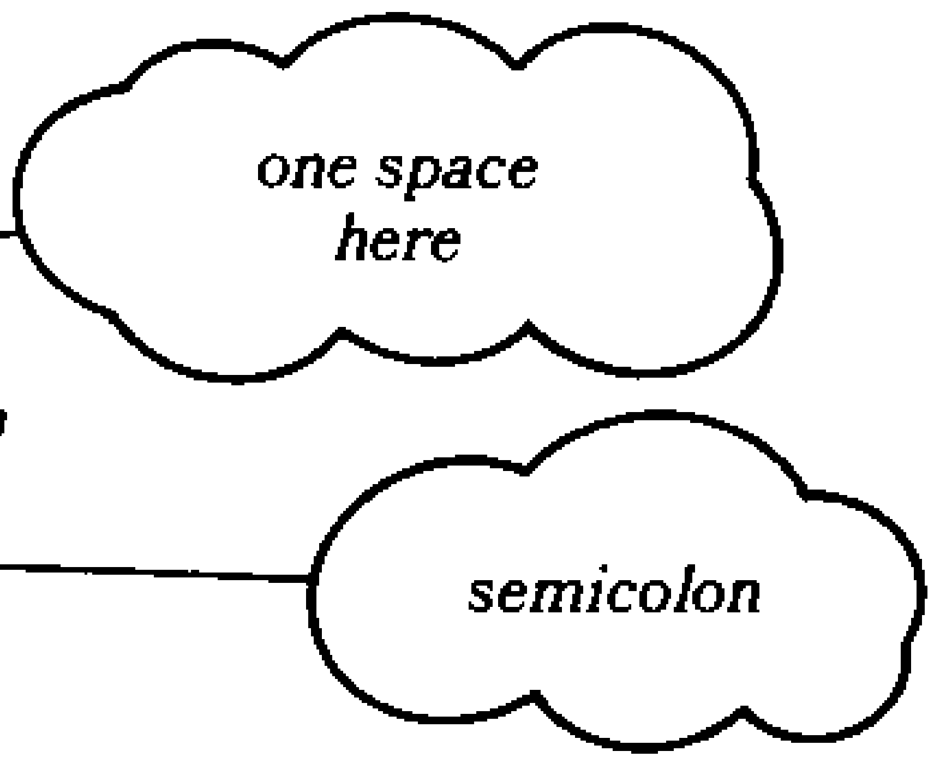
```
LET N$="JACK SPRAT"

PRINT N$
```

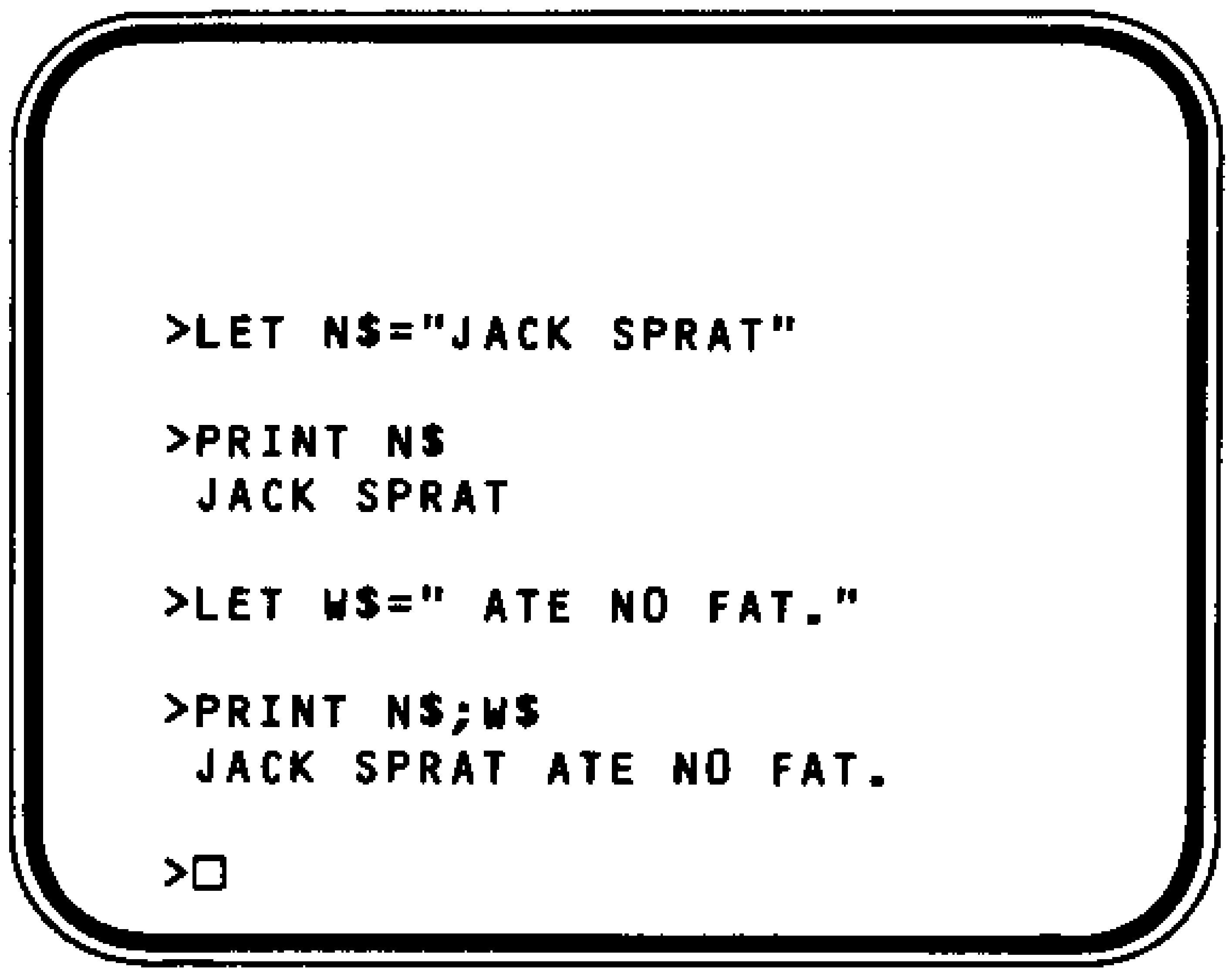


```
>LET N$="JACK SPRAT"

>PRINT N$
 JACK SPRAT

>□
```

Now type:

