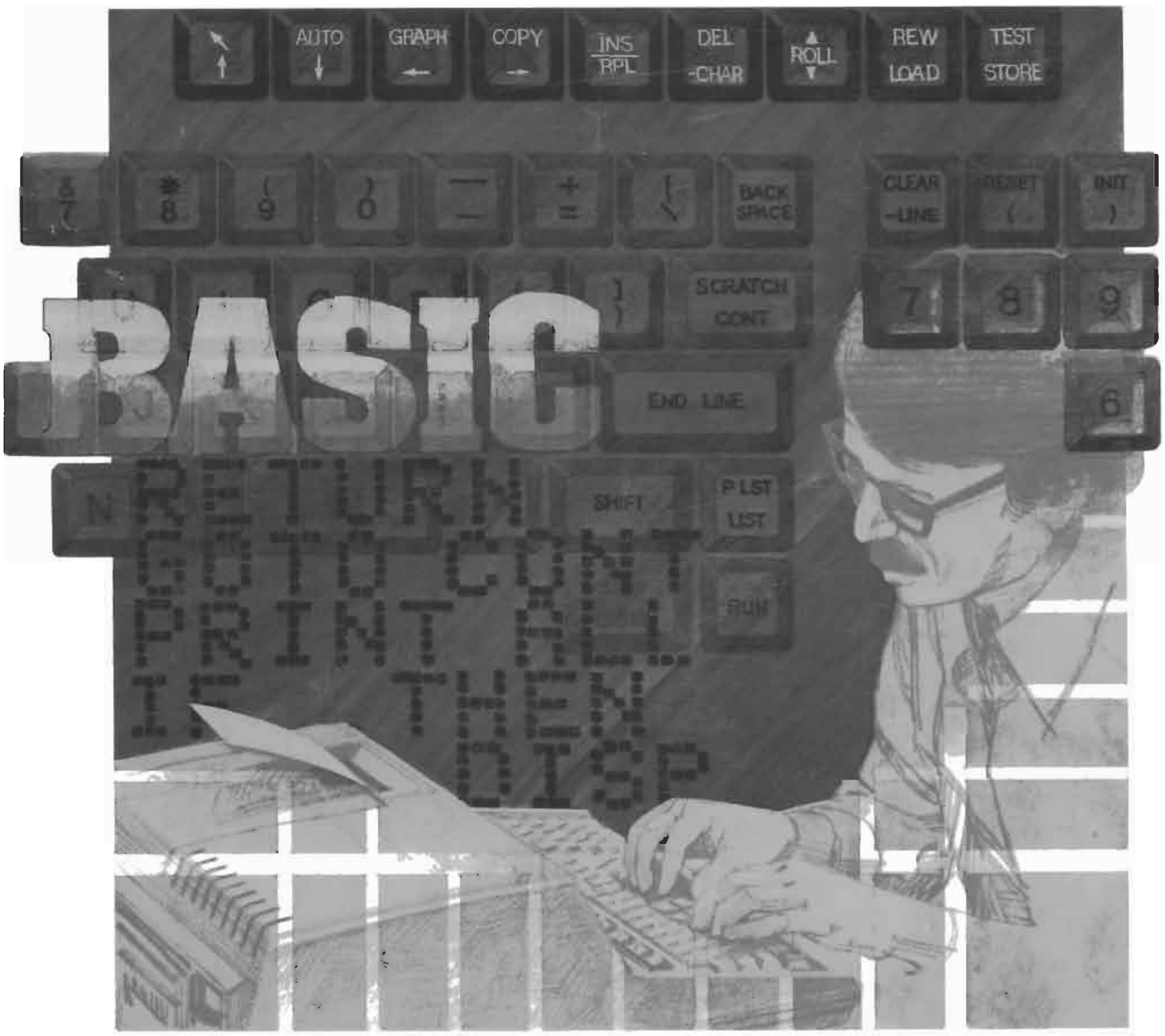
HEWLETT-PACKARD

HP-85

BASIC TRAINING PAC



Hewlett-Packard Intercontinental
3495 Deer Creek Road
Palo Alto, CA 94304
(415) 856-1501

HPSA
7, 4u3 du Bois-du-Lan
P.O. Box
CH-1217 Meyrin 2
Geneva, Switzerland
(022) 82 70 00

Hewlett-Packard
1000 N.E. Circle Blvd.
Corvallis, OR 97330
(503) 757-2000

NOTICE

HEWLETT
PACKARD

HP-85

BASIC Training Pac

December 1979

00085-90052

Contents

Part I: Getting Down to BASIC

Is This Course Written for You?

The answer to this question is YES if:

1. You want to learn how to program your HP-85, AND
2. You have some knowledge of high school algebra, AND
3. One of the statements below fits you:
   - You are new to programming and feel that computers are cold, unforgiving, and confusing machines.
   - You are new to programming and want an easy-paced, friendly, forgiving, do-it-yourself guide to the functions of the HP-85 and the tools of BASIC programming.
   - You have already been introduced to BASIC, but do not consider yourself a programmer. Also, you are unfamiliar with the HP-85. You would like to learn how to use the HP-85, and then find out quickly what you know and don't know about BASIC. You wish to spend time *only* on what you *don't* know.
   - You want a self-teaching course written specifically for the HP-85 which is slower paced and more thorough than the BASIC Programming Guide in the HP-85 Owner's Manual.
   - You want a course you can efficiently study an hour or so at a time. When you resume your study after a few days, you want to pick up where you left off without spending a lot of time figuring out where you are.

To summarize, if you want to learn HP-85 programming, if you have some knowledge of high school algebra, and if one of the statements above fits you, this is your course.

How to Take This Course

Getting Down to BASIC is a friendly, supportive beginner's course in BASIC programming using your HP-85 computer. It is designed to:

1. Let you proceed at your own pace, anywhere from an hour every few days to six to eight hours every day.
2. Let you start anywhere in the course depending on your previous experience.

Proceed at Your Own Pace

Each of the early chapters covers only a small amount of new material, so that each chapter can be completed in an hour or less. Later chapters, those containing more extensive programming exercises, will necessarily take longer, simply because it takes longer to write, test and correct longer programs. However, for the entire course, it is practical to study no more than an hour every few days.

Each chapter is separated from the next by a summary and a review test. The answer to every question is available, so you can check your work. After finishing each chapter, read the summary and take the test. These summaries and tests will help fix in your mind the new ideas presented in each chapter. They will also point out possible areas requiring more study. If you leave the course for a few days, review the last chapter's summary and test to insure a good foundation for the ideas presented in the next chapter.

If you finish several chapters in one sitting, you should, of course, take each review test as it comes along. These reviews will help fix in your mind the new ideas presented in each chapter. They will also point out possible areas requiring a restudy of some of the pages in the previous chapters.

Later, when you're writing your longer programs, you will know how to record on your tape your unfinished program at the end of one session. At the beginning of the next session, you will get your program back from the tape and continue programming where you left off. You will also know at that time how to write messages to yourself within your own program to remind you what each part of your program does. Also, by that time, you will know how to draw a diagram (flowchart) of your program, which is a sort of road map showing where your program goes. All these—a recording of your unfinished program, notes to yourself within your program, and your "road map"—will help you start a new session several days later with little lost time.

Start Anywhere in the Course

What if you already have some BASIC or HP-85 experience and wish to skip those chapters which repeat what you already know? The way to proceed is to take each review test, starting with the first. If you know the answers to a review test, skip that chapter. If the answers are not obvious, you've found a chapter you should study.

Chapter 1

Correct Those Sentences!

Preview

In Chapter 1, you will:

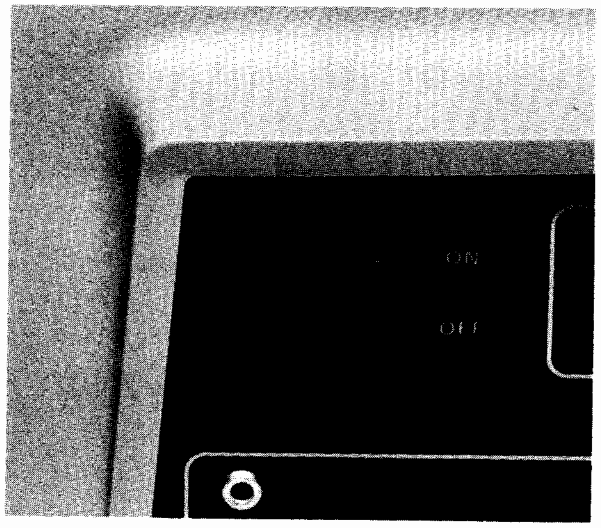
- Type words onto the HP-85's screen.
- Easily correct typing errors.
- Shake hands with a friend: One of the HP-85's error messages.
- Copy your screen onto the printer.

RELAX

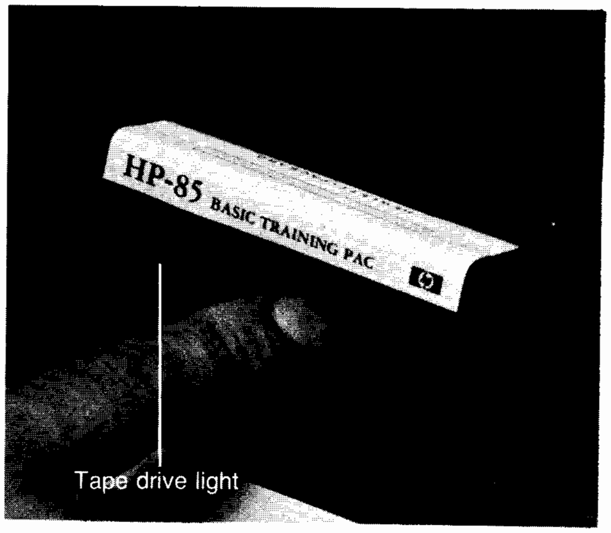
YOU CANNOT HURT YOUR HP-85 BY ANY KEYBOARD OPERATION

As you complete each of the following paragraphs, check it off so you can easily find your place.

(✓) 1. If a tape cartridge is inserted into the HP-85, remove it as shown in Figure 1.



ON-OFF switch



How to remove tape cartridge

FIGURE 1

(✓) 2. Switch the HP-85 ON (see Figure 1). If already ON, switch OFF, then ON. When the HP-85 is switched OFF, a bright spot will appear at the center of your screen. This is normal and harmless. When the HP-85 is either switched ON or OFF, the yellow tape drive light will turn on briefly. This is also normal and harmless.

( ) 3. After a few seconds warmup, a small dash should be visible in the upper left corner of the HP-85's screen, as shown in Figure 2.

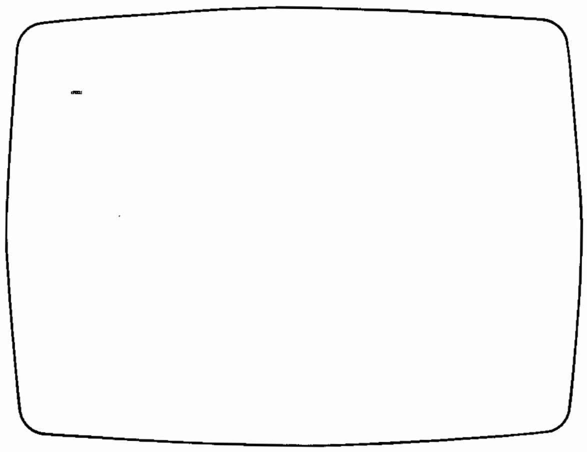


FIGURE 2
Cursor location when the HP-85 is first turned on

This dash is called a **cursor.** The cursor shows where the next character you type will go. You'll be moving it around soon.

( ) 4. The HP-85 has two modes of operation, calculator and program.

CALCULATOR MODE: Used to write and edit text and to perform calculations. Also used to write BASIC programs.

PROGRAM MODE: Used to run BASIC programs.

In this chapter, you'll be in calculator mode exclusively. I'll tell you about program mode later.

REMEMBER
YOU ARE IN CHARGE
THE HP-85'S DESIRE IS TO OBEY

If the HP-85 gets confused, you can always switch it OFF, then back ON.
And you can always start this course again at this page at any time.

( ) 5. The HP-85 keyboard is like a typewriter keyboard with some important differences. I'll tell you about some of these differences now, and later you'll see these differences in action when you exercise the HP-85's keys yourself. See Figure 3 for the location of the keys mentioned in the next several paragraphs.

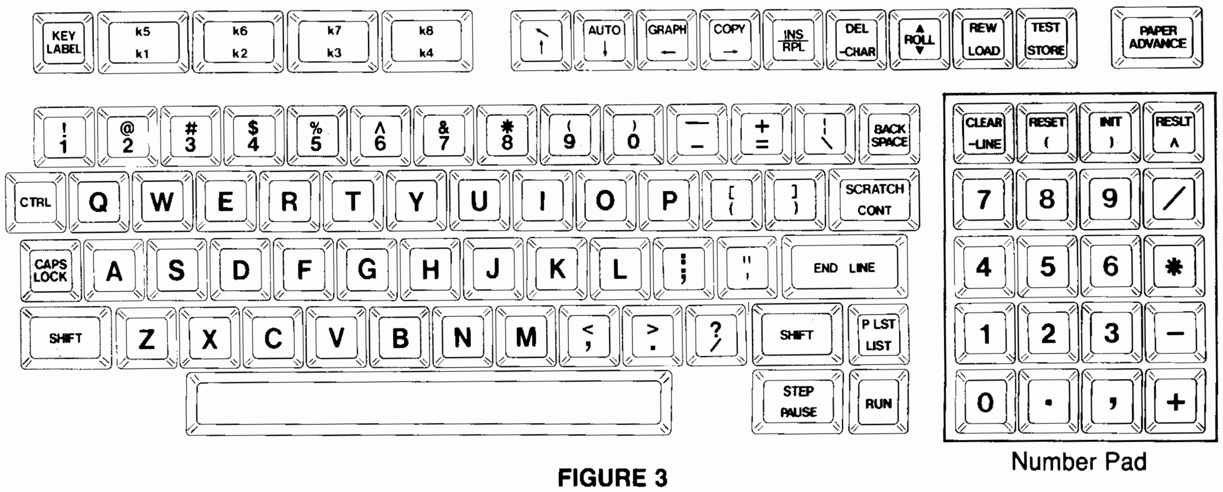


FIGURE 3
Locations of some keys

(√) 6. If you want to produce one B, for instance, press and release the key just like a typewriter. If you want to produce a string of characters, hold the key down.

( ) 7. The letter keys normally produce capital letters. To get small letters, you normally hold down the (SHIFT) key, then press the letter key—just the opposite from an ordinary typewriter.

Capital letters are generally easier to read on a screen, and most people prefer to write their programs in capital letters. So we made capitals standard.

( ) 8. When a key has two symbols on it, like (% 5) (above the (R) key), pressing the key by itself gives you a 5. If you hold down (SHIFT) while pressing (% 5), you get %.

Only the 26 letter keys normally act the opposite way from a typewriter.

( ) 9. The (CAPS LOCK) key affects only the 26 letter keys.

( ) 10. Notice the group of keys at the right edge of the keyboard, which include the 10 numerals 0 through 9. This is called the **number pad**. All of these 10 numerals plus the other 9 key symbols (() ()) (∧) (/) (✱) (−) (+) (,) (·) also appear on the typewriter portion of the keyboard. They're grouped here for convenience only.

(✓) 11. The curved parentheses appear 3 places; 1) On the number pad—top row; 2) Just right of (P); and 3) Above the letter (O). Each pair of parentheses serves the same purpose. Parentheses are used often in programming and calculations, so we have tried to accommodate the touch typist, the two finger typist, and the user who does a lot of calculating.

( ) 12. Before going further, make sure the (CAPS LOCK) key is released. It should be level with the other keys. Press to release.

( ) 13. Before you type a message onto the HP-85's screen, how about a little practice with the keys.

( ) 14. To type the capital letter ''B'' onto the screen, press and release the (B) key just like you were using a typewriter. One B should appear on the HP-85's screen, and the cursor should now be moved over one position.§ If you started with a clear screen, see Figure 4 for a picture of what your screen should now show.