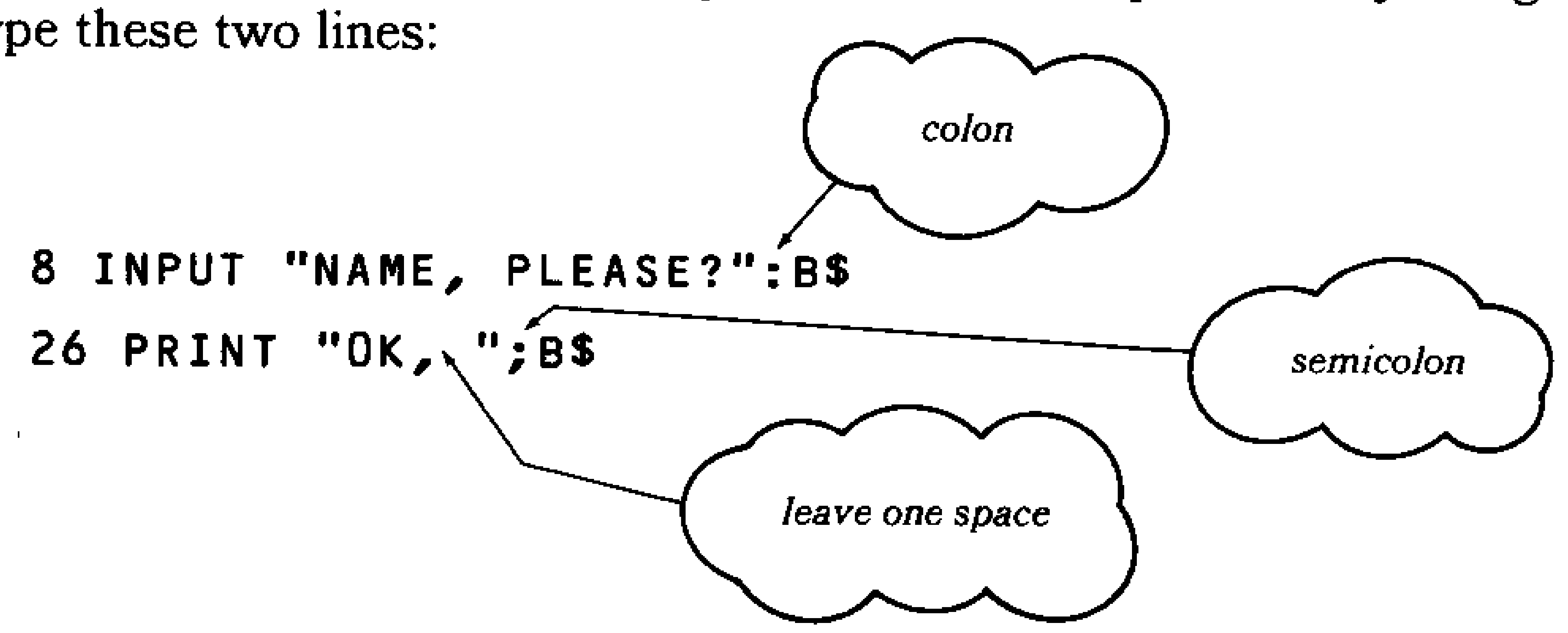
```
LET W$=" ATE NO FAT."

PRINT N$;W$
```





```
>LET N$="JACK SPRAT"

>PRINT N$
 JACK SPRAT

>LET W$=" ATE NO FAT."

>PRINT N$;W$
 JACK SPRAT ATE NO FAT.

>□
```

Let's make your conversion program a little more personal by using a string variable. Type these two lines:



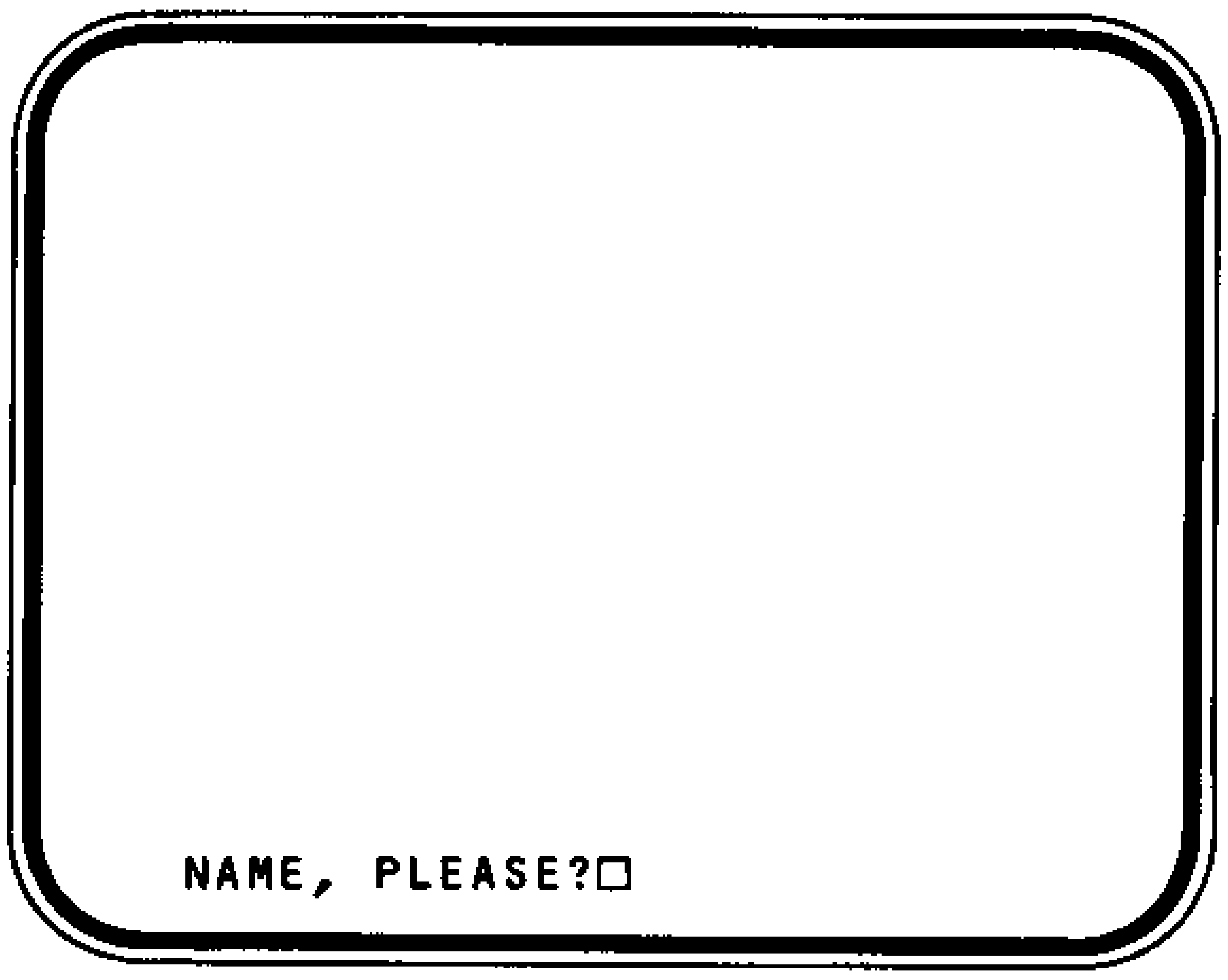
```
8 INPUT "NAME, PLEASE?":B$
26 PRINT "OK, ";B$
```

(Clear the screen and list the program again so you can see how the new lines fit in.)

When you run the program this time, the two INPUT statements will stop the program twice:

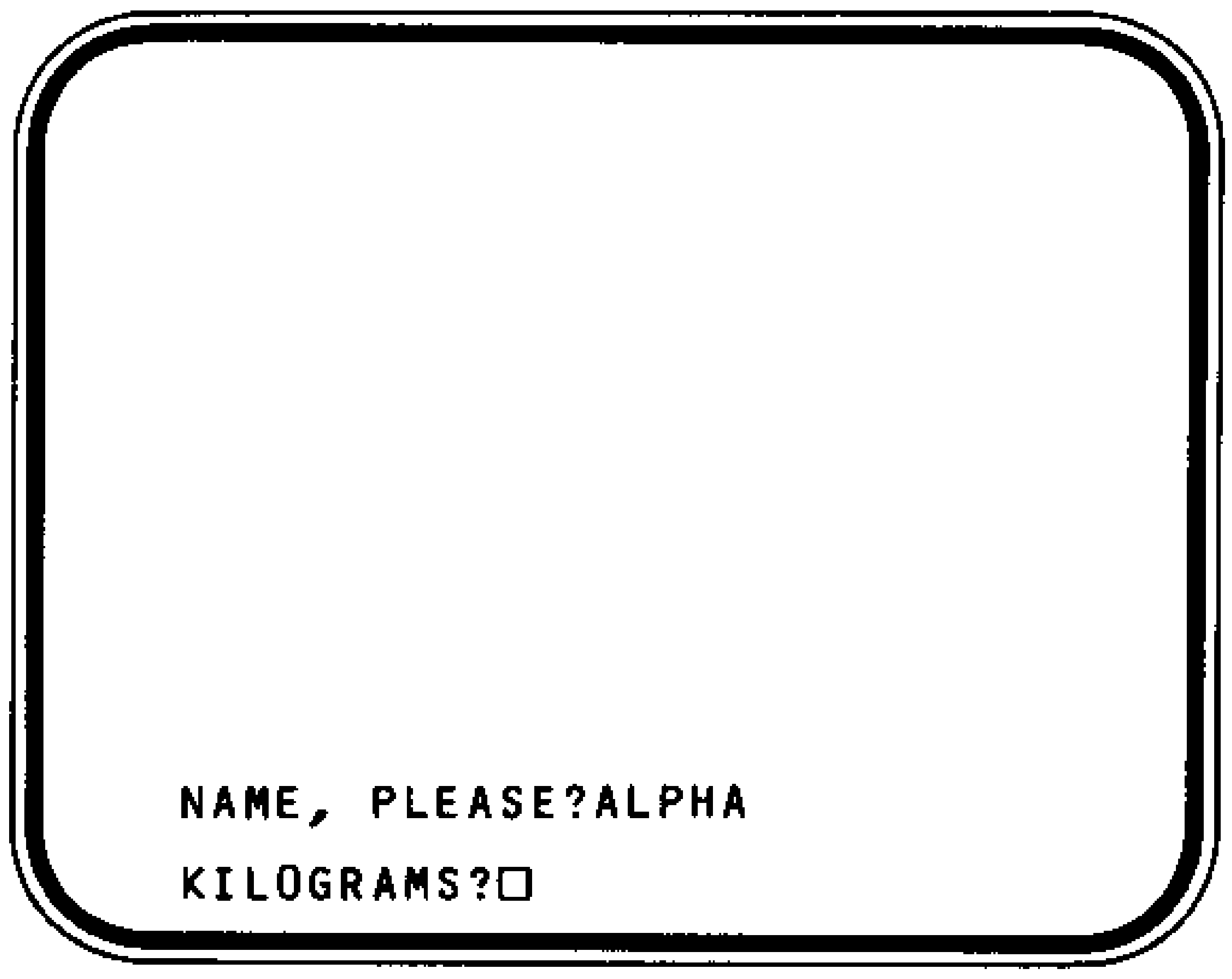
The computer asks	*You type in*
NAME, PLEASE?□	Your name and then press **ENTER**.
KILOGRAMS?□	The number of kilograms and then press **ENTER**.

Let's try it. Type RUN and press **ENTER**.



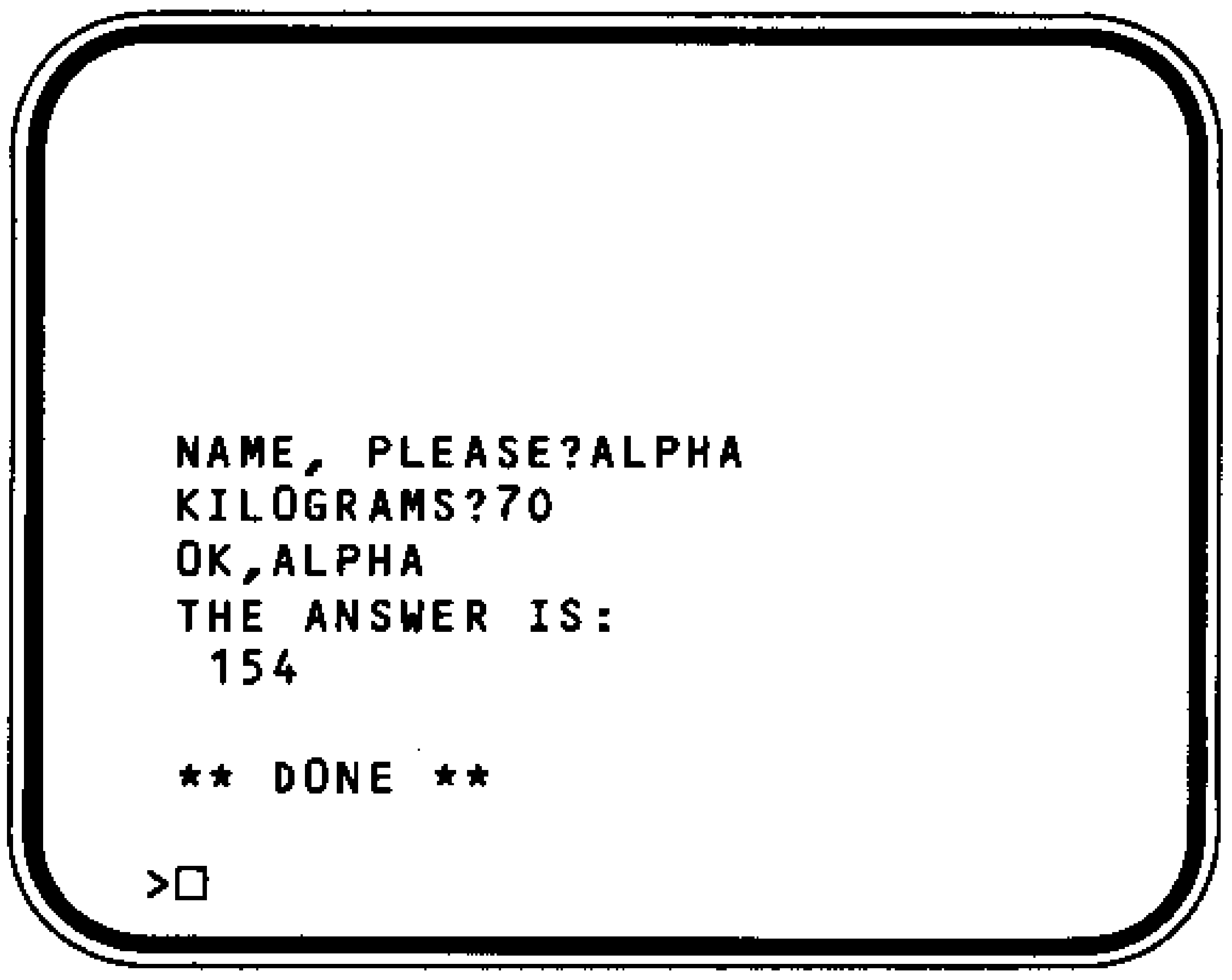
```
NAME, PLEASE?□
```

We'll type in Alpha (that's a nice name) and press **ENTER**. Then we'll see



```
NAME, PLEASE?ALPHA
KILOGRAMS?□
```

Again let's type *70* for the number of kilograms. Press **ENTER** again and you'll see:



```
NAME, PLEASE?ALPHA
KILOGRAMS?70
OK,ALPHA
THE ANSWER IS:
 154

** DONE **

>□
```

String variables can save a lot of typing when you're using a message (a name or a prompting statement, for example) more than once in a program.