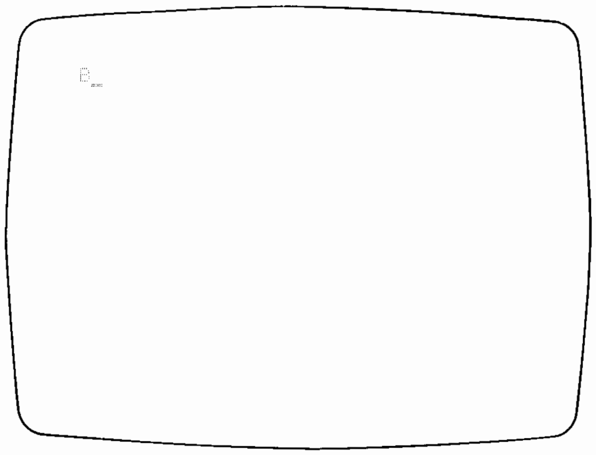


FIGURE 4
Screen after Step 14

( ) 15. I'll show you now how to use the CLEAR command to clear your screen. Press and hold down one of the two (SHIFT) keys, Figure 3. While holding (SHIFT) down, press and release the (CLEAR -LINE) key (upper left corner of number pad, Figure 3), then release (SHIFT). Your screen is now clear, except for the HP-85's faithful cursor.

§If the screen intensity needs adjustment, see page 36 under The Display in your Owner's Manual.

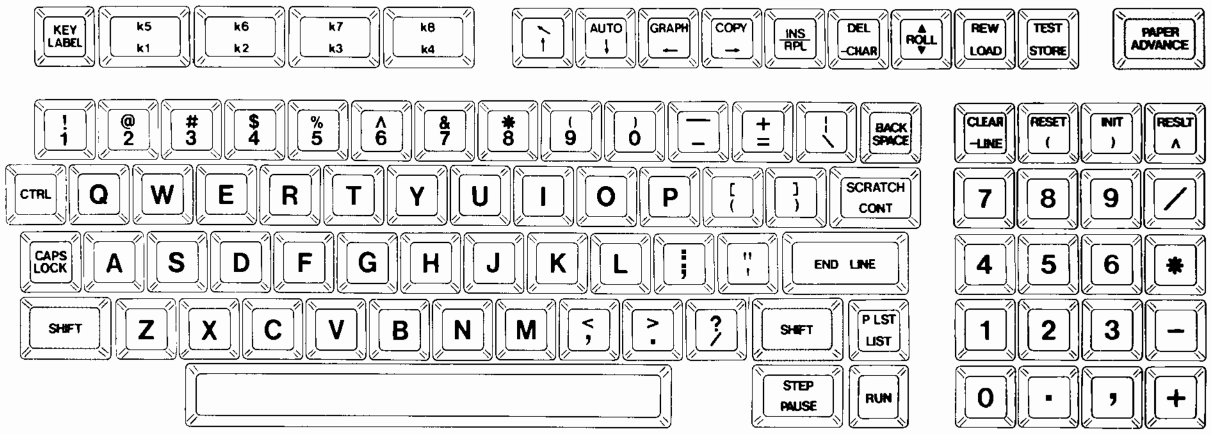
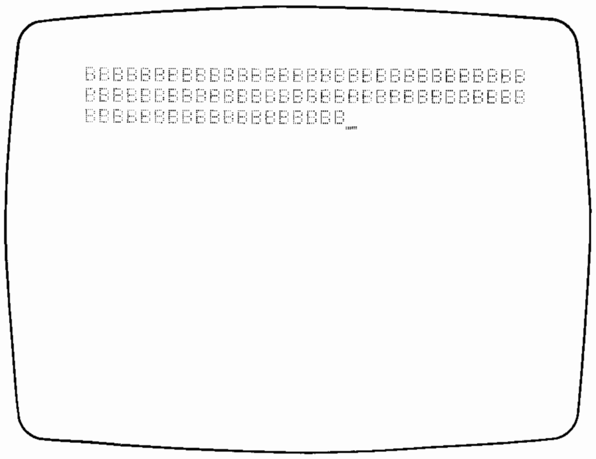


FIGURE 5

Locations of some keys

(√) 16. Notice how the top word, CLEAR, on the CLEAR -LINE key cap is the name of the action this key performs when used with SHIFT. The same idea is used with all of the HP-85's keys. When a key cap has two names or symbols on it, SHIFT is used to get the upper symbol or action. Four exceptions are CAPS LOCK, BACK SPACE, PAPER ADV and KEY LABEL. Each names only one function. I'll tell you about these functions later in this book.

( ) 17. Most of the HP-85's keys show a repeating action when held down. To get a feel for this, press B again, and this time hold the key down for five seconds or so to get a lot of "B's" on the HP-85's screen. If you hold B down for just six seconds, your screen will look something like Figure 6.



```
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBB_
```

FIGURE 6

A lot of "B's"

( ) 18. The steps you will perform next will make most sense if your bottom row of "B's" fills at least half of that line. Again, Figure 6 shows what I mean. If your present bottom line does not satisfy you, simply press and hold down the B key until you create a new bottom line that is OK.

✓
( ) 19. It's time to meet another eraser, $\boxed{\text{BACK SPACE}}$. This key erases characters as it moves the cursor backwards. Press $\boxed{\text{BACK SPACE}}$, Figure 7, three times to remove three "B's." If your screen looked like Figure 6 before, it should look like Figure 8 now.

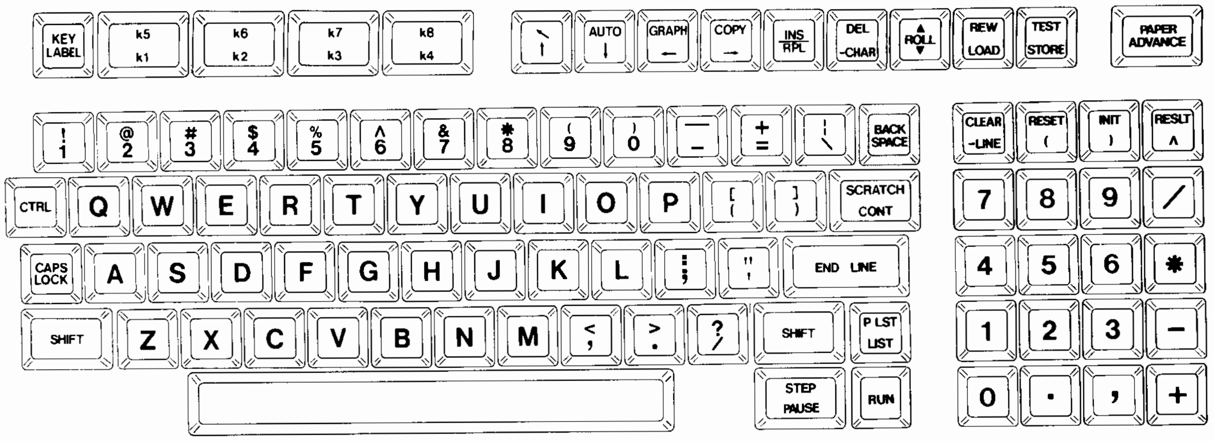
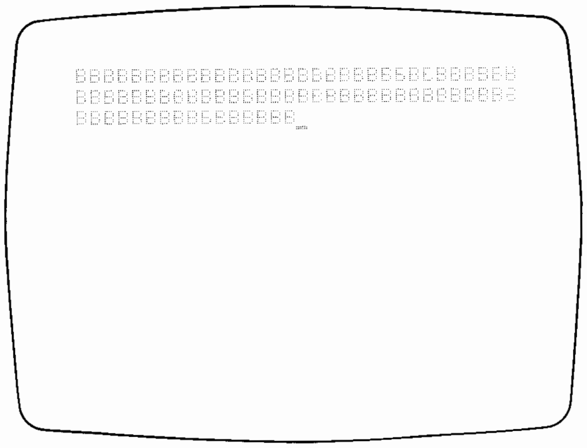


FIGURE 7

Location of $\boxed{\text{BACK SPACE}}$



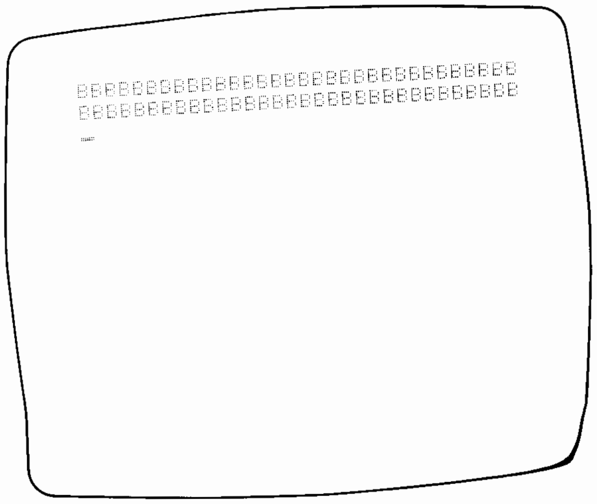
```
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB_
```

FIGURE 8

Three fewer "B's"

( ) 20. Let me show you another $\boxed{\text{BACK SPACE}}$ power. The shifted function of $\boxed{\text{BACK SPACE}}$ moves the cursor rapidly to the left edge of the screen, wiping out characters as it goes.

(✓) 21. Press SHIFT + BACK SPACE and see the bottom row of "B's" wiped clean. Your screen should look like Figure 9.



```
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
_
```

FIGURE 9

One less line of "B's"

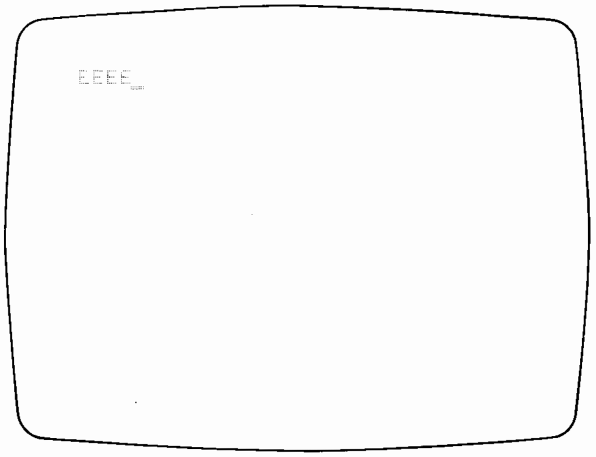
( ) 22. Now you know how to erase single characters with BACK SPACE, how to erase rapidly back to the beginning of the line using SHIFT + BACK SPACE, and how to clear your screen using SHIFT + CLEAR -LINE. You'll now exercise BACK SPACE and CLEAR a little, and then type a message onto the HP-85's screen.

( ) 23. Clear your screen (press SHIFT + CLEAR -LINE).

( ) 24. Type four capital (unshifted) letter "E's," so your screen looks like Figure 10.

HELP MESSAGE:

If you have trouble, use BACK SPACE and SHIFT + BACK SPACE to erase your mistakes. If you need more powerful medicine, use SHIFT + CLEAR -LINE to clear your screen, and start again at step 24. If all else fails, turn the HP-85 OFF, then ON, and start at step 24.



```
EEEE_
```

FIGURE 10

Screen "E's"

(✓) 25. Using BACK SPACE and the A, S and Y keys, change your four "E's" into EASY. Figure 11 shows the result you should get. Again, use the trouble-killers mentioned in step 24 as needed.

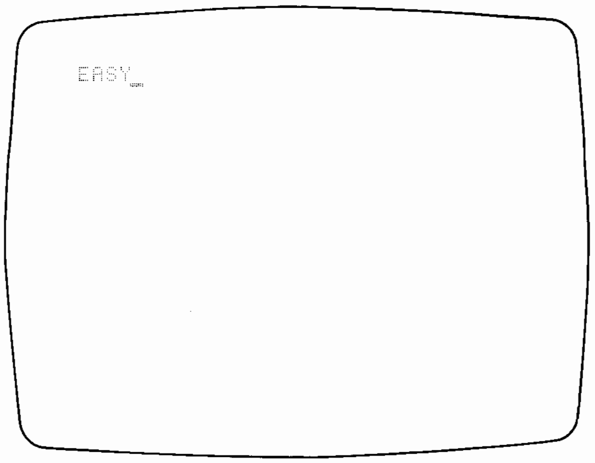


FIGURE 11

Screen "EASY"

( ) 26. OK. Now your HP-85 muscles are beginning to develop. Let's move ahead. You're going to gain more control over your cursor.

( ) 27. You now have some strong medicine in your medicine cabinet: BACK SPACE to erase characters, and CLEAR to clear your screen. But they both remove characters. Very often, you will wish to move your cursor without killing characters. You have that power at your fingertips. Figure 12 shows the location of four cursor moving keys that control five cursor movers: ↖ ↑ ↓ ← →. To use ↖, you press SHIFT + ↑. The other four cursor movers are unshifted functions. What do these keys do? Read on.

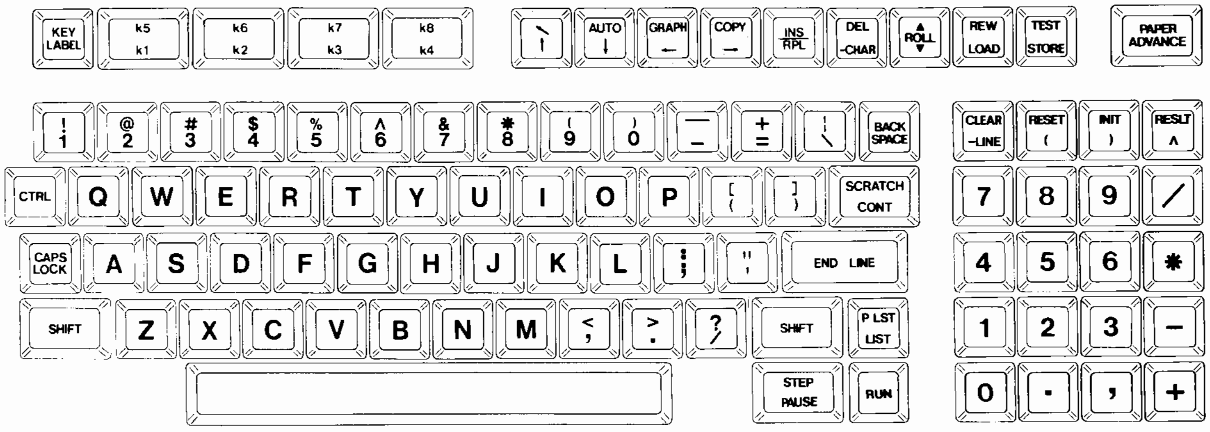


FIGURE 12

Locations of some keys

(✓) 28. CURSOR MOVING KEYS—What they do:

a. They all move the cursor *without erasing characters*.

b. ↖: Moves the cursor "home", that is, to the upper left corner of the screen.

c. ↑: Moves the cursor up.

d. ↓: Moves the cursor down.

e. ←: Moves the cursor left.

f. →: Moves the cursor—you guessed it—right.

( ) 29. The four no-shift arrows ↑↓←→ move the cursor either one position or many positions, depending on how long the key is held down.

( ) 30. The home key, ↖, has no repeating action. Once the cursor is home, it stays, regardless how long ↖ is held down; that is, how long SHIFT + ↑ are held down.

( ) 31. Get your fingers ready. You're going to take your cursor on a short round trip. It's easy to get these cursor moving keys mixed up, so don't be concerned if your cursor moves off in an unexpected direction. Just keep pressing those cursor movers, and you'll succeed. The cursor responds well to the trial and error method of movement.

( ) 32. If you feel things are getting out of control—maybe you've hit some key that causes strange things to happen—bring in the trouble-killers. Press SHIFT + CLEAR -LINE to clear your screen, or switch the HP-85 OFF, the ON. Type EASY again, and start at step 33.

( ) 33. Now for your cursor journey. Press → to move your cursor one space to the right.

( ) 34. Next, press ↓ to move your cursor down one line.

( ) 35. Now press ← to move your cursor one space to the left. It now is positioned one line below Y and one position right of Y.

( ) 36. To complete this short trip, press ↑ to put the cursor back where it started, on the top line just right of Y.

( ) 37. Your cursor can run as well as walk. When the ↑, ↓, ← and → keys are held down, the cursor moves rapidly in the direction shown by the arrow. What happens when the cursor reaches a screen boundary? Hold each of these four arrow keys down for a good 10 seconds each and find out.

( ) 38. Did you hold the ← and → keys down long enough to move the cursor to the upper right and lower left corners of the screen? When the ← key is held down, the cursor moves left to the screen boundry, then reappears on the right one line up, moves left on that line to the screen's left edge, and so on until it reaches the upper left corner. Its next appearance is at the lower right corner where its upward journey begins again. When → is held down, the cursor follows a similar path in the opposite direction.

√
( ) 39. Hold down [←] long enough to see the cursor visit every position on the screen. It should take about 25 seconds to take a complete round trip.