Now list your program and review these latest changes. We've given you a lot of information, and we've given it pretty quickly. This would be a good time for you to do a little experimenting on your own, trying out some of the things you've learned.

Experiment!

Want a challenge? Try writing another conversion program – one that converts a temperature in degrees Fahrenheit (F) to degrees Celsius (C). The conversion formula is

Degrees C = 5/9 (Degrees F − 32)

Don't forget to use INPUT statements and CALL CLEAR at appropriate places! Hint: Let C = 5/9*(F − 32) – the parentheses must be there in your program!

2

The GO TO Statement

So far, you've been developing programs that operate from beginning to end in a straight sequential order. There are many situations, however, in which you want to interrupt this orderly flow of operation. Look at the following program, but don't enter it yet:

```
10 CALL CLEAR
20 INPUT K
30 PRINT K
40 PRINT
50 K=K+1
60 GO TO 30
```

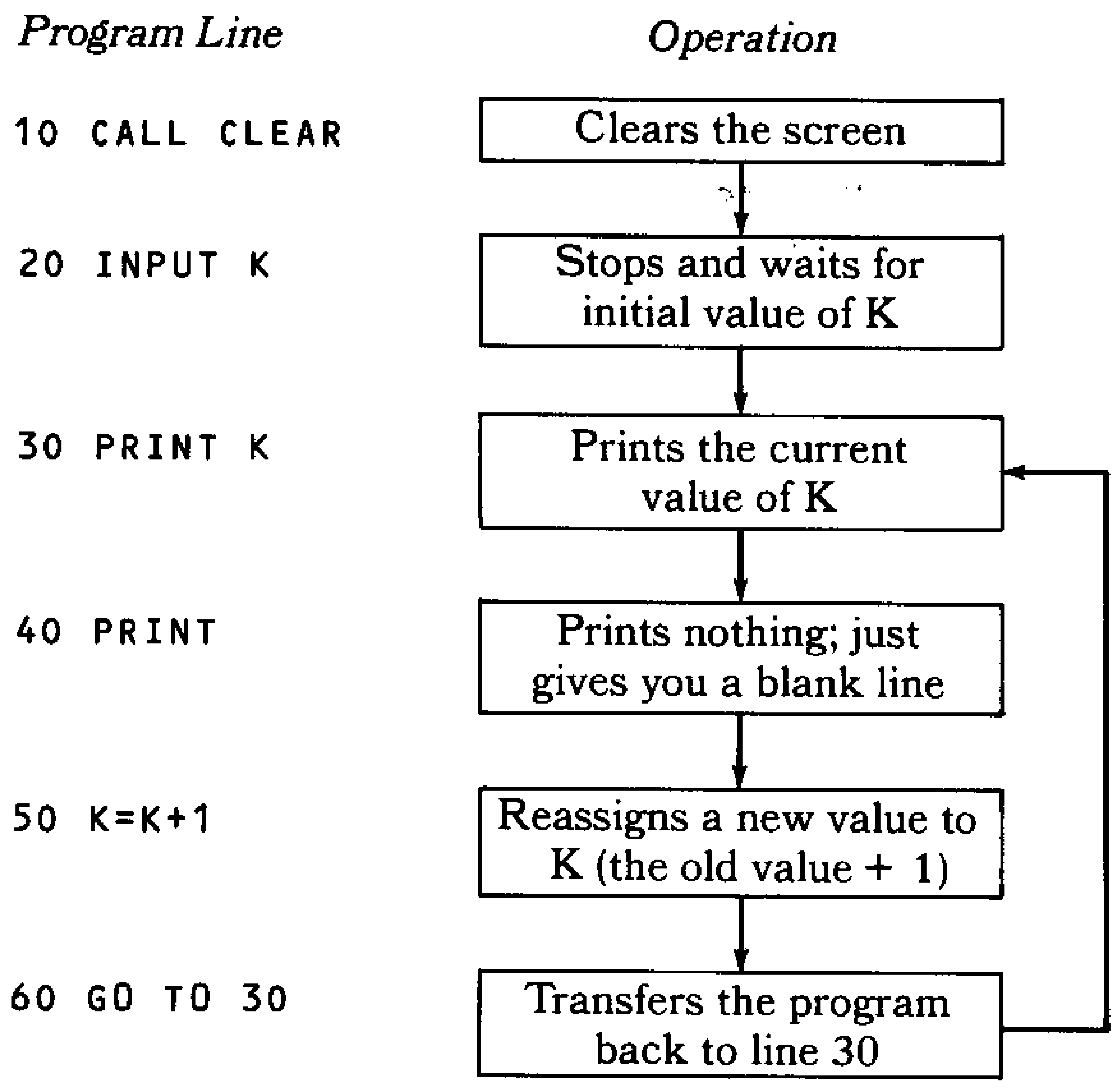
Here we "send" the program back to line 30 by using a GO TO statement in line 60. The GO TO statement causes the actions performed by lines 30, 40, and 50 to be repeated over and over again, setting up what's called a *loop.* (Notice that we don't use an END statement. That's because the program will never get beyond line 60! It won't stop until you tell it to by pressing **CLEAR.** This is called an "endless loop.")

Let's enter the program now. First, type NEW and press **ENTER** to erase the computer's memory, and then type these lines:

```
10 CALL CLEAR
20 INPUT K
30 PRINT K
40 PRINT
50 K=K+1
60 GO TO 30
```

Note that LET is optional.

Before you run the program, we'll examine a diagram called a *flowchart,* explaining how the program works.


Program Line
Operation
10 CALL CLEAR
Clears the screen
20 INPUT K
Stops and waits for initial value of K
30 PRINT K
Prints the current value of K
40 PRINT
Prints nothing; just gives you a blank line
50 K=K+1
Reassigns a new value to K (the old value + 1)
60 GO TO 30
Transfers the program back to line 30

Now run the program, putting in 1 for the beginning value of K. Watch how quickly the computer counts — almost too fast to follow! That's why we added the blank line (line 40). This line spaces out the numbers a bit so that you can see them better.

Let the computer count as long as you want to. When you are ready to stop the program, press **CLEAR.** You'll see *BREAKPOINT AT (#) on the screen, indicating where the program stopped. Run the program as many times as you want, using whatever number you wish as the initial value for K (50,100,500,etc.).

GO TO can be typed as GOTO in your program. The computer isn't fussy about that. If you try to send the program to a non-existent line number, however, you'll get an error message.

Suppose, for example, we type in

```
60 GO TO 25
```

and press **ENTER**. Try it, run the program, and see what happens! You'll see this error message:

```
* BAD LINE NUMBER IN 60
```

So correct the line by typing and entering

```
60 GO TO 30
```

and run the program again.

Can we change the program to make it count by 2's, or 5's? You bet we can! By making one program change, let's make the computer count by 2's: Type:

```
50 K=K+2
```

and press **ENTER**. Now run the program, typing in 2 when the computer asks for the starting value of K.