( ) 40. If you wish, hold [→] down to see the reverse round trip.

( ) 41. Now use the home key (press [SHIFT] + [↖]) to put the cursor in the upper left corner under the E of EASY.

( ) 42. There is one simple keystroke error that is harmless, and yet this error has been known to send strong men into shock. Take a look at the [←] key. Note that the whole key cap says [GRAPH ←]. Pressing [GRAPH] ([SHIFT] + [GRAPH ←]) puts the HP-85 into graphics mode, one of the HP-85's most powerful abilities. While this course is too short to include graphics (your Owner's Manual has an excellent graphics section), you should know one thing about this key. When pressed, the screen goes blank, yet *everything that was on your screen is preserved.* It's as though the HP-85 had two pages in its hands, an alpha page—the one you're using—and a graphics page. When [GRAPH] is pressed, the HP-85 holds the blank graphics page up to the screen, inviting you to draw. To get the alpha page back, *simply press any arrow key.*

When you're busy editing, you might, by mistake, press [SHIFT] + [GRAPH ←] intending, for example, to press [SHIFT] + [↖]. Unless you're forewarned, the sight of everything disappearing from your screen might make you nervous.

( ) 43. Press [SHIFT] + [GRAPH ←]. See the screen go dark.

( ) 44. Now press [←] and get your old familiar alpha screen back unharmed.

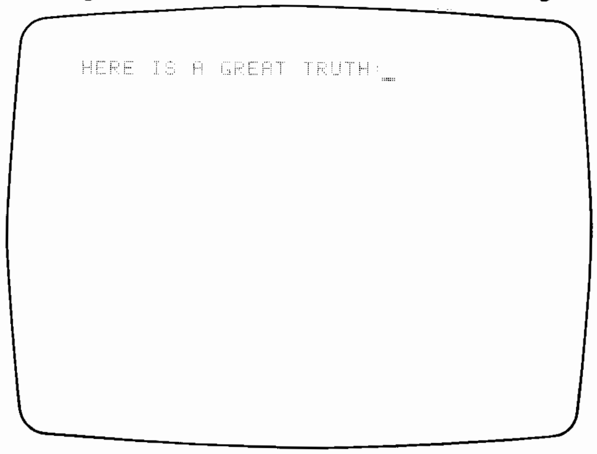
( ) 45. Now press [→] (or [SHIFT] + [↖]) to get the cursor back home again, under the E of EASY.

( ) 46. In the next step, I'm going to ask you to type a new message. Use your [BACK SPACE] eraser as needed, and if things go bad for you, clear your screen by pressing [SHIFT] + [CLEAR -LINE] and start again at step 47.

( ) 47. Type this message right over EASY:

```
HERE IS A GREAT TRUTH:
```

After you type this message, the HP-85's screen should look like Figure 13.



```
HERE IS A GREAT TRUTH:_
```

FIGURE 13

Screen after Step 47

(√) 48. The suspense created by this line will be satisfied soon, but first, let me introduce you to one of the HP-85's friendly error messages.

( ) 49. When the HP-85 does not know what to do next, it tries to explain its confusion by showing you an error message. To see one of these messages, you're going to deliberately confuse the HP-85.

( ) 50. Find the END LINE key. See Figure 14.

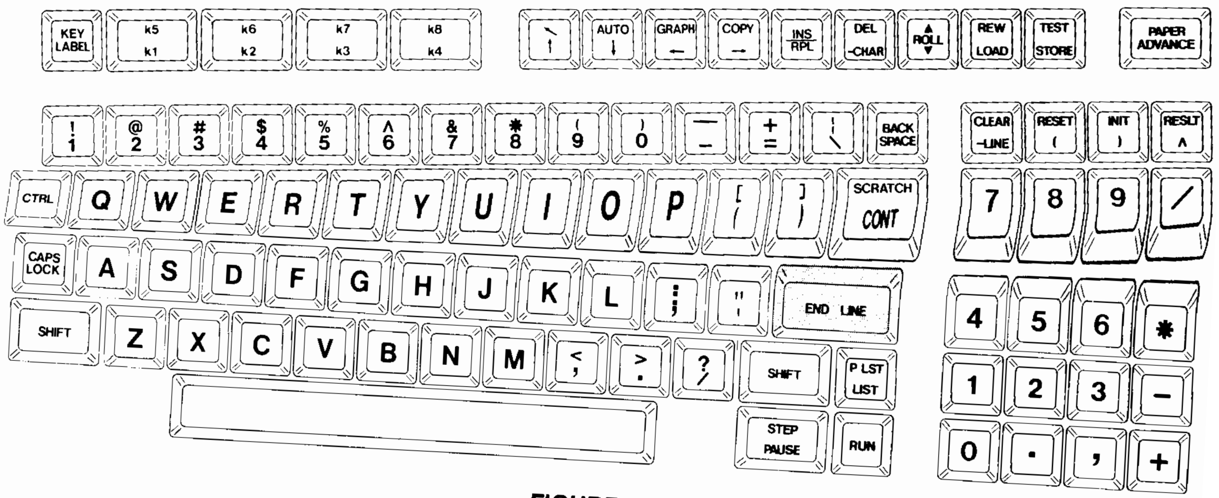


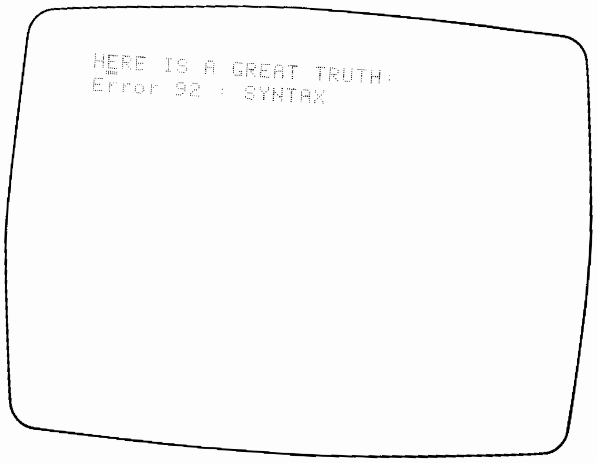
FIGURE 14

Location of END LINE

( ) 51. Now press END LINE. The HP-85 tells you it did not understand that by beeping and by showing you:

```
Error 92 : SYNTAX
```

as shown in Figure 15.



```
HERE IS A GREAT TRUTH:
Error 92 : SYNTAX
```

FIGURE 15

Error 92

√

( ) 52. END LINE is perhaps the most important and most frequently used key on the keyboard. Whenever you wish the HP-85 to act on what you have typed, you press END LINE.

Throughout this course I'll ask you to type a number of instructions into the HP-85 which the HP-85 will understand perfectly when you press END LINE. However, that's for later on. Right now, the important thing to know is that you do *NOT* press END LINE when you're simply using the HP-85 as a typewriter, as you are now.

( ) 53. Notice that the cursor is now under the first E of HERE. The HP-85 is pointing out the first character it did *not* understand. Later in the course you'll understand what the HP-85 thinks that first H is. For now, it's not important.

( ) 54. When the HP-85 gives you an error message, it's telling you:

> "I don't understand what you told me, but please try again. I'm ready to accept more instructions, and I'll do my very best to obey."

The key words here are "try again." The HP-85 has not turned off or gone off into the fourth dimension somewhere. It is ready to respond just as before.

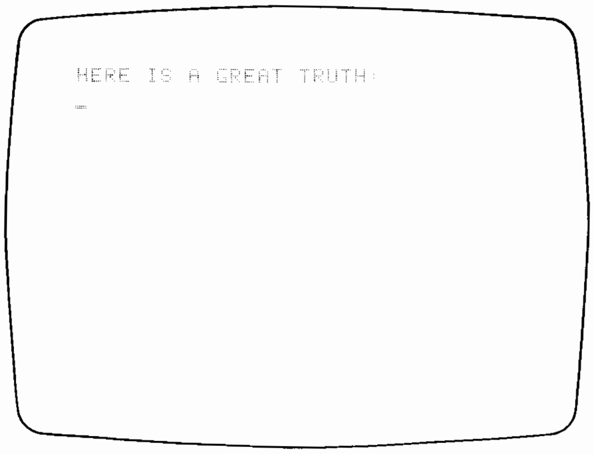
So let's go!

( ) 55. Use ↓ to move the cursor down one line.

( ) 56. Now use ← to move the cursor left one position to the beginning of the second line.

( ) 57. You will soon get rid of that error message and discover a new character clearing power at the same time. The CLEAR -LINE key has two functions, one using SHIFT, the CLEAR command, and the other, unshifted function -LINE. You already know about CLEAR. It clears your screen. The unshifted function, -LINE, clears the line from the cursor position to the right edge of the screen.

( ) 58. If you're in trouble after trying this step, read step 59. Now press -LINE (press the CLEAR -LINE key) and see all characters on the second line above and to the right of the cursor wiped away. Your screen should look like Figure 16.



```
HERE IS A GREAT TRUTH:
_
```

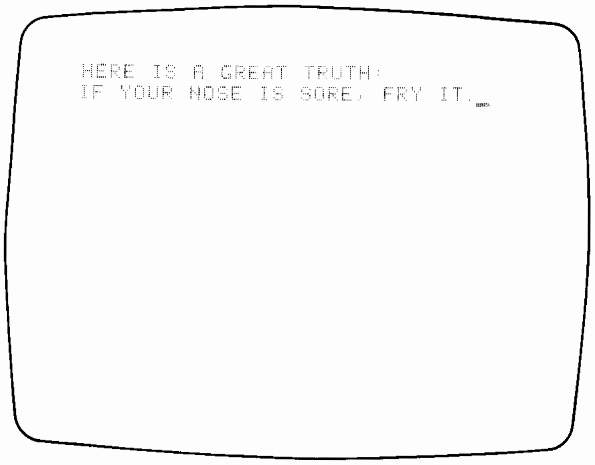
FIGURE 16
Screen after Step 58

(✓) 59. If your screen looks like Figure 16, you get a reward: You may skip this step. If you've gotten into trouble, press CLEAR; that is, press SHIFT + CLEAR -LINE to clear your screen. Then perform steps 46 and 47 to get `HERE IS A GREAT TRUTH:` back on the screen. Next, use ⌴ to move your cursor to the beginning of the second line. Now proceed to the next step.

( ) 60. I've deliberately put some errors into this next text to give you a chance to use the HP-85's unique, simple and powerful editing tools. Use your eraser, BACK SPACE, if necessary to help you type this line onto your screen just as I show it:

```
IF YOUR NOSE IS SORE, FRY IT.
```

( ) 61. Your screen should now look like Figure 17. If the HP-85 acts obstreperous, smooth its feathers by starting at step 59.



```
HERE IS A GREAT TRUTH:
IF YOUR NOSE IS SORE, FRY IT._
```

FIGURE 17
Screen after Step 60

( ) 62. I'll lead you, step by step, through the editing of this sore nose message to turn it into:

```
IF YOU'RE NOT SURE, TRY IT.
```

Now, meet one of your strongest editing tools, INS RPL. See Figure 18 for its location.

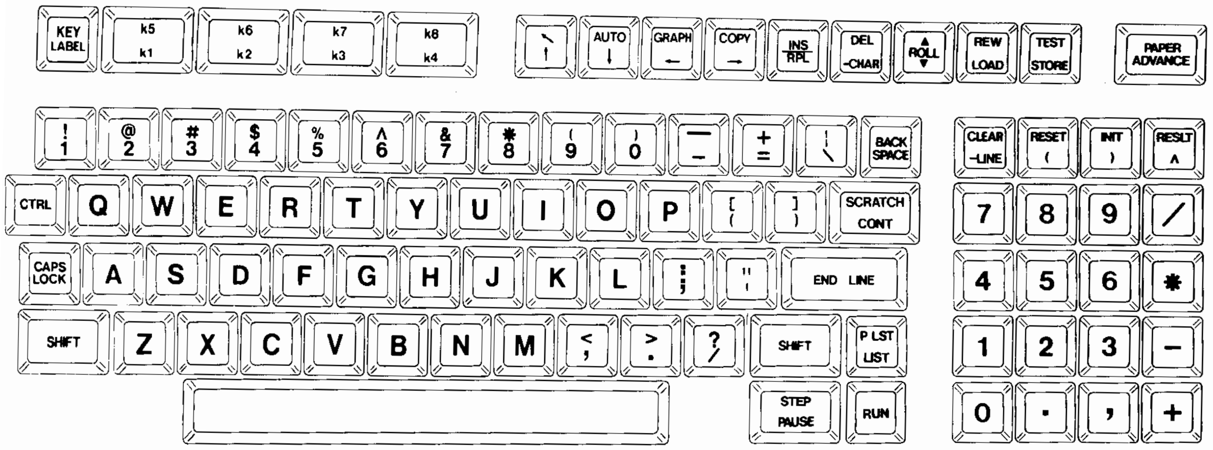


FIGURE 18

Location of (INS RPL)

( ) 63. I'll describe the (INS RPL) key here. Please don't touch any keys until step 65. The INSERT/REPLACE key is a toggle. When the HP-85 is turned on, this key is in the REPLACE mode, as shown by the single cursor. When the single cursor is under a character, say A, typing another character, say B, causes the A to be replaced by the B. You saw this happen when you typed a new top line over EASY in step 47.

Pressing the (INS RPL) key puts the HP-85 into INSERT mode, as shown by the double cursor, . Say the two cursors are under the characters AC. If (B) is pressed, B would be inserted between A and C, giving ABC. Pressing (INS RPL) again would put the HP-85 back into REPLACE mode, and you would see ABC.

( ) 64. HELP MESSAGE:

> If things turn sour during your editing of IF YOUR NOSE IS SORE, FRY IT., and you have trouble getting things organized, follow this plan:
>
> a. Clear your screen by pressing (CLEAR); in other words, by pressing (SHIFT) + (CLEAR -LINE).
>
> b. Perfrom steps 46 and 47 to get HERE IS GREAT TRUTH: back on the top line.
>
> c. Using (←), move the cursor to the beginning of the second line.
>
> d. Perform step 60 to put IF YOUR NOSE IS SORE, FRY IT. on the second line.
>
> e. Proceed with step 65.

( ) 65. Now let's take care of that sore nose. Using (←), move your cursor to the R of YOUR and see IF YOUR NOSE ...

( ) 66. Now press (INS RPL) and see ... YOUR ...

( ) 67. Press the apostrophe, ('), unshifted, just left of (END LINE). Now you have YOU'R.

(√) 68. Using →, move the double cursor one position right, and see YOU'R_NOSE.

( ) 69. Press E and see YOU'RE_NOSE.

( ) 70. Now press INS/RPL to put the HP-85 back into its normal REPLACE mode, and see YOU'RE_NOSE.

( ) 71. Using →, move your cursor three positions right, and see YOU'RE NOSE.

( ) 72. Here's another editor key: -CHAR, whose location is shown in Figure 19. -CHAR is the unshifted function of the DEL -CHAR key. -CHAR, when pressed, deletes the character above the cursor, and all characters to the right of the cursor move left to fill in the gap.

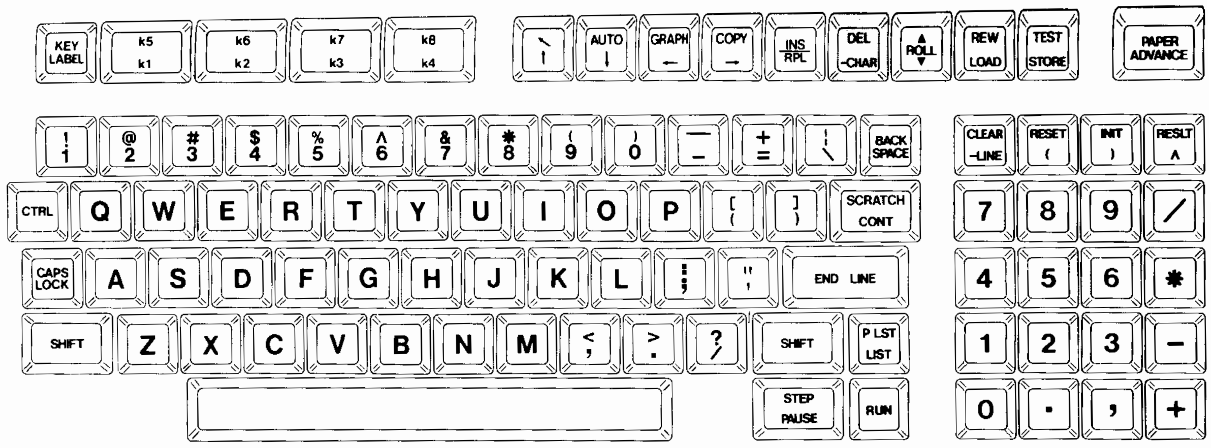


FIGURE 19

Location of DEL -CHAR

( ) 73. If you wish to erase a character without the characters on the right moving left, use BACK SPACE or the space bar.