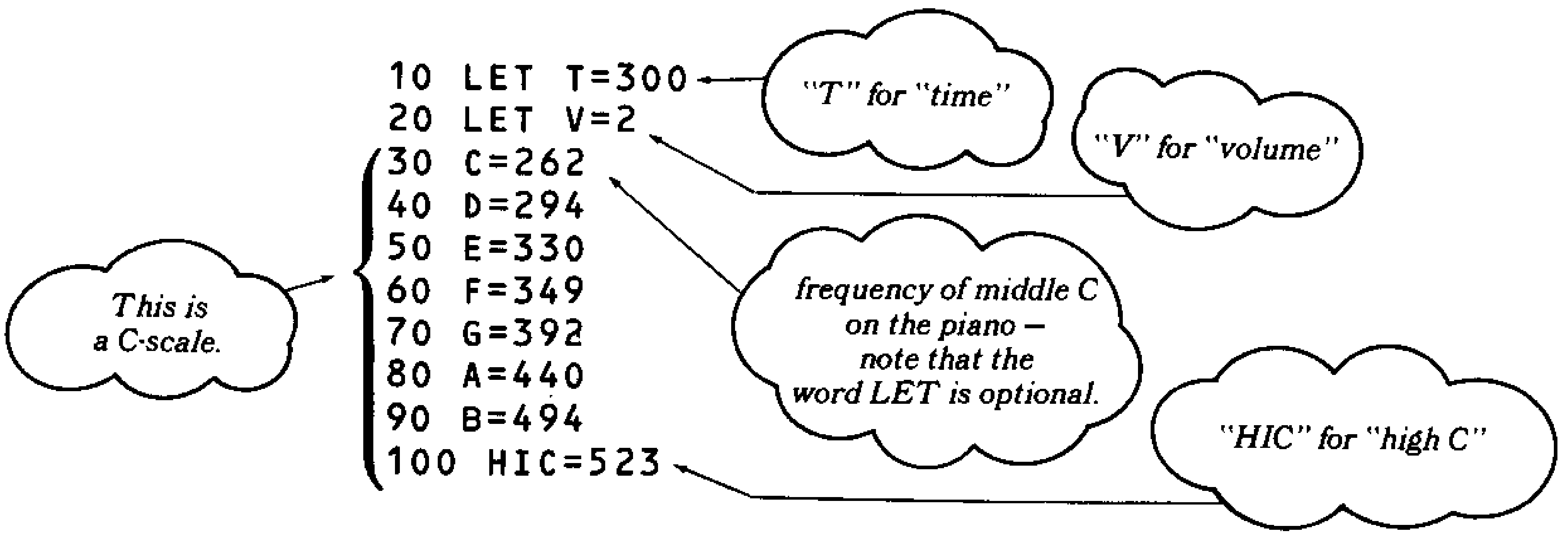
Experiment with the program for a while, making it count by 3's, 5's, 10's, etc.

A GO TO Loop With CALL SOUND

GO TO *loops* have many applications, of course, beyond simple counting. We could use a loop, for example, to practice a musical scale.

Before we start the program, you might want to review the CALL SOUND section in Chapter 1 (see page 17) to help you remember how the CALL SOUND statement performs in the Immediate Mode. (It behaves essentially the same way in a program.)

When you're ready to start the program, type NEW and press **ENTER**. Our first task in the program will be to assign values to the variables we'll use. Type these lines:



```
10 LET T=300
20 LET V=2
30 C=262
40 D=294
50 E=330
60 F=349
70 G=392
80 A=440
90 B=494
100 HIC=523
```

"T" for "time"
"V" for "volume"
This is a C-scale.
frequency of middle C on the piano – note that the word LET is optional.
"HIC" for "high C"

Now you're ready for the CALL SOUND statements to tell the computer when to play each note:

```
200 CALL SOUND(T,C,V)
300 CALL SOUND(T,D,V)
400 CALL SOUND(T,E,V)
500 CALL SOUND(T,F,V)
600 CALL SOUND(T,G,V)
700 CALL SOUND(T,A,V)
800 CALL SOUND(T,B,V)
900 CALL SOUND(T,HIC,V)
```

Finally, set up the loop with a GO TO statement:

```
950 GOTO 200
```

Check the program now for errors, and correct any that you find. When everything is correct, run the program. Again, this is an endless loop. (Notice that the screen background stays light green until you stop the program.) You'll have to press **CLEAR** to stop it.

Experiment!

Practice building other musical scales and patterns, using the note frequencies listed in *Appendix A*.

A GO TO Loop with CALL COLOR

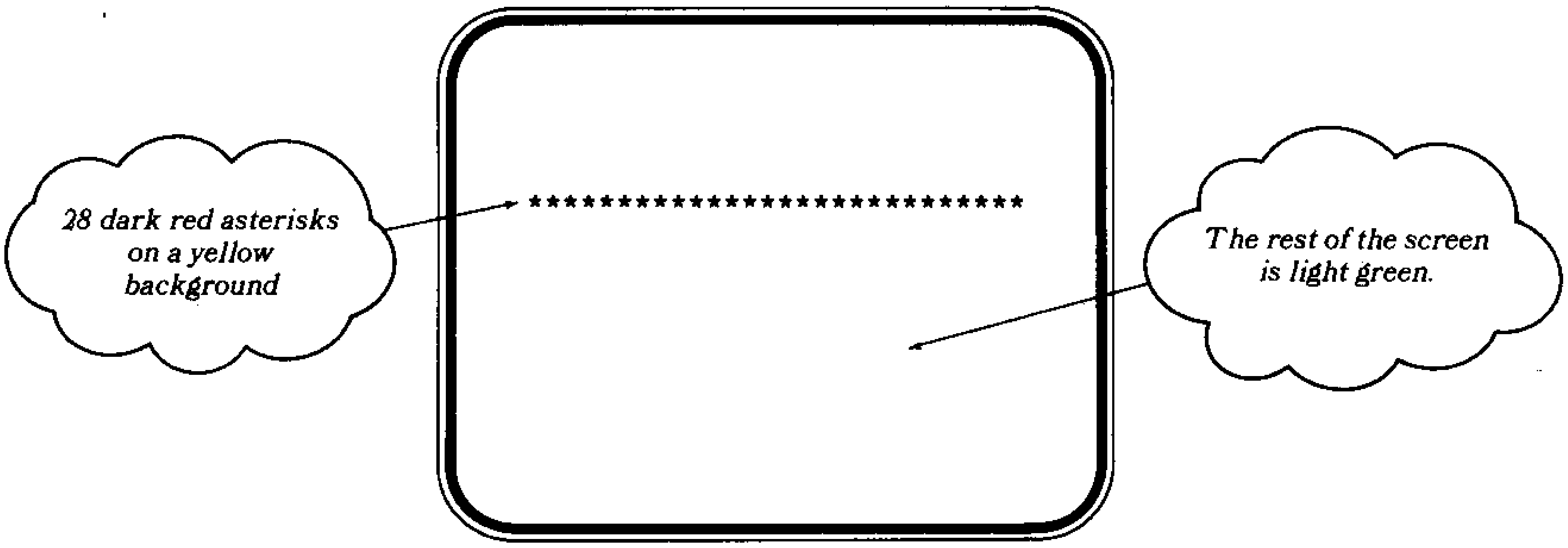
Up to now, you've seen only three colors in BASIC on your display screen. (Maybe you've only noticed two – but there really are three.) First, while you're entering a program, the screen background is cyan (a light blue color), and the characters (letters and numbers) that you're typing are black. Then, while the program is running, the screen becomes a light green color. When the program stops, the screen returns to cyan with black characters.

These are only three of the sixteen colors available with your computer, and the way you control the colors in a program is through the CALL COLOR statement. Let's try a program with a CALL COLOR statement and a slightly different GO TO loop. Clear your old program from the computer's memory (NEW; press **ENTER**), and type these lines:

```
10 CALL CLEAR
20 CALL COLOR(2,7,12)
30 CALL HCHAR(12,3,42,28)
40 GO TO 40
```

A GO TO loop that "goes to" itself!