( ) 74. Press -CHAR, and see YOU'RE NOE IS.

( ) 75. Now press T to complete one more word. See YOU'RE NOT_IS.

( ) 76. Now press -CHAR three times and see YOU'RE NOT_SORE,. You're getting close.

( ) 77. Using →, move your cursor two positions right, and see NOT SORE,.

( ) 78. Press U, and see SURE,.

( ) 79. Using →, move four positions right and see SURE, FRY.

( ) 80. Press T, and see the correct line IF YOU'RE NOT SURE, TRY IT.

(✓) 81. HELP MESSAGE:

> If you have trouble you can't cure between here and the end of the chapter, follow this plan, using your BACK SPACE eraser when needed:
>
> a. Clear your screen by pressing CLEAR; that is, by pressing SHIFT + CLEAR -LINE.
>
> b. Perform steps 46 and 47 to get `HERE IS A GREAT TRUTH:` back on the top line.
>
> c. Using →, move your cursor to the beginning of line 2.
>
> d. Type: `IF YOU'RE NOT SURE, TRY IT.`
>
> e. Proceed with step 82.

( ) 82. Now use → to move the cursor to the beginning of the third line.

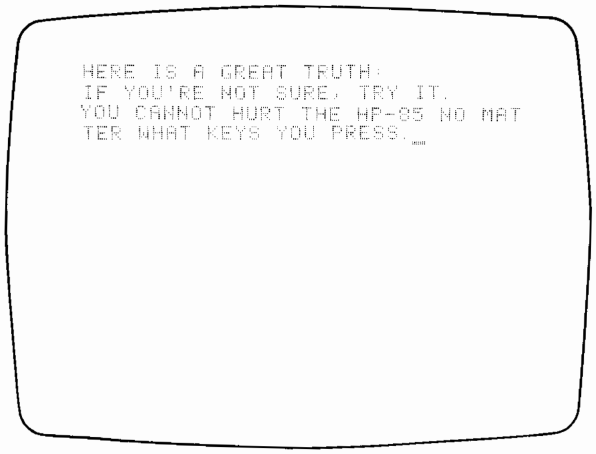
( ) 83. I'm going to lead you into trouble again, and soon after I'll show you how to edit the trouble away. You will then see the final message of Chapter 1 on the screen and on the printer.

( ) 84. In this step, keep your fingers off the HP-85's keys. Just read. One line on the screen and on the printer can contain a maximum of 32 characters, including spaces. In step 85 I'm going to ask you to type too many characters for one line. Have faith. Don't worry when you see, in step 85, one word splitting between the third and fourth line. I'll soon show how to correct the problem.

( ) 85. Try to type the entire sentence below on the third line. It won't fit, but keep trying. The HP-85 will automatically move to the fourth line after 32 characters are typed, including spaces. However, the word `MATTER` will be split in the middle between lines three and four. Here is the sentence for you to type:

`YOU CANNOT HURT THE HP-85 NO MATTER WHAT KEYS YOU PRESS.`

Your screen should look like Figure 20.



```
HERE IS A GREAT TRUTH:
IF YOU'RE NOT SURE, TRY IT.
YOU CANNOT HURT THE HP-85 NO MAT
TER WHAT KEYS YOU PRESS._
```

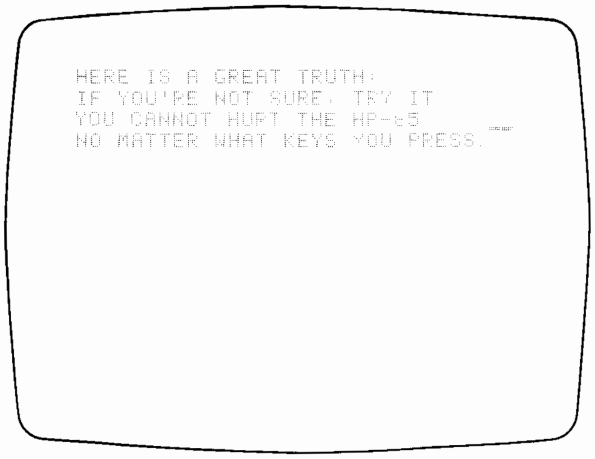
FIGURE 20

Screen after Step 85

(✓) 86. Now, using $\boxed{\uparrow}$ and $\boxed{\rightarrow}$, move your cursor to the space between `THE HP-85` and `NO`. You should see ... `THE HP-85_NO` ...

( ) 87. Press $\boxed{\frac{\text{INS}}{\text{RPL}}}$ to put the HP-85 into INSERT mode. You should see `THE HP-85_NO`.

( ) 88. Press the space bar six times and see your final message properly displayed. Figure 21 shows this final message as it should appear on your screen.



```
HERE IS A GREAT TRUTH:
IF YOU'RE NOT SURE, TRY IT
YOU CANNOT HURT THE HP-85
NO MATTER WHAT KEYS YOU PRESS.
```

FIGURE 21
Screen after Step 88

( ) 89. This message is more than a typing exercise. The HP-85 cannot be hurt by any keyboard operation. However, programs can be hurt, so do your experimenting with NO cartridge inserted and with NO program in memory. If the HP-85 is free of cartridge and program, as it is now, you can learn a great deal, risk free, by trying things. If you're not sure what some key does in a certain situation, try it. Remember, try as you might, you will be unable to create a mess that cannot be cleared up by clearing the HP-85's screen or by turning the HP-85 OFF, then ON.

( ) 90. If the printer has no paper, insert a roll as described in your Owner's Manual on page 277.

( ) 91. The `COPY` command is the shifted function of $\boxed{\overset{\text{COPY}}{\rightarrow}}$. Press $\boxed{\text{SHIFT}}$ + $\boxed{\overset{\text{COPY}}{\rightarrow}}$ and print a copy of your good work.§ Enshrine it in the place of honor reserved for it below.

PLACE OF HONOR

§ If the printing intensity needs adjustment, see page 35 under Printer Control in your Owner's Manual.

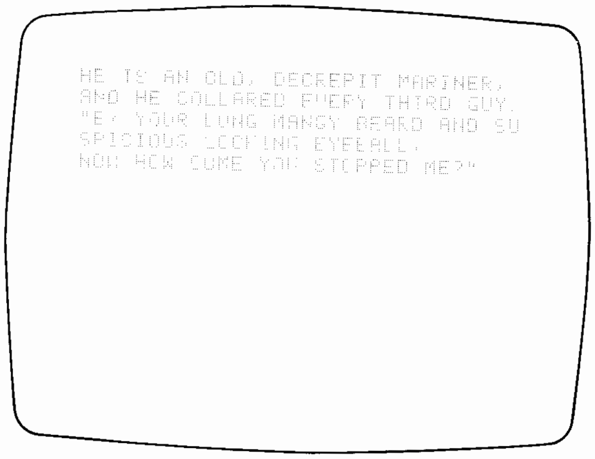
Summary of Chapter 1

- **Cursor:** A dash on the screen displayed by the HP-85. It shows where the next typed character will go. Reference: Page 1-7, step 3.
- **Calculator mode:** For using the HP-85 as a typewriter, for doing calculations, and for writing BASIC programs. Reference: Page 1-7, step 4.
- **Program mode:** For running BASIC programs. Reference: Page 1-7, step 4.
- **Capital letters are standard.** Use SHIFT to get small letters. Reference: Page 1-8, step 7.
- **Use SHIFT to get the symbol or function shown at the top of a key cap.** Reference: Page 1-8, step 8.
- CAPS LOCK **works only for the 26 letter keys.** Reference: Page 1-8, step 9.
- SHIFT + CLEAR -LINE **clears the HP-85's screen.** Reference: Page 1-9, step 15.
- CLEAR **means** SHIFT + CLEAR -LINE. Reference: Page 1-10, step 16.
- BACK SPACE **erases characters.** As it moves the cursor backwards, one position at a time, it clears the space the cursor moves to. Reference: Page 1-11, step 19.
- SHIFT + BACK SPACE **erases characters rapidly** as it moves the cursor to the beginning of the line. Reference: Page 1-11, step 20.
- ↑, ↓, ← and → **move the cursor without erasing characters.** Pressing each key moves the cursor in the arrow direction one position at a time or rapidly, if the key is held down. Reference: Page 1-14, steps 28, 29.
- ↖ (a shifted function) moves the cursor "home," which is the upper left corner of the screen, without erasing characters. Reference: Page 1-14, steps 28 and 30.
- **If your screen suddenly blanks,** perhaps you pressed SHIFT + GRAPH. To get your normal (alpha) screen back, press any arrow key. Reference: Page 1-15, step 42.
- END LINE **should not be pressed when using the HP-85 as a typewriter.** Reference: Page 1-17, step 52.
- An **error message** is the HP-85's way of saying "I don't understand you." The keyboard is left alive, and you may proceed with freedom. Reference: Page 1-17, step 54.
- -LINE **clears the line from the cursor position to the right edge of the screen.** Reference: Page 1-17, step 57.
- INS RPL **is a toggle that switches between INSERT and REPLACE modes.** Reference: Page 1-19, step 63.

- **Insert mode:** Two cursors appear, and the next character typed is inserted between the two cursors. Reference: Page 1-19, step 63.
- **Replace mode:** One cursor appears, and the next character typed replaces the character above the cursor. The HP-85 is in replace mode when first switched on. Reference: Page 1-19, step 63.
- -CHAR **deletes the character above the cursor.** The characters to the right of the cursor move left to fill in the gap. Reference: Page 1-20, step 72.
- **The line length on screen and printer is 32 characters,** including spaces. Reference: Page 1-21, step 84.
- COPY **prints the contents of the screen** onto the printer. Reference: Page 1-22, step 91.
- COPY = SHIFT + COPY. Reference: Page 1-22, step 91.

Review Test for Chapter 1

Samuel Taylor Coleridge did not create the final version of "The Rime of the Ancient Mariner" on his first try. Through extensive research, I have uncovered a little known early draft of this famous work which I share with you in Figure 22.

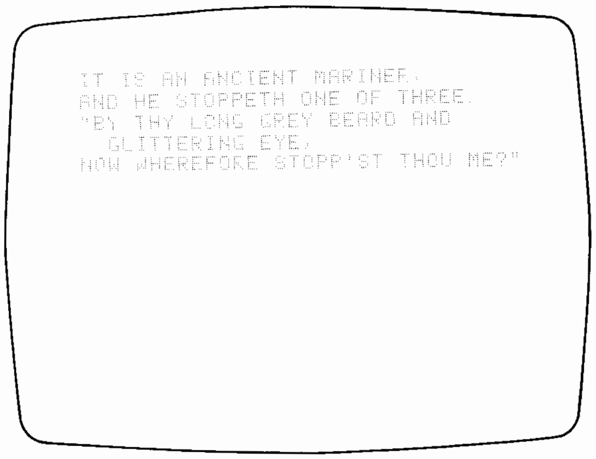


```
HE IS AN OLD, DECREPIT MARINER,
AND HE COLLARED EVERY THIRD GUY.
"BY YOUR LONG MANGY BEARD AND SU
SPICIOUS LOOKING EYEBALL,
NOW HOW COME YOU STOPPED ME?"
```

FIGURE 22
Early draft

Your review test is the following: Type onto the HP-85's screen this early draft of the first four lines, then press SHIFT + COPY so you can compare your effort more easily with Figure 22, to check your typing.

Then edit this early draft to produce the final version, Figure 23. No fair clearing your screen and typing your final version from scratch. Use your editing tools to change the early draft into the finished product. When you complete your editing, copy your efforts to allow easy comparison with Figure 23 and to show to your friends and relatives.



```
IT IS AN ANCIENT MARINER,
AND HE STOPPETH ONE OF THREE.
"BY THY LONG GREY BEARD AND
  GLITTERING EYE,
NOW WHEREFORE STOPP'ST THOU ME?"
```

FIGURE 23
Final version

Chapter 2

2 + 2 *Does* Equal 4

Preview

In Chapter 2, you will:

- Meet more friendly error messages.
- Learn how to print automatically what is displayed.
- Meet [END LINE], the HP-85's "enter" key.
- Learn how to solve math problems.
- See how the HP-85 can save you a lot of typing by allowing you to recycle instructions typed earlier onto the screen.

(√) 1. HELP MESSAGE:

> IF YOU NEED HELP
> during this chapter, follow these steps:
>
> a. First, check for typing errors. If you find any, try your editing tools, like the cursor movers, [↖], [↑], [↓], [←], [→], plus [BACK SPACE], [INS RPL], [-CHAR], and [-LINE]. See the summary of Chapter 1, page 1-23, for reminders of their functions.
>
> b. If the text editors don't work, clear your screen (press [SHIFT] + [CLEAR -LINE]). Then go back and start at the last step that gives you a convenient starting place, like the beginning of a new calculation example.
>
> c. If that doesn't work, start the chapter over again.

( ) 2.If the HP-85 is OFF, switch it ON. See Figure 1, page 6 for switch location. If it's already ON, make sure the HP-85 has a clear head and screen by turning it OFF, then ON.

( ) 3. If it's been several days since you completed Chapter 1, you might take the review test over again to sharpen your text editing tools. The review test starts on page 1-24. When you finish your test, clear your screen ([SHIFT] + [CLEAR -LINE]) and continue with step 4.

(√) 4. PRINTALL§ **command:**

You've already used two other commands: CLEAR and COPY.

CLEAR and COPY are both immediate execute commands. Press the key, and ZOT! the HP-85 obeys. This latest command requires more work on your part. When I ask you later to execute it, you will have to type in the name of the command and press END LINE.

Later in this chapter, you'll give the HP-85 a few calculation examples to solve. Normally these appear only on the screen. With the PRINTALL command in action, these problems and their answers will appear both on the screen and on the printer.

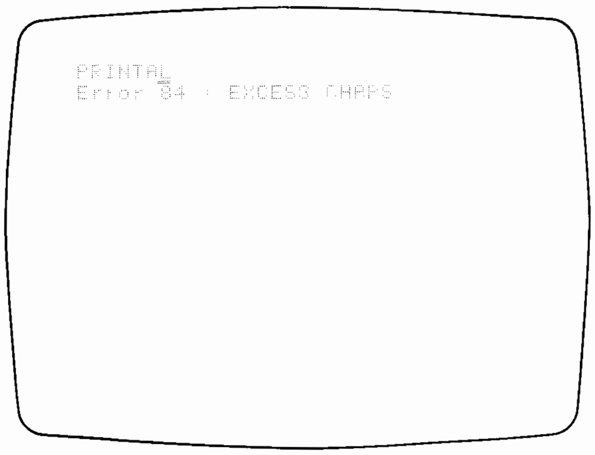
( ) 5. After PRINTALL has performed its service, I'll ask you to get rid of PRINTALL by executing another new command: NORMAL.

( ) 6. It's time once again to deliberately confuse the HP-85 forcing another error message. You will misspell PRINTALL by leaving off the final "L."

( ) 7. Press P R I N T A L END LINE* and generate a beep plus this error message:

```
Error 84 : EXCESS CHARS
```

which means "excess characters." See Figure 24 for a picture of what your screen should look like now.



```
PRINTAL
Error 84 : EXCESS CHARS
```

FIGURE 24
Screen after Step 7

( ) 8. The HP-85 knows it's confused, but doesn't know why. It so happens that PRINTA is a valid instruction, as you'll learn in a later chapter, so you could have intended to type only PRINTA, but by mistake pressed L before pressing END LINE. The HP-85 cannot always read our minds correctly, but often its error messages help us greatly in our efforts to find out what went wrong.

§ Another spelling, also acceptable to the HP-85, is a two word version: PRINT ALL. I like PRINTALL since it's easier to type.

* END LINE is three keys right of L.

(✓) 9. Remember that the HP-85 is patiently waiting for more instructions, so press (L) twice as an easy way to type the final L, then press (END LINE). No beep this time, so all is well.

( ) 10. Get rid of that error reminder by pressing (-LINE); that is, by pressing the (CLEAR -LINE) key.

( ) 11. Before you test the HP-85's math skills, there are four keys you should pay particular attention to: the number one (1), the small L (l), the letter O and the number zero. Their locations are shown in Figure 25.

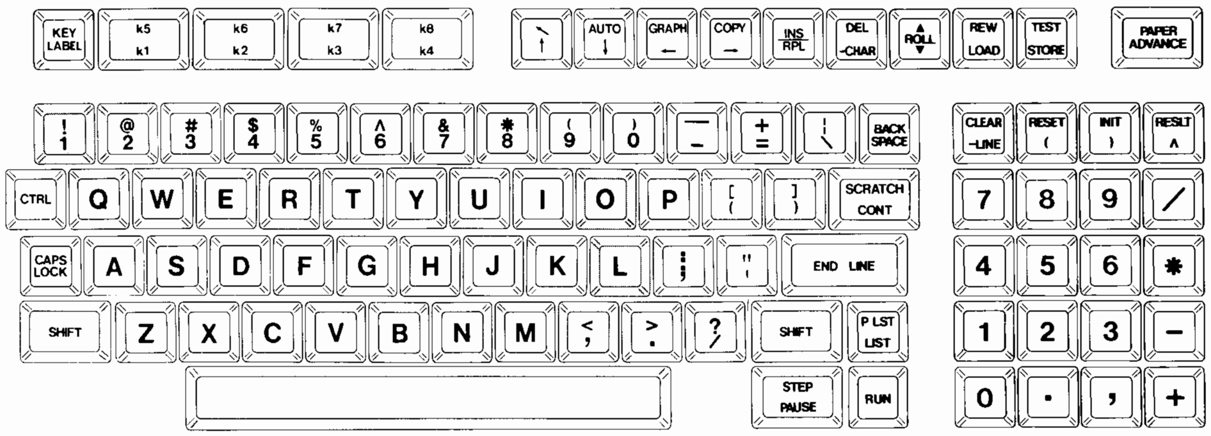


FIGURE 25

Locations of some keys

( ) 12. A small L (l) is not used for the number one (1). If you want a number one, you must press number (1).

( ) 13. Press (SHIFT) + (L) to display a small L (l) on the screen.

( ) 14. Next, press a number one key ((1)). Figure 26 shows enlargements of these characters. Notice that the only difference is in the tops. A small L has a flat top, while the number one has a sloped top. It's very easy to get these mixed up, which is one reason capital letters have been chosen as non-shifted standard letters.

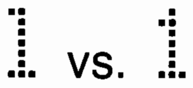


FIGURE 26

l vs. 1

( ) 15. Press the letter (O) key to display a capital O. Now press the number (0) key (above the letter (O) key). Notice that the number, when displayed on the screen, has a line through it (0), to make it distinct from the letter O. It's very common for computers to use 0 for zero.

( ) 16. **IMPORTANT:** Clear this line by pressing (SHIFT) + (BACK SPACE). Otherwise you'll get an unscheduled error message when you perform step 20.

( ) 17. In Chapter 1 you learned to avoid the (END LINE) key when using the HP-85 as a typewriter. For calculations, you *use* (END LINE) *in place of* the = sign. The = sign is very important in BASIC, as you will learn in a later chapter, but *don't use = for calculations.*

( ) 18. (END LINE) is the HP-85's "enter" key. When you want the HP-85 to calculate, memorize, execute, or do some other computer-type thing, type whatever it is you want the HP-85 to do, and press (END LINE). When you press (END LINE), you are "entering" something into the HP-85.

TO "ENTER" SOMETHING INTO THE HP-85 MEANS TO TYPE IT ONTO THE SCREEN AND TO PRESS (END LINE).

( ) 19. When performing calculations, you may use either the numbers in the number pad, Figure 27, or in the top row of the typewriter keyboard. Also, you may use either of the two keys provided for each of the most frequently used math operations: addition, subtraction, multiplication, division and exponentiation (raising to a power). Again, see Figure 27.

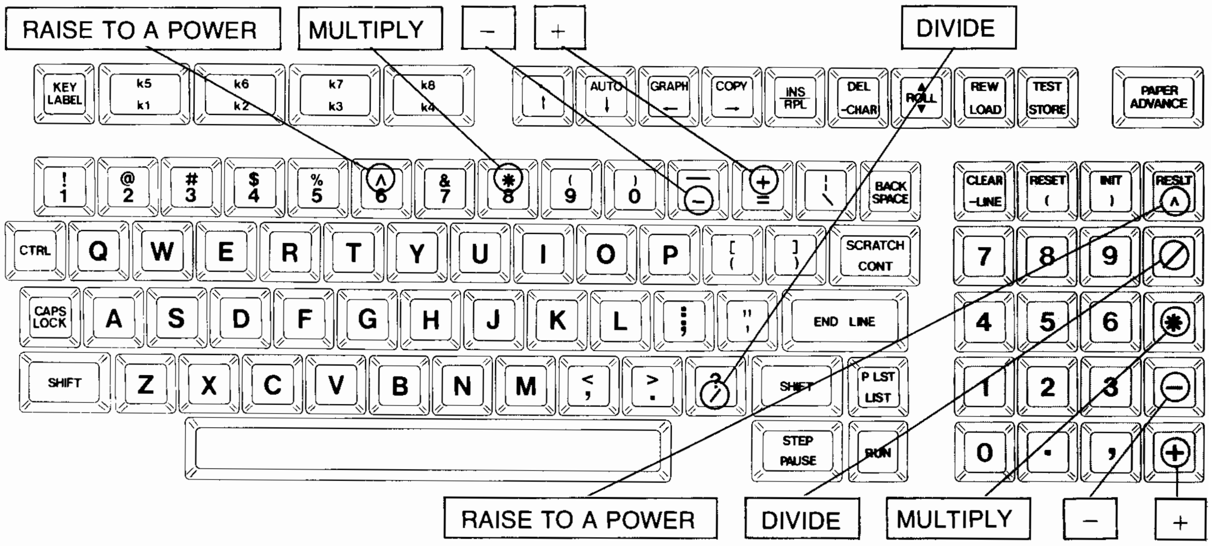


FIGURE 27

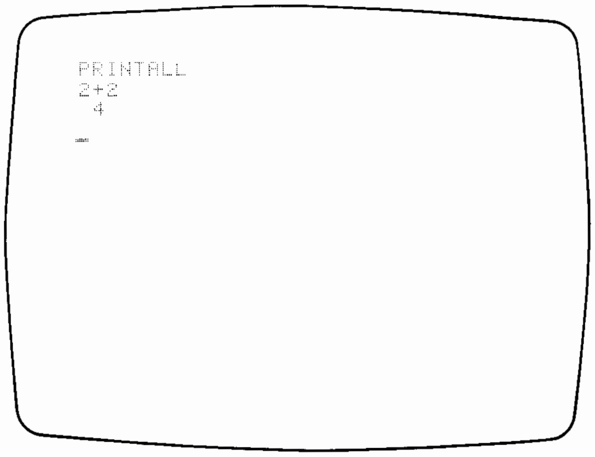
Locations of some math keys

Number Pad

( ) 20. Let's give the HP-85 an easy one to start. Press these keys:

(2)(+)(2)(END LINE)

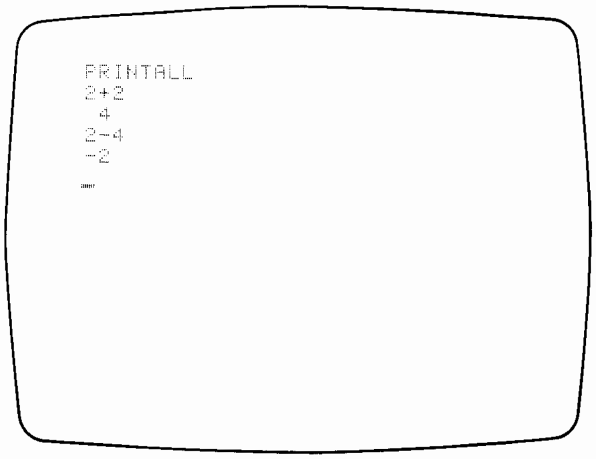
Your screen should look like Figure 28. Also notice that your `PRINTALL` command was obeyed. During printing, the screen turns off to conserve power.



```
PRINTALL
2+2
 4
```

FIGURE 28
Screen after Step 20

√
( ) 21. To see how the HP-85 handles 2 − 4, press (2)(−)(4)(END LINE). Figure 29 shows what your screen should look like now. As you see, the HP-85 is smart enough to leave room for a minus sign in front of the answer if one is needed.



```
PRINTALL
2+2
 4
2-4
-2
```

FIGURE 29
Screen after Step 21

( ) 22. Perform a multiplication. Press (5)(*)(7)(END LINE) and see `35`.

( ) 23. Try a division. Press (1)(3)(/)(6)(END LINE) and see `2.16666666667`. The HP-85 answers with 12 digits unless an exact answer can be shown with fewer digits, as in the earlier examples.

( ) 24. **A WORD OF CAUTION:** Look at the (⤹) key just left of (BACK SPACE). You will not use this key in this course. There is only one thing you should know about (⤹): DO *NOT* USE THE BACKWARD

SLASH, $\boxed{\backslash}$, FOR NORMAL DIVISION. IT MAY GIVE YOU THE *WRONG ANSWER*. § REMEMBER TO ALWAYS USE $\boxed{/}$, NOT $\boxed{\backslash}$, FOR YOUR NORMAL DIVISION PROBLEMS.

√

( ) 25. Now it's time to raise to a power. NOTE: In algebra, it's 10^4. In the HP-85, it's 10^4. Press $\boxed{1}\boxed{0}\boxed{\wedge}\boxed{4}\boxed{\text{END LINE}}$ and see 10000.

( ) 26. Several math operators (+, −, *, / and ^) may be used in the same problem. For example, press $\boxed{5}\boxed{*}\boxed{6}\boxed{/}\boxed{2}\boxed{+}\boxed{3}\boxed{\text{END LINE}}$ and see the answer, 18.

( ) 27. The next example will use parentheses. As you recall, there are three sets of parentheses, as shown in Figure 30, and any of them may be used in calculations.

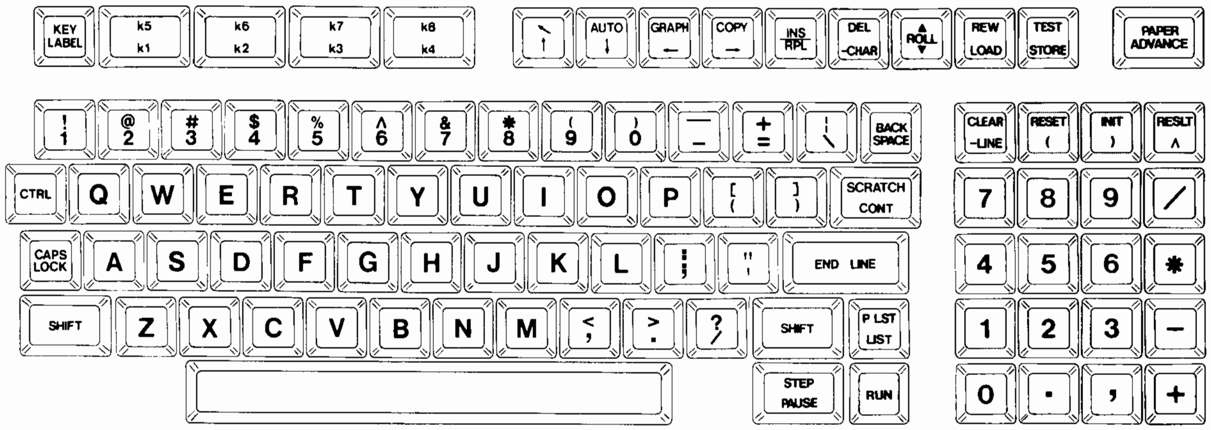


FIGURE 30
Locations of parentheses keys

( ) 28. **Another word of caution.** Look at the $\boxed{\{}\boxed{\}}$ keys just left and above $\boxed{\text{END LINE}}$. The shifted function gives square brackets, which cannot be used in calculations. They have a special purpose in BASIC that you'll learn about in this course. So, DO *NOT* USE [ ] IN CALCULATIONS. *DO* USE ( ).

( ) 29. Multiply the sum of 3 + 2 by 5 by pressing $\boxed{5}\boxed{*}\boxed{(}\boxed{3}\boxed{+}\boxed{2}\boxed{)}\boxed{\text{END LINE}}$. See the answer, 25.

( ) 30. Why the parentheses? Let's try the same problem without them. Press $\boxed{5}\boxed{*}\boxed{3}\boxed{+}\boxed{2}\boxed{\text{END LINE}}$. Why 17 this time and 25 before? The answer to that involves calculation priorities, which I'll discuss in step 31.

§ If you want to know more about $\boxed{\backslash}$, see page 44 in your Owner's Manual under MOD and DIV.

(✓) 31. When the HP-85 calculates an answer, it works from *LEFT to RIGHT*. To completely calculate an answer, this left to right journey may be made many times. On each trip, one or two operations are performed in the following order.

ORDER OF CALCULATION

a. The first thing the HP-85 does when calculating an answer is to perform the calculations inside parentheses, working LEFT to RIGHT.

b. Next, all exponentiation (^) calculations are performed.

c. Then multiplication (*) and division (/).

d. Finally, addition and subtraction.

Furthermore, when an expression within parentheses is evaluated, the calculation priorities are followed: ^ first, then * and /, and finally + and −, until the expression within the parentheses is reduced to a single number.

( ) 32. Relax your fingers. I won't ask you to use them again on the HP-85's keyboard until step 35. Until then, just read. Look at the problem in step 29, 5*(3+2). The first thing the HP-85 did was to calculate the expression within the parentheses. It determined that (3+2) equals 5. Then it calculated 5*5 and got 25. Writing it a different way:

```
                                5*(3+2)
Perform + within ():            5*(  5)
Now the () may be removed:      5*   5
Finally perform *:                   25
```

( ) 33. When the parentheses are removed from 5*(3+2) to give 5*3+2, the HP-85 solves it this way:

```
                 5*3+2
* first:          15+2
then +:             17
```

( ) 34. While we were busy with parentheses, the screen did something quietly that's worth noting. When 5*3+2 was evaluated in step 30, giving 17, the screen scrolled up. The top two lines,

```
PRINTALL
2+2
```

moved up off the screen, and a fresh blank line rolled up from the bottom, where the cursor now sits.

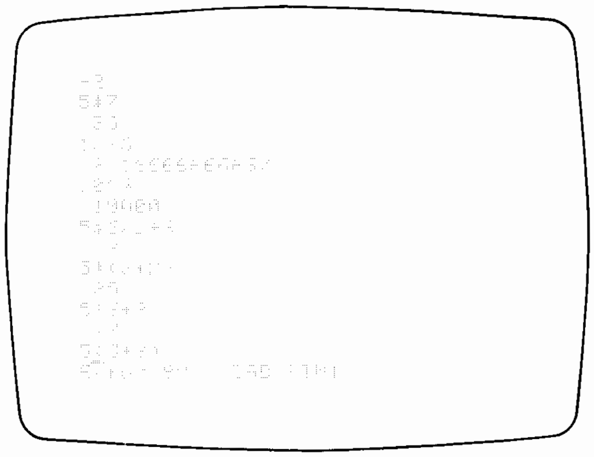
( ) 35. Take another look at the step 29 example. This problem was typed into the HP-85 as 5*(3+2), not 5(3+2). 5(3+2) works in algebra, but not in computers which use BASIC. If you're like me, you'll forget the * from time to time, so why not try leaving it off and see what happens. Press 5 (3 + 2) END LINE, hear the beep, and see

```
Error 88 : BAD STMT
```

which means "bad statement."

Figure 31 shows how your screen should look now.

The HP-85 is guessing that you intended to type a BASIC statement (a BASIC instruction), which could have been true. Again, the HP-85 cannot read your mind. But, as always, the HP-85 is ready to listen, so correct your error as I suggest in step 36.



```
[illegible]
5*7
 35
[illegible]
[illegible]
[illegible]
 1.0E99
[illegible]
[illegible]
5*(4+1)
 25
[illegible]
[illegible]
5(3+2)
Error 88 : BAD STMT
```

FIGURE 31
Screen after Step 35

(✓) 36. Your cursor is in the right place, so press INS RPL and *, then END LINE. Not only did the correct answer, 25 appear, but the HP-85 courteously wiped out the error message as well. The HP-85's courtesy did not end there. It also put you back into the normal REPLACE mode (only one cursor).