Now run the program, and the screen should look like this:


28 dark red asterisks on a yellow background
****
The rest of the screen is light green.

Our program prints twenty-eight asterisks across the screen. The asterisks are dark red. In the area where the asterisks are displayed, the screen color is a light yellow. The rest of the screen remains light green.

(Line 40 puts your program into a holding pattern that keeps your graphic on the screen. When you're ready to stop the program, press **CLEAR** to break the loop. Remember, you can run the program as many times as you like.)

A CALL COLOR statement requires three numbers, enclosed in parentheses and separated by commas:

```
20 CALL COLOR(2,7,12)
```

The first number after the open parenthesis symbol is a *character set number*. As we mentioned in Chapter 1, each *character* (letters, numbers, and symbols) that prints on the screen has its own numeric code, ranging from 32 through 127 for a total of ninety-six characters. These characters are organized by the computer into twelve *sets* with eight characters in each:

Set #1		**Set #2**		**Set #3**		**Set #4**	
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	?

Set #5		**Set #6**		**Set #7**		**Set #8**	
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	_

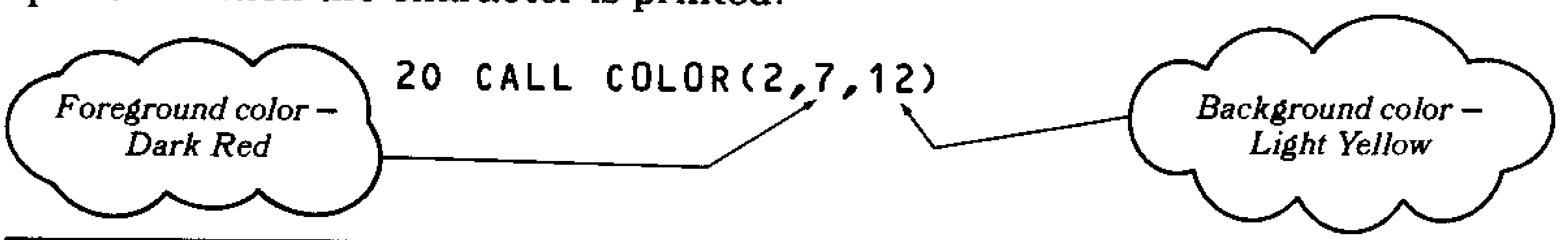
Set #9		**Set #10**		**Set #11**		**Set #12**	
Code	*Character*	*Code*	*Character*	*Code*	*Character*	*Code*	*Character*
96	`	104	H	112	P	120	X
97	A	105	I	113	Q	121	Y
98	B	106	J	114	R	122	Z
99	C	107	K	115	S	123	{
100	D	108	L	116	T	124	¦
101	E	109	M	117	U	125	}
102	F	110	N	118	V	126	~
103	G	111	O	119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

The set number you use in a CALL COLOR statement, then, is determined by the character you want to print. (And what happens if you want to print characters from different sets in the same colors? We'll discuss that in a few minutes.)

The second and third numbers in parentheses determine the colors used in your graphic. Each of the sixteen colors has its own numeric code.

Color	*Code #*	*Color*	*Code #*
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

The second number sets the *foreground* color; that is, the color of the character you designate. The third number sets the *background* color – the color of the block or square in which the character is printed.

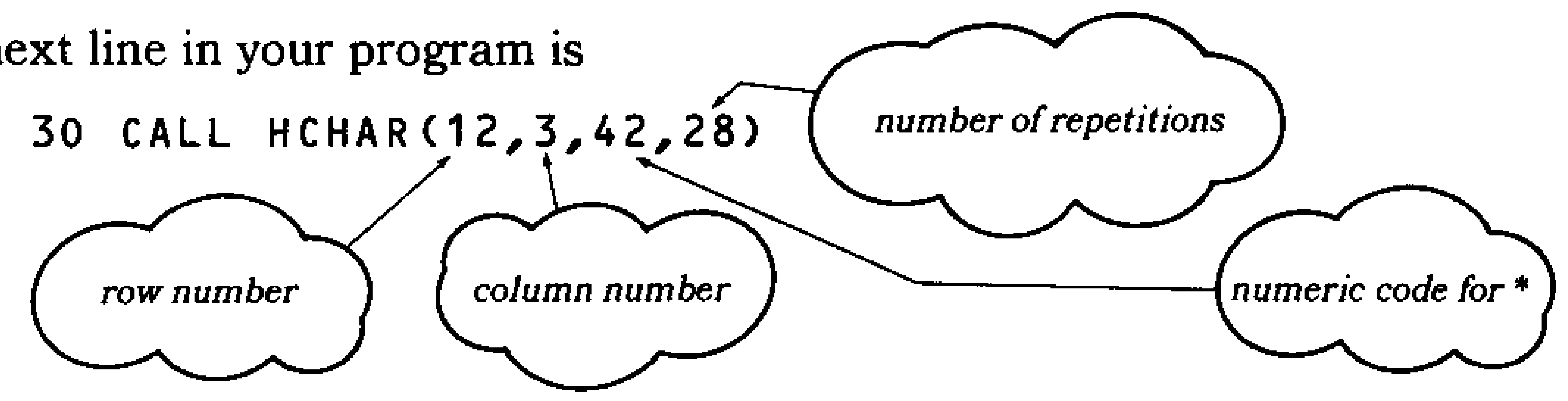


```
20 CALL COLOR(2,7,12)
```

Foreground color – Dark Red
Background color – Light Yellow

The next line in your program is

```
30 CALL HCHAR(12,3,42,28)
```



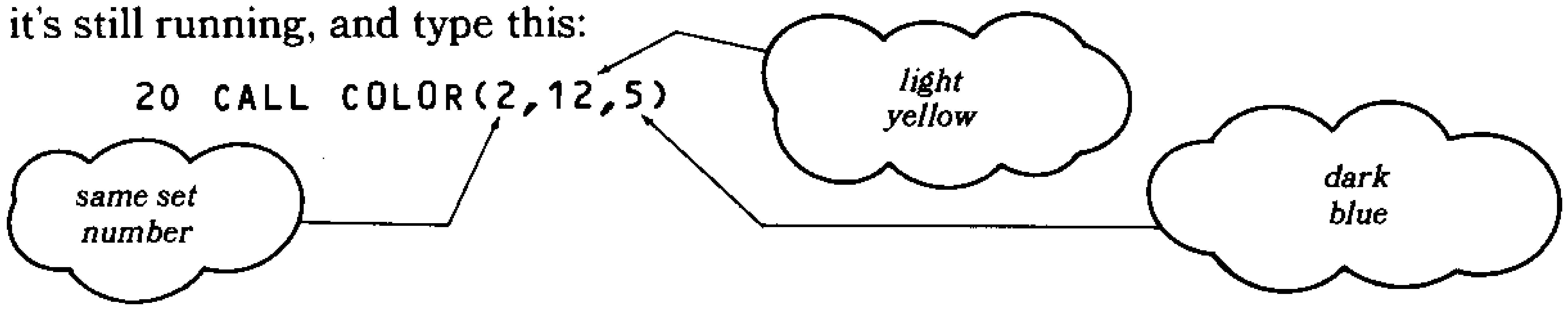
(If you need to review the CALL HCHAR examples in Chapter 1, this would be a good time to do it.)