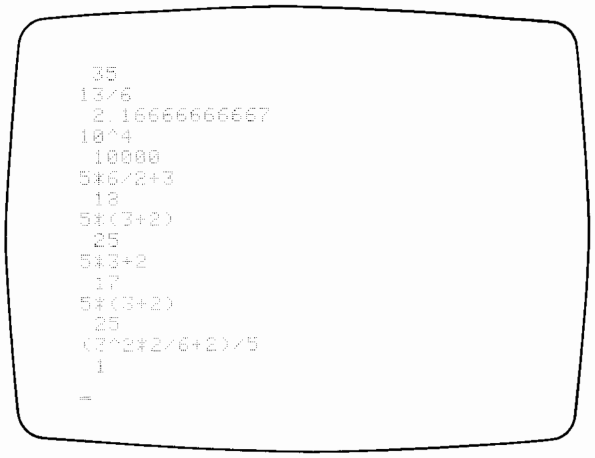
( ) 37. Here's an example with all six levels of priorities in it:

$$(3^2 \times \frac{2}{6} + 2) \div 5$$

To ask the HP-85 for the answer, press

(3 ^ 2 * 2 / 6 + 2) / 5 END LINE

Figure 32 shows how your screen should look now.



```
 35
13/6
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^2*2/6+2)/5
 1
_
```

FIGURE 32

Screen after Step 37

(√) 38. Here's how the example in step 37 is solved.

	`(3^2*2/6+2)/5`
^ within ( ) first:	`(  9*2/6+2)/5`
* and / within ( ) second:	`(   18/6+2)/5`
	`(      3+2)/5`
+ within ( ) next:	`(        5)/5`
Now remove ( ) for clarity:	`5 /5`
and perform /:	`1`

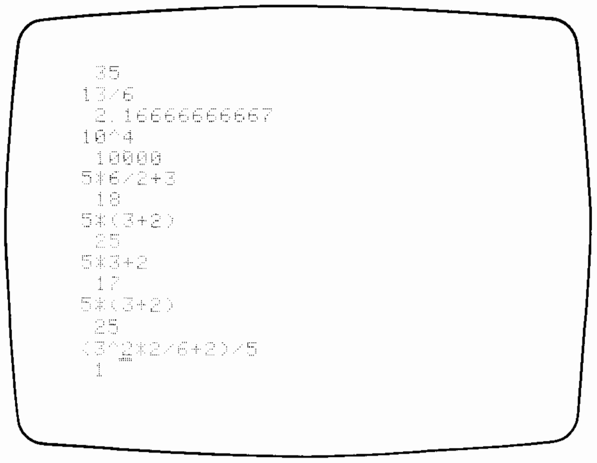
( ) 39. Now grasp the editing tools you acquired in Chapter 1. I'm going to show you one of the HP-85's goodies.

Don't type in this next example. Just look at it and keep reading.

$$(3^3 \times \frac{2}{6} + 2) \div 5$$

If this problem looks familiar, there's a good reason. It's identical to the example in step 37 except the exponent is changed from 2 to 3.

( ) 40. No need to type this problem onto your screen to solve it. Move your cursor to the exponent `2` in your step 37 example. Your screen should now look like Figure 33.

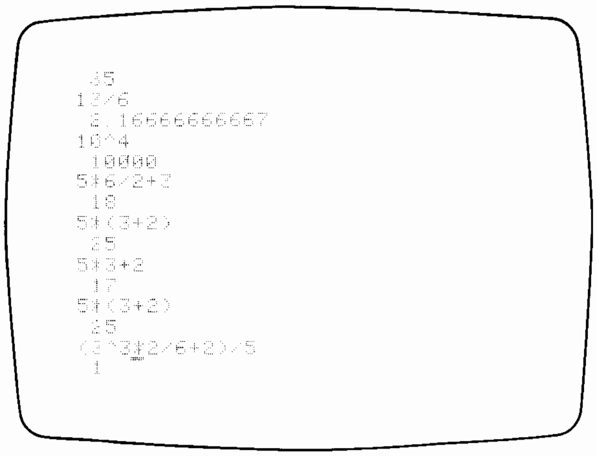


```
 35
13/6
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^2*2/6+2)/5
 1
```

FIGURE 33

Screen after Step 40

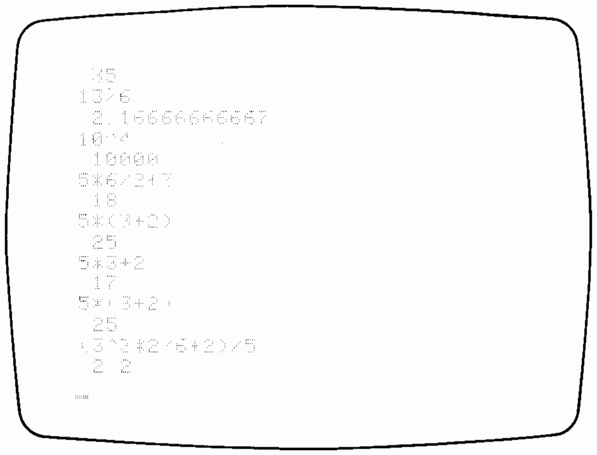
√
( ) 41. Type a 3. Figure 34 shows how your screen should look now.



```
 35
13/6
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^3*2/6+2)/5
 1
```

FIGURE 34

Screen after Step 41



```
 35
13/6
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^3*2/6+2)/5
 2.2
```

FIGURE 35

Screen after Step 42

( ) 42. Now, without moving your cursor, press (END LINE), and presto! Your new answer, 2.2, appears. See Figure 35.

( ) 43. Notice what happened on your printer. Even though you didn't retype the problem—you only changed one number—when you pressed (END LINE), the HP-85 considered the entire line to be as fresh as if you had just typed it in. That's why the printer printed the entire problem. Also, that's why the HP-85 calculated a new, correct answer. This leads to an

IMPORTANT TRUTH

When you press (END LINE), WHAT YOU SEE IS WHAT YOU GET. The line identified by your cursor is entered into the HP-85, no matter where it came from or where on the screen it appears.

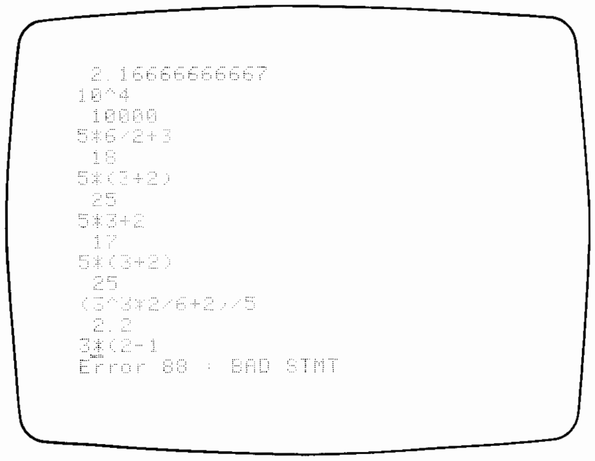
This ability to use displayed characters over and over again is an important time saver, not only in calculations, but in writing and editing BASIC programs. But I'm getting ahead of my story.

√

( ) 44. Let's get back to parentheses. You can use more than one pair of parentheses in a problem. Just remember: you must always use pairs. The number of right facing parentheses should equal the number of left facing parentheses. Otherwise, a friendly error message will remind you of your oversight when you press END LINE. To see this error, use the following keystrokes to enter this expression, which omits the right parenthesis:

3(2 − 1

Press 3 * (2 − 1 END LINE and see the `BAD STMT` error message shown in Figure 36.



```
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^3*2/6+2)/5
 2.2
3*(2-1
Error 88 : BAD STMT
```

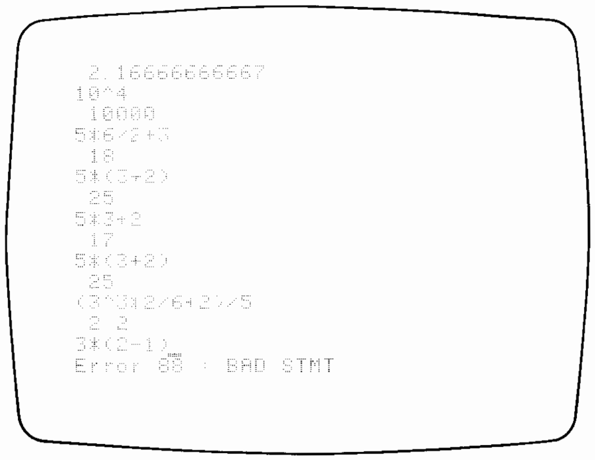
FIGURE 36

Screen after Step 44

( ) 45. Now use your editing tools to add the right hand parenthesis to get this problem.

3(2 − 1)

Figure 37 shows how your screen should look now, *before* you press END LINE.



```
 2.16666666667
10^4
 10000
5*6/2+3
 18
5*(3+2)
 25
5*3+2
 17
5*(3+2)
 25
(3^3*2/6+2)/5
 2.2
3*(2-1)_
Error 88 : BAD STMT
```

FIGURE 37

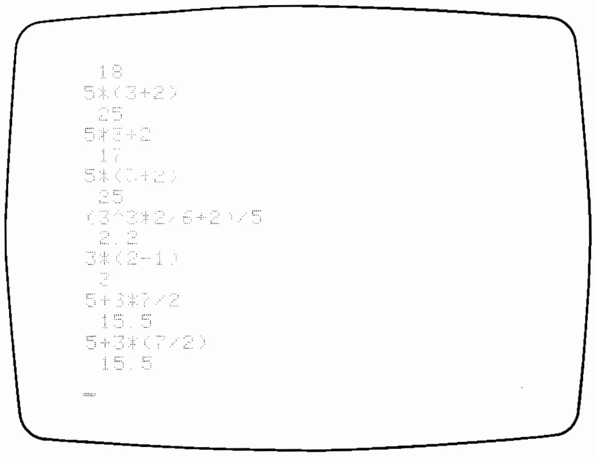
Screen after Step 45

√
( ) 46. Press [END LINE] and see the answer, 3, replace the error message.

( ) 47. You may use more pairs of parentheses than the HP-85 requires if you wish. Sometimes humans like to add parentheses to make problems clearer. To illustrate this, first enter a problem with no extra parentheses:

$$5 + 3\left(\frac{7}{2}\right)$$

Press [5][+][3][*][7][/][2][END LINE] and see the answer 15.5. Now enter the same problem, but add an extra pair of parentheses. Press [5][+][3][*][(][7][/][2][)][END LINE]. Your screen should look like Figure 38 showing the same answer, 15.5.



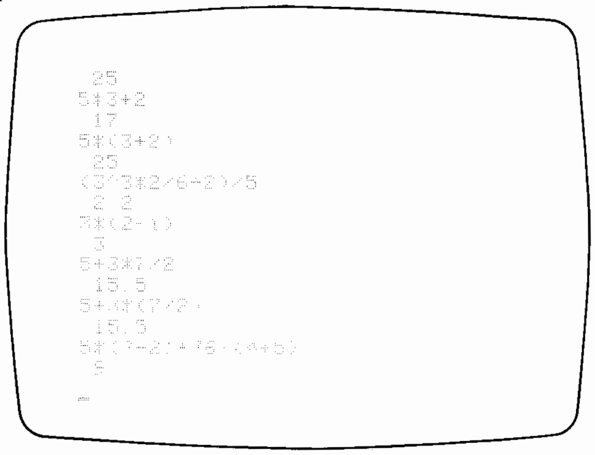
```
  18
5*(3+2)
  25
5*3+2
  17
5*(3+2)
  25
(3^3*2/6+2)/5
  2.2
3*(2-1)
  3
5+3*7/2
  15.5
5+3*(7/2)
  15.5
_
```

FIGURE 38
Screen after Step 47

( ) 48. Here's an example of a problem requiring two pairs of parentheses:

$$5(3-2) + \frac{36}{4+5}$$

Solve this problem using these keystrokes: [5][*][(][3][-][2][)][+][3][6][/][(][4][+][5][)][END LINE] and see the answer, 9, as in Figure 39. Many pairs of parentheses may be used in this way, each pair separated from the next.



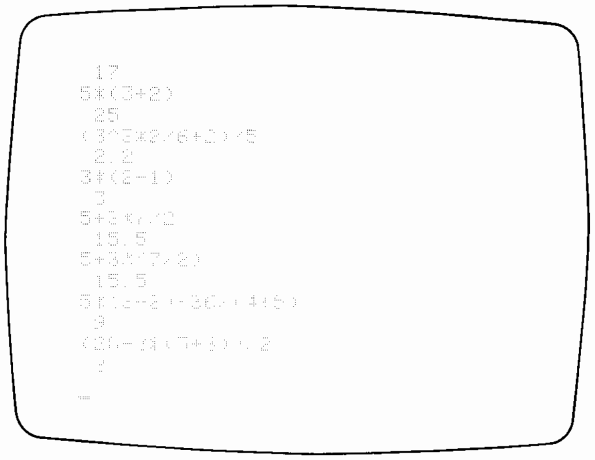
```
  25
5*3+2
  17
5*(3+2)
  25
(3^3*2/6+2)/5
  2.2
3*(2-1)
  3
5+3*7/2
  15.5
5+3*(7/2)
  15.5
5*(3-2)+36/(4+5)
  9
_
```

FIGURE 39
Screen after Step 48

√
( ) 49. Two or more pairs of parentheses may also be used in another way, called "nesting." To "nest" means to put a pair of parentheses inside another pair, as shown by this example:

$$(28 - 3(5 + 3))/2$$

Enter this problem by pressing (2 8 − 3 * (5 + 3)) / 2 END LINE and see the answer, 2, as shown in Figure 40.



```
 17
5*(3+2)
 25
(3^3*2/6+2)/5
 2.2
3*(2-1)
 3
5+3*7/2
 15.5
5+3*(7/2)
 15.5
5*(3-2)+36/(4+5)
 9
(28-3*(5+3))/2
 2
```

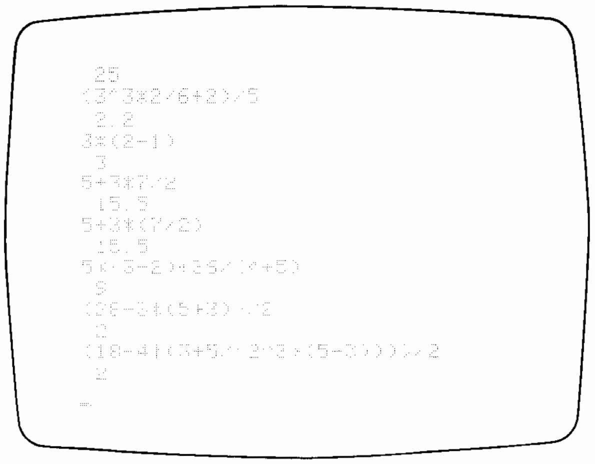
FIGURE 40

Screen after Step 49

( ) 50. There is no practical limit to how deeply you can nest parentheses. That is, you may have one pair of parentheses nested inside another pair, which, in turn, is nested inside another pair, which, in turn, is ... and so on. For example, this problem shows parentheses nested four deep:

$$\frac{18 - 4\left(3 + \frac{5}{2^3 + (5 - 3)}\right)}{2}$$

Press (1 8 − 4 * (3 + 5 / (2 ^ 3 + (5 − 3)))) / 2 END LINE and see 2, the answer shown in Figure 41.



```
 25
(3^3*2/6+2)/5
 2.2
3*(2-1)
 3
5+3*7/2
 15.5
5+3*(7/2)
 15.5
5*(3-2)+36/(4+5)
 9
(28-3*(5+3))/2
 2
(18-4*(3+5/(2^3+(5-3))))/2
 2
```

FIGURE 41

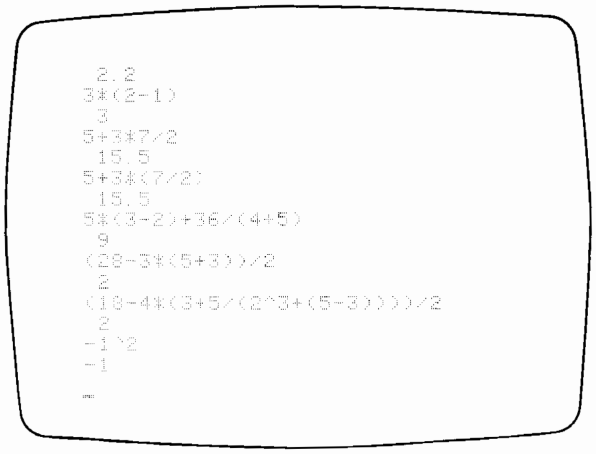
Screen after Step 50

(✓) 51. Here is how the problem in step 50 is evaluated:

a.	Original problem:	`(18-4*(3+5/(2^3+(5-3))))/2`
b.	Evaluate the innermost `()` first:	`(18-4*(3+5/(2^3+(   2))))/2`
c.	Now the innermost `()` may be removed:	`(18-4*(3+5/(2^3+    2 )))/2`
d.	First, perform `^` within the `()` which are now innermost:	`(18-4*(3+5/(   8+    2 )))/2`
e.	Next, perform + within `()`:	`(18-4*(3+5/(        10 )))/2`
f.	Now these `()` may be removed:	`(18-4*(3+5/         10  ))/2`
g.	Now perform `/` within the innermost `()`:	`(18-4*(3+            .5  ))/2`
h.	Next, perform + within the innermost `()`:	`(18-4*(             3.5  ))/2`
i.	Now these `()` may be removed:	`(18-4*              3.5   )/2`
j.	Now perform `*` within the remaining `()`:	`(18-                 14   )/2`
k.	And perform `-`:	`(                     4   )/2`
l.	Remove the `()`:	`4    /2`
m.	And divide to get the answer:	`2`

( ) 52. Parentheses and negative numbers deserve some attention. Enter this simple looking problem: -1^2

Press [-] [1] [^] [2] [END LINE] and see the answer, `-1` (Figure 42).



```
 2.2
3*(2-1)
 3
5+3*7/2
 15.5
5+3*(7/2)
 15.5
5*(3-2)+36/(4+5)
 9
(28-3*(5+3))/2
 2
(18-4*(3+5/(2^3+(5-3))))/2
 2
-1^2
-1
```

FIGURE 42
Screen after Step 52

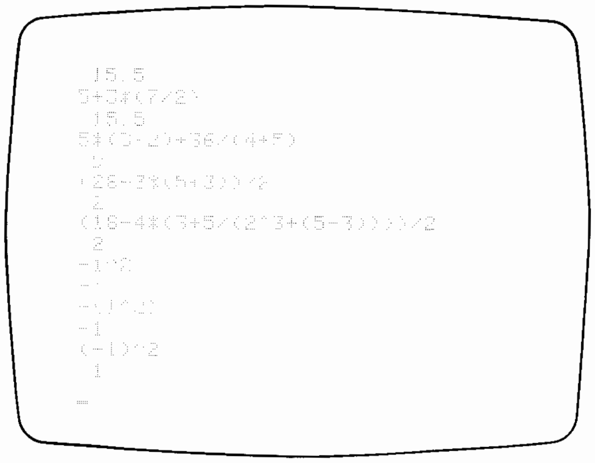
But isn't $-1 \times -1 = +1$? Yes, it is. This is what's happening: the HP-85 sees `-1^2` as `-(1^2)`.

√
( ) 53. To demonstrate this, press (−)(()(1)(^)(2)())(END LINE) and see the same answer, −1.

( ) 54. How do you square a negative one? In other words, how do you enter an exponent problem that multiplies a negative one by a negative one? Here's how:

```
(-1)^2
```

Enter this problem (don't forget to press (END LINE)—remember, "Enter" means type it in and press (END LINE)). See a positive one as the answer, as shown in Figure 43.



```
 15.5
5+3*(7/2)
 15.5
5*(3-2)+36/(4+5)
 9
(26-3*(5+3))/2
 1
(18-4*(3+5/(2^3+(5-3))))/2
 2
-1^2
-1
-(1^2)
-1
(-1)^2
 1
_
```

FIGURE 43
Screen after Step 54

( ) 55. Sometimes the HP-85 doesn't need parentheses for negative numbers, but I suggest you use them always.

( ) 56. That's all the calculating for now. You'll see more in later chapters when you write programs that perform calculations.

( ) 57. To cancel `PRINTALL`, execute the `NORMAL` command:

Press (N)(O)(R)(M)(A)(L)(END LINE).

With its dying gasp, `PRINTALL` prints `NORMAL`.

( ) 58. If you wish to confirm that `PRINTALL` has indeed retired, you may execute `NORMAL` again. Move your cursor up one line to `NORMAL`, press (END LINE), and the printer remains silent. `NORMAL` was executed a second time, as shown by your cursor moving to the line below `NORMAL` when (END LINE) was pressed.

( ) 59. `NORMAL` cancels a few other commands as well. You'll learn which ones some chapters ahead.

( ) 60. Find the (PAPER ADV) key at the extreme upper right corner of the keyboard. To advance your paper so you can tear off your record, press (PAPER ADV) a few times, or hold it down for a second or two to get its repeating action.

Summary of Chapter 2

- PRINTALL: **A command**

  When P R I N T A L L END LINE is pressed, everything that is entered into the HP-85, like a math problem, is printed as well as displayed. In addition, any message or result, like an error message or the answer to a math problem, is also printed and displayed.

  Reference: Page 2-27, step 4.

- **Error message:** Gives one of many possible valid reasons for the error. Reference: Page 2-27, steps 6-8.

- **Do not use a small L (l) for the number one (1).** Reference: Page 2-28, step 12.

- **The small L (l) and the number one (1) are not easy to tell apart.** Reference: Page 2-28, step 14.

- **The number zero (0) looks like a capital letter O with a line through it.** Reference: Page 2-29, step 15.

- **Use END LINE, not =, to get answers to math problems.** Reference: Page 2-28, step 17.

- **END LINE is the HP-85's "enter" key.** Reference: Page 2-29, step 18.

- **ENTER means type into the HP-85 and press** END LINE. Reference: Page 2-29, step 18.

- **Math operators**

  + addition
  - subtraction
  * multiplication
  / division
  ^ exponentiation

  Reference: Page 2-29, step 19.

- **Do NOT use \ for ordinary division.** It may give you the *WRONG ANSWER*. Reference: Page 2-30, step 24.

- **Do NOT use square brackets [ ] in math problems.** Reference: Page 2-31, step 28.

- **DO use curved parentheses ( ) in math problems.** Reference: Page 2-31, step 28.

- **Order of Calculation**

  1. Evaluate ( ) first.
  2. Perform ^.
  3. Perform * and /.
  4. Perform + and - last.

  Reference: Page 2-32, step 31.

- **Method of calculation**

  Each kind of calculation is performed as the entire expression is read *LEFT to RIGHT*, starting with ^ within all unnested ( ) and within all innermost ( ). The process is repeated as many times as necessary to complete all ^ operations within all ( ). Then * and / within ( ) are calculated in the same way, followed by + and - within ( ). Finally, after all ( ) expressions have been calculated, all remaining ^ are calculated, followed by * and /, then + and -.

  Reference: Pages 2-32–2-34 and 2-36–2-40, steps 31-38 and 44-55.

- **To multiply, for instance, the sum of 3 + 2 by 5:**

  Do *NOT* use 5(3 + 2). That works fine for algebra, but *NOT* for the HP-85.

  DO use 5*(3+2) or (3+2)*5.

  Reference: Page 2-32, step 35.

- **When you press (END LINE), WHAT YOU SEE IS WHAT YOU GET,** no matter when it was typed, or where on the screen it appears. Reference: Pages 2-34–2-35, steps 39-43.

- **In a math expression, the numbers of right and left facing parentheses must be equal.** Reference: Page 2-36, step 44.

- **For clarity, you may use more pairs of parentheses than the HP-85 needs.** Reference: Page 2-37, step 47.

- **To nest parentheses** means to put one pair of parentheses inside another pair. Nesting parentheses three deep, for instance, means using them as shown by this example:

  ```
  4/(((6+3)/2-4)*5+3)
  ```

  Reference: Page 2-38, steps 49, 50.

- **Raising a negative number to a power**

  (-) (1) (^) (2) (END LINE) gives the same result as
  (-) (() (1) (^) (2) ()) (END LINE), which gives -1.
  (() (-) (1) ()) (^) (2) (END LINE) gives positive 1.

Saying the same thing without the key cap symbols:

`-1^2` gives `-1`
`-(1^2)` gives `-1`
`(-1)^2` gives `1`

Reference: Pages 2-39–2-40, steps 52-55.

- `NORMAL`: **A command**

  Cancels some commands, including `PRINTALL`.

  Reference: Page 2-40, steps 57-59.

- **The $\boxed{\text{PAPER ADV}}$ key**

  Pressing $\boxed{\text{PAPER ADV}}$ advances blank paper out of the printer.

  Reference: Page 2-40, step 60.

Review Test for Chapter 2

These reviews are intended to help you learn. Don't be concerned if you miss a few questions. You'll find out where a little more study might help.

Try to understand each chapter well before going on. Such good preparation will make the future chapters easier and more fun.

Note: Before you start, execute a command that will print everything that appears on the screen. Otherwise some of your answers will be lost during your completion of this review test.

Answer all questions, then see the answers starting on page 2-46.

1. Your friend tried to solve five math problems on the HP-85. After failing every time, your friend asks for your help.

   a. Your friend tried to solve this problem:

$$5 \times \frac{9}{6} + 30 - 5$$

   by pressing $\boxed{5}\boxed{*}\boxed{9}\boxed{/}\boxed{6}\boxed{+}\boxed{3}\boxed{0}\boxed{-}\boxed{5}\boxed{=}\boxed{\text{END LINE}}$.

   The HP-85 beeped and displayed `Error 88 : BAD STMT`.

   Now it's your chance to show off. Type this problem correctly and show your friend the right answer.

b. The next problem your friend had trouble with was this:

$$\left(6+\frac{27}{3^2}\right)170$$

Pressing $\boxed{(}\boxed{6}\boxed{+}\boxed{2}\boxed{7}\boxed{/}\boxed{3}\boxed{\wedge}\boxed{2}\boxed{)}\boxed{1}\boxed{7}\boxed{0}\boxed{\text{END LINE}}$ gave the same `Error 88 : BAD STMT` plus the beep.

Help your friend by typing this problem correctly and getting the correct answer.

c. Your friend had no better luck with this one:

$$25[12 + 3 \times 41] - 10$$

Pressing $\boxed{2}\boxed{5}\boxed{*}\boxed{[}$§$\boxed{1}\boxed{2}\boxed{+}\boxed{3}\boxed{*}\boxed{4}\boxed{1}\boxed{]}$§$\boxed{-}\boxed{1}\boxed{0}\boxed{\text{END LINE}}$ gave the same old beep and the same old error 88.

Enter this problem correctly to get the right answer.

d. Now your friend was getting a little frustrated. Surely a simple problem would work, like this one:

$$\frac{3}{2}$$

Your friend pressed $\boxed{3}\boxed{\backslash}$*$\boxed{2}\boxed{\text{END LINE}}$, and at first was elated, since no beep nor error message greeted him when he pressed $\boxed{\text{END LINE}}$. But elation deflated to gloom when the answer, `1`, was shown. Show your friend how to get `1.5`.

e. Perhaps an even simpler problem is the answer, your friend hoped, so this was tried

$$1 + 20$$

When your friend displayed `l+2O` and pressed $\boxed{\text{END LINE}}$, the HP-85 produced another message: `Error 92 : SYNTAX`. By `SYNTAX`, the HP-85 is saying "You've given me a bunch of characters that don't make sense together."

Inspect your friend's keystrokes carefully before entering this problem yourself. Two of the four characters are wrong.

2. Solve this problem by moving your cursor, typing only one character, and pressing $\boxed{\text{END LINE}}$. HINT: See problem 1c.

$$15(12 + 3 \times 41) - 10$$

§ $\boxed{[}$ means $\boxed{\text{SHIFT}}$ + $\boxed{\{ [}$. $\boxed{]}$ means $\boxed{\text{SHIFT}}$ + $\boxed{\}]}$.

* $\boxed{\backslash}$ means the $\boxed{| \backslash}$ key.

3. Now ask the HP-85 to solve these problems.

   a. $75 - \frac{23}{(4 + 42)}$

   b. $\frac{15^5}{2000} - \frac{11}{16}$

   c. $(13 + 3)^3 - 255$

   d. $10\left(5 - \frac{3^4}{15 - 3(37 - 14)}\right)$

See the next page for answers.

Answers to Review Test Questions for Chapter 2

1. a. Do not use [=] in calculations. After typing the last [5], press [END LINE] and see the answer 32.5. Reference: Page 2-29, step 17.

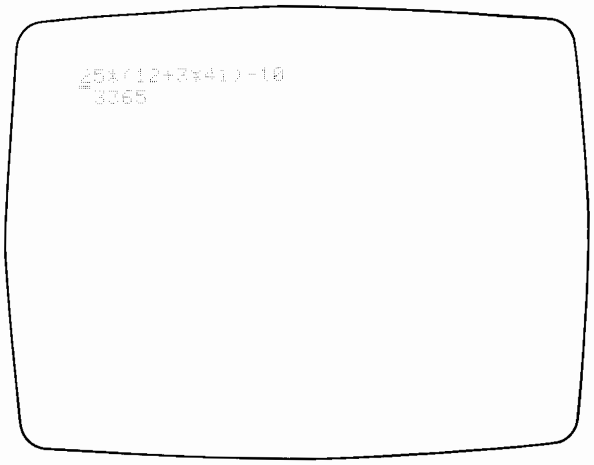
   b. This problem should be entered with [*] between [)] and [1][7][0][END LINE]. The correct answer is 1530. Reference: Page 2-32, step 35.

   c. Square brackets [[] []] are not used in calculations. Use rounded brackets [(] [)] instead. See the answer: 3365. Reference: Page 2-31, step 28.

   d. Do not use the backward slash [\] for ordinary division. Using [/] gives 1.5. Reference: Page 2-30, step 24.

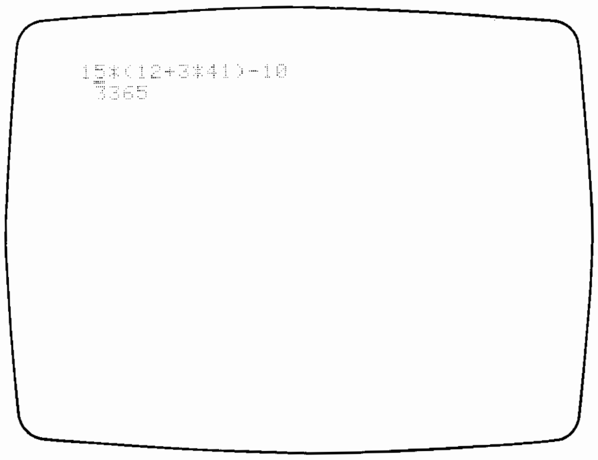
   e. Use one (1), not a small L (l). Also, use zero (0), not a capital letter O (O). With these changes, the correct 21 is seen when [END LINE] is pressed. Reference: Page 2-28, steps 12 and 15.

2. To solve this problem, move your cursor to the first numeral of problem 1c, like this:



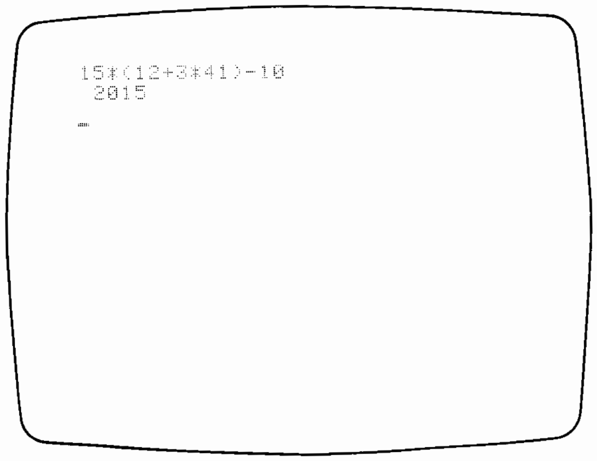
```
25*(12+3*41)-10
 3365
```

Press (1), and see:



```
15*(12+3*41)-10
 3365
```

Finally, press (END LINE) and get the answer:



```
15*(12+3*41)-10
 2015
```

Reference: Pages 2-34–2-35, steps 39-43.

3. Here is a correct `PRINTALL` record:

```
75-23/(4+42)
 74.5
15^5/2000-11/16
 379
(13+3)^3-255
 3841
10*(5-3^4/(15-3*(37-14)))
 65
```

Reference: Most pages and steps of Chapter 2.

Chapter 3

The Tape Cartridge and BASIC

Preview

In Chapter 3, you will learn:

- How to use your tape cartridge.
- How to overcome 5 problems that can prevent the execution of a command.
- How to get a program from tape into the HP-85's memory.
- How to run a program.
- What a BASIC statement is.
- What a BASIC program is.
- How to enter a simple program.
- Three BASIC vocabulary words: `DISP`, `END` and `PRINT`.
- How to read the HP-85's mind (how to look at program instructions stored in the HP-85's memory).
- How to change a BASIC statement.