Now you know why we indicated Set #2 in our CALL COLOR statement! The asterisk (code number 42) is a part of Set #2.

Line 40 of the program is a GO TO statement that "goes to" itself. It keeps the computer "idling" until you press **CLEAR.** When you do, the program stops, and the screen changes back to its normal color. All the reds, yellows, and greens disappear.

Now let's change line 20 of the program to see some new colors. Stop the program, if it's still running, and type this:

```
20 CALL COLOR(2,12,5)
```

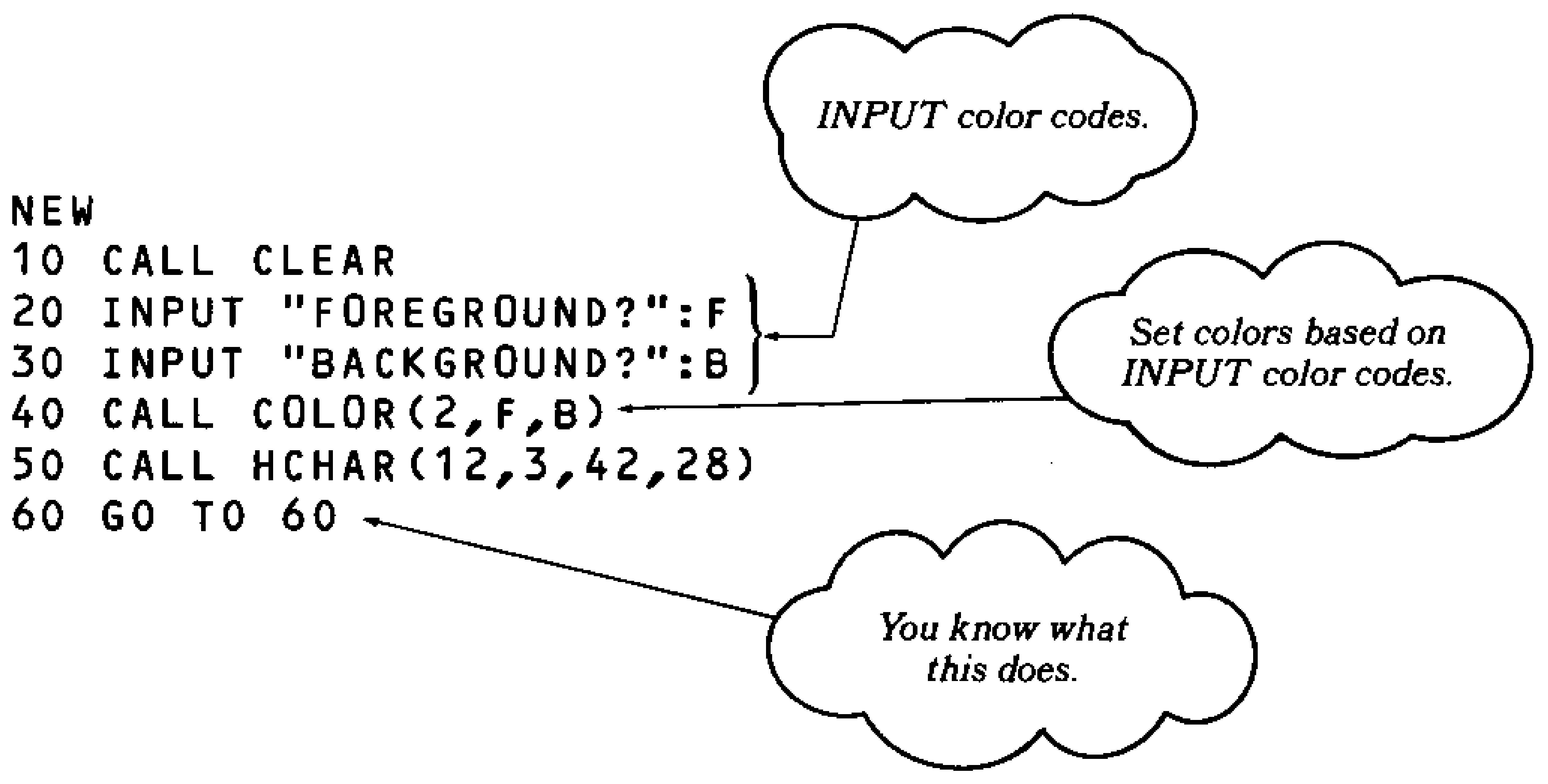


Press **ENTER** to store your new line, and list the program (LIST; press **ENTER**) to review your program.

When you're ready, run the program. You'll see 28 light yellow asterisks against a dark blue background this time.

You could, of course, continue to experiment by stopping the program, entering a new line 20 and running the modified program over and over. Don't. Instead, save wear and tear on your fingers by entering the following program which allows you to experiment more easily. With this program, you enter foreground (F) and background (B) colors in response to INPUT statements.

```
NEW
10 CALL CLEAR
20 INPUT "FOREGROUND?":F
30 INPUT "BACKGROUND?":B
40 CALL COLOR(2,F,B)
50 CALL HCHAR(12,3,42,28)
60 GO TO 60
```

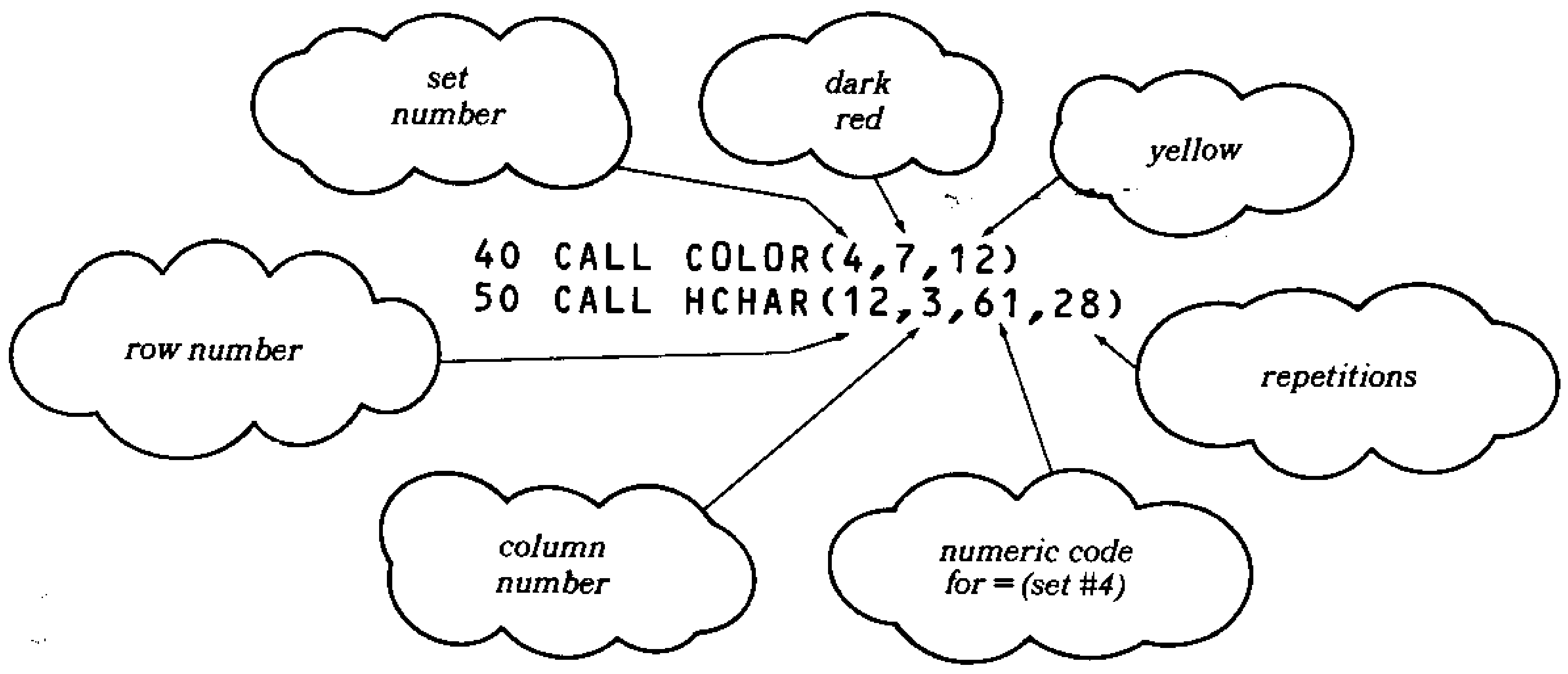


When the computer asks you for the "foreground" and "background" colors you want to use, you can type in any color code from *1* through *16*. Remember, however, that color number 1 is transparent and color 4 is the screen color when the program is running. These may not be satisfactory in this program. (Also, color number 2, black, can cause display distortion on some screens.) Here are some combinations you might find interesting:

Foreground Color	*Background Color*
3	16
2	11
5	16
2	14
7	15
7	12
13	12
14	16

OK, have you checked your program for typographical errors? Have you chosen the foreground and background colors you want to use first? Then run the program.

After you've experimented with different color combinations, you might enjoy trying some other characters, as well. You can do this by retyping line 50, substituting a different character code number for the "42" (asterisk code number). Just remember, if you select a character from any set other than #2, you'll also have to change line 40 to reflect the new set number. For example:


```
40 CALL COLOR(4,7,12)
50 CALL HCHAR(12,3,61,28)
```

Earlier we brought up this subject: What happens if you want to print characters from different sets, all in the same color? One way to do this is to include in your program twelve CALL COLOR statements – one for each set of characters. You'll have to do quite a bit of typing, *but* you'll be free to use any of the characters you choose. Try the following program, which covers eight character sets.

```
NEW
100 CALL CLEAR
110 CALL COLOR(1,6,16)
120 CALL COLOR(2,6,16)
130 CALL COLOR(3,6,16)
140 CALL COLOR(4,6,16)
150 CALL COLOR(5,6,16)
160 CALL COLOR(6,6,16)
170 CALL COLOR(7,6,16)
180 CALL COLOR(8,6,16)
190 PRINT "           "
200 GOTO 200
```

Light blue

White

You decide what to do here!

Use any message you want in line 190; just remember to enclose it in quotation marks. With these CALL COLOR statements you have told the computer to print any of the sixty-four characters in light blue (6) on a white (16) background.

Experiment!

Put a little COLOR in your life! Try some experiments of your own with different colors and character sets. For example, what happens if you enter the same color code for foreground and background? Try it!

Error Messages

We haven't talked much in this chapter about error messages because, for the most part, the ones you'd run into in these program examples are the same as – or very similar to – those you learned about in Chapter 1. For example, a spelling or typing error in NEW, RUN, or LIST will cause the computer to return an "INCORRECT STATEMENT" message as soon as you press **ENTER**.

Errors in program statements may be detected by the computer either when the line is entered *or* when the program is run. Here is a sample of error conditions and messages you might see when you *enter* an incorrect line:

Condition	*Message*
Omitting a quotation mark: `10 INPUT "WHAT COLOR:F`	`* INCORRECT STATEMENT`

Below are some examples of line errors that would cause error messages when you *run* a program:

Condition	*Message*
Misspelling a statement: `10 INPT "WHAT COLOR":F`	`* INCORRECT STATEMENT IN 10`

Omitting necessary punctuation or typing an incorrect punctuation mark:
`10 INPUT "WHAT COLOR"F`
`10 INPUT "WHAT COLOR";F`	

Leaving the variable out of an INPUT statement:

```
10 INPUT "WHAT COLOR":
```

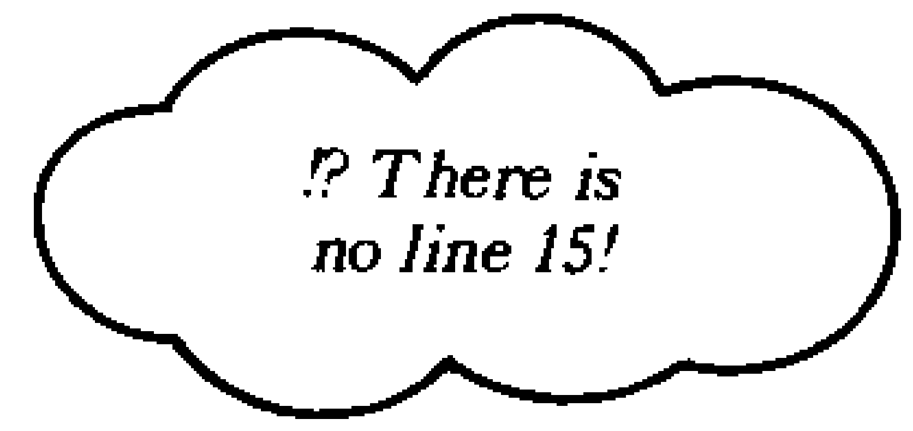
Leaving out the space between GO TO and the line number:

```
10 GO TO30
```

```
* INCORRECT STATEMENT IN 10
```

Using a non-existent line number in a GO TO statement:

```
10 LET A=5
20 PRINT A
30 GO TO 15
```



```
* BAD LINE NUMBER IN 30
```

Notice that the error messages given during a program run usually indicate the number of the troublesome line. If you'd like to view the line in question (let's say it's line 10), just type

```
LIST 10
```

and press **ENTER**. The computer will obediently print line 10 on the screen for you to review. You can also list the whole program on the screen if you prefer. Type

```
LIST
```

and press **ENTER**.

Remember, too, that failing to press **ENTER** at the end of each program line may cause the computer to give you an error message or an incorrect result, depending on the kind of operation you're performing.

Making mistakes is a normal part of learning – so don't be disturbed when the computer gives you an error message. Just list the line or the program, identify the error, retype the line correctly, and go right on your way!

(*Note:* If you'd like to see all the error messages your computer can give you, or if you don't understand a message you're given, you'll find a complete list of error messages – and when they occur – in the "BASIC Reference Section" of your *User's Reference Guide.*)

SUMMARY OF CHAPTER 2

In this chapter you've covered a lot of very important ground. You've learned how to:

- Enter a program
- Use the commands NEW, LIST, and RUN
- "Edit" or change a program
- Use INPUT statements with *numeric variables* and *string variables*
- Build a mathematical conversion program
- Create a GO TO loop within a program
- Stop an endless loop with **CLEAR**
- Use a GO TO loop in a CALL SOUND program
- Use the CALL COLOR statement and a GO TO loop in a graphics program

When you started working with Chapter 2, you were a beginner in learning BASIC and programming. Now you're well on your way to becoming an experienced computer programmer.

Quick Review of Program Structure

1. Begin each line with an identifying line number (1-32767).
2. Number the lines in the order you want the computer to follow in performing the program.
3. Press **ENTER** when you have finished typing a program line.