(✓) 1. I suggest you take the Review Test for Chapter 2 again if you've been away from this course for a few days. The editing review this test offers will help you follow easily the adventures I've prepared for you in Chapter 3.

( ) 2. HELP MESSAGE:

> IF YOU NEED HELP
> during this first part of Chapter 3, follow steps a, b and c below.
>
> When you're using your tape cartridge, I'll have other help messages for you.
>
> a. First, check for typing errors. If you find any, use your editing tools to correct them. See the Summary of Chapter 1, page 1-23, for a summary of these tools.
>
> b. If these text editors don't work, clear your screen (SHIFT + CLEAR LINE). Then go back and start at the last step that gives you a convenient starting place.
>
> c. If all else fails, start the chapter over again.

( ) 3. Insert the BASIC Training tape cartridge (see Figure 44). Push it until you hear a click.

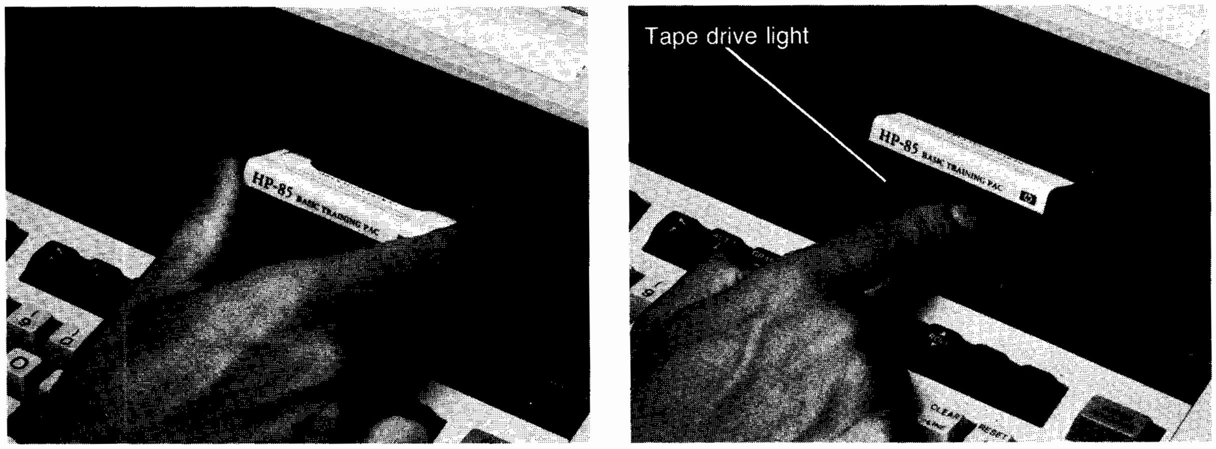


FIGURE 44

How to insert and eject tape cartridge

( ) 4. Press the tape eject bar (Figure 44) and practice removing your cartridge. If you remove a cartridge without pressing the bar, a cartridge cannot be inserted unless the eject bar is pressed first.

( ) 5. Now reinsert your cartridge.

( ) 6. If you turn the HP-85 ON when a tape cartridge is already inserted, the tape drive will run. This is normal. The HP-85 is looking for a special program which might be on the tape named ''Autost'' (auto start). If an ''Autost'' program is on the tape, the HP-85 will find it and start it automatically. Your BASIC Training tape cartridge has no ''Autost'' program, so the HP-85 will search for it, and after learning ''Autost'' is not there, the HP-85 will turn its tape drive off.

( ) 7. To experience this search for ''Autost,'' switch the HP-85 ON. If the HP-85 is already ON, switch OFF, then ON.

( ) 8. When the tape drive light goes out, showing that the tape drive has stopped, you're ready to go on to step 9.

( ) 9. Soon you're going to load your first program. Then you will run it, and I'll instruct you further from the HP-85's screen and printer. This will all happen after you become even more familiar with the HP-85's error messages.

( ) 10. I have led you into trouble before. But this time I have you scheduled for BIG trouble. Five conditions must be met before a typed command will be executed by the HP-85. You're going to violate every one of those five conditions, and then you're going to fix each violation, one by one.

( ) 11. Please do not press any keys until step 14. Steps 12 and 13 are for reading only.

( ) 12. Here are the five conditions that must be met before the HP-85 will execute a typed command.

TO EXECUTE A COMMAND

a. The last character on the preceeding line of your screen must be blank (forget this if you're typing on the first line). DO *NOT* USE THE SPACE BAR TO MAKE THIS BLANK.

b. There must be no characters on the screen to the left of where you start your command. DO *NOT* USE THE SPACE BAR TO MAKE THIS BLANK.

c. Your command must be typed correctly. The correct way to type the `LOAD` command you'll be using soon is:

```
LOAD "CH3"
```

You must type the quotation marks (SHIFT + " ')

d. After the command is typed, the rest of the line must be blank. DO *NOT* USE THE SPACE BAR TO MAKE THIS BLANK.

e. After the command is typed, END LINE must be pressed.

√
( ) 13. Why all the emphasis on *NOT* USING THE SPACE BAR in step 12? A complete answer giving the correct reasons is beyond the scope of this course. The best I can do is say there are two different kinds of blanks. One is a typing character, produced by the space bar. The other is a computer character called a "carriage return." This special kind of blank is the kind the HP-85 looks for when trying to understand your command. That special kind of computer blank is produced only by BACK SPACE, by -LINE and by CLEAR. You will use only two of these, BACK SPACE and -LINE (that is, the unshifted CLEAR -LINE key) to produce the blank spaces called for by the rules in step 12.

( ) 14. Now, set the stage for your horrible mistakes. Hold X down until more than four but less than five full lines are filled with `X` characters. This will take about 10 seconds.

( ) 15. Move your cursor to the 10th position from the left on line three. Figure 45 shows about how your screen should look.

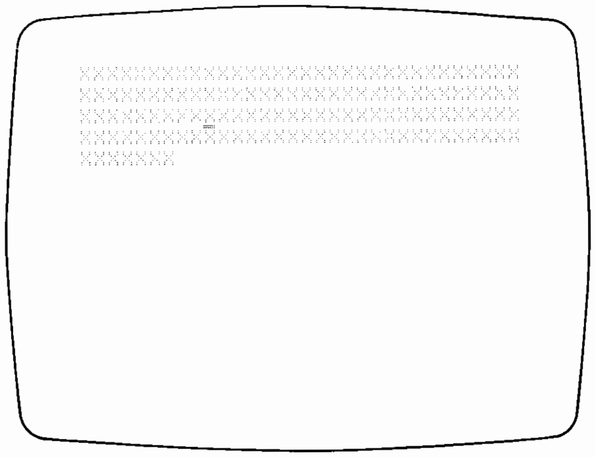


FIGURE 45
Screen after Step 15

(✓) 16. Don't press any keys in steps 16 and 17. Your fingers will get their chance in step 18. Figure 46 gives the location of the REW LOAD key. LOAD is one of several typing aids on the keyboard. Pressing a typing aid puts a word on the screen. No other action occurs. The effect is identical to typing the same word letter by letter. These typing aid keys are used often, and their use saves time. Specifically, pressing LOAD (pressing REW LOAD) puts the word `LOAD` on the screen. The first letter goes above the cursor.

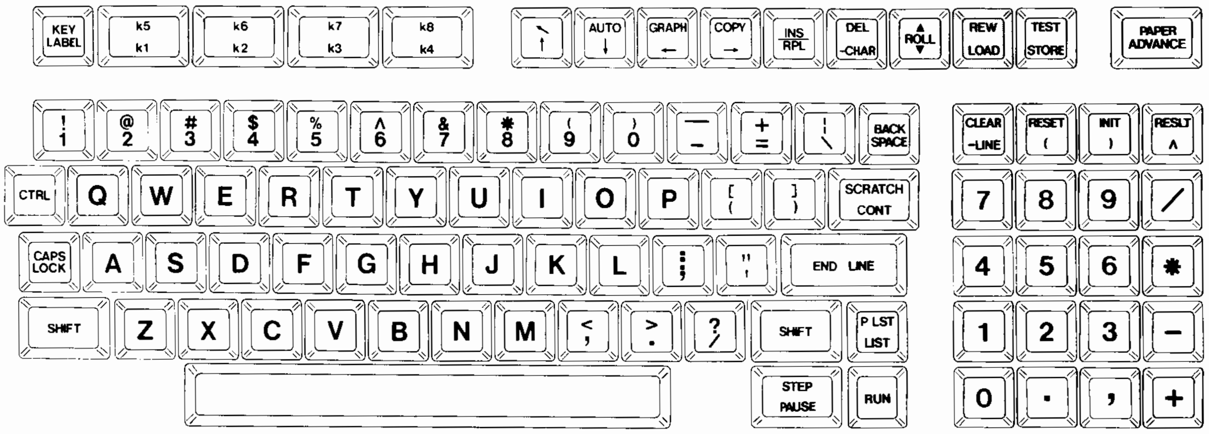
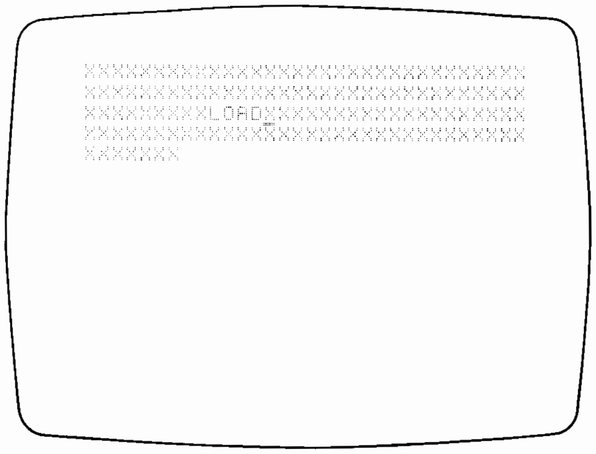


FIGURE 46

Location of REW LOAD

( ) 17. REW stands for REWIND. Pressing SHIFT + REW LOAD executes the `REWIND` command, which rewinds the tape. If you didn't know this, and you pressed in error SHIFT + REW LOAD instead of the REW LOAD key, the result might cause you to panic. Instead of the word `LOAD` being displayed on the screen, the screen would go blank to conserve the HP-85's power and the tape drive and its light would turn on. See Figure 44 for the location of the tape drive light. The tape would rewind (unless it were rewound already), then the light would turn off and the screen would come back to life. If you press SHIFT + REW LOAD by mistake some time, relax. No harm will come to you.

( ) 18. Press LOAD (press REW LOAD). Your screen should look like Figure 47.

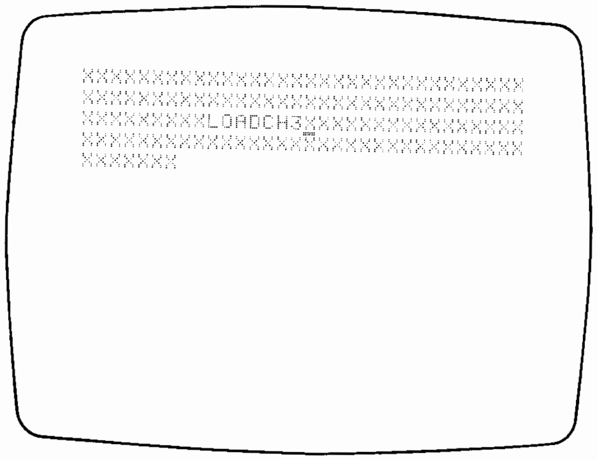


```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXLOADXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXX
```

FIGURE 47

Screen after Step 18

(✓) 19. Now commit your fifth and final mistake. Press (C) (H) (3), and see Figure 48. This shows how your screen should look now.



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXLOADCH3XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXX
```

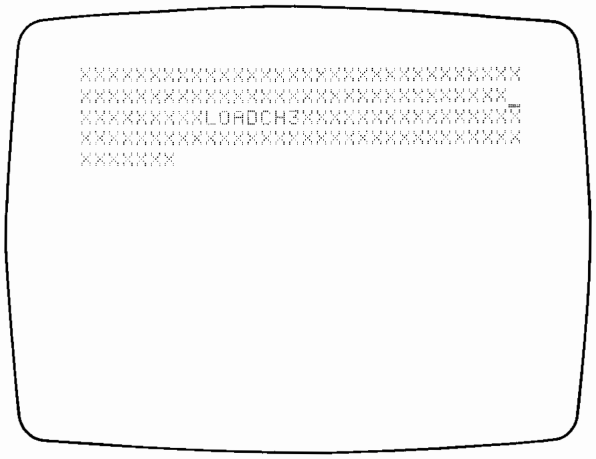
FIGURE 48
Screen after Step 19.

( ) 20. You have just violated all five rules laid down in step 12.

( ) 21. DO NOT CLEAR YOUR SCREEN, although ordinarily a mess like this would be wiped away quickly with a simple screen clearing.

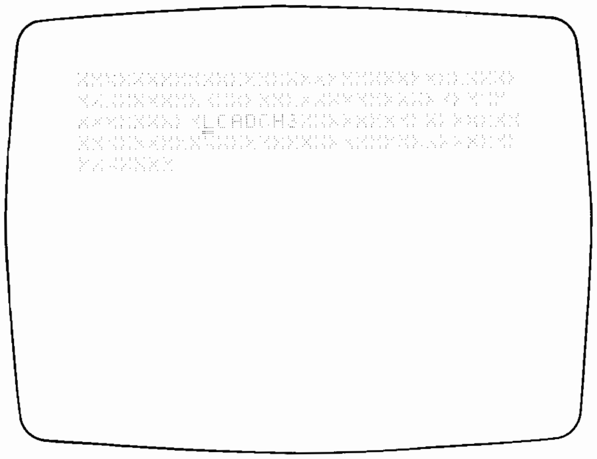
( ) 22. You will eliminate each of these violations of the five command requirements in the same order they're listed in step 12.

( ) 23. First, you will get rid of the last character on the preceeding line, line 2. Press (←) 15 times to put your cursor at the right end of the third line. Then press (↑) once. Finally, press (-LINE), and get the screen shown in Figure 49.



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXLOADCH3XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXX
```

FIGURE 49
Screen after Step 23



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX XLOADCH3XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXX
```

FIGURE 50
Screen after Step 24

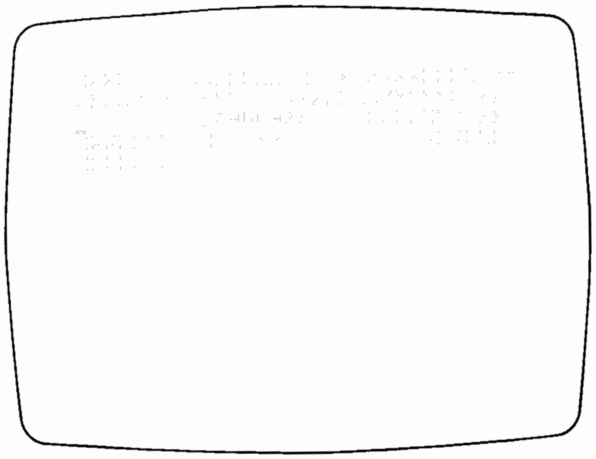


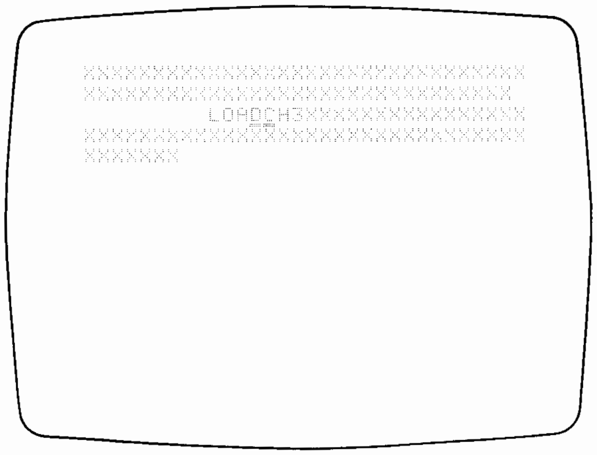
FIGURE 51
Screen after Step 25

(✓) 24. Next, you'll get rid of those characters to the left of where your command starts. Using (←) and (↓), move your cursor to the L of LOAD. See Figure 50.

( ) 25. Now press (SHIFT) + (BACK SPACE) to erase those X characters in a hurry. See Figure 51. Notice the characters to the left of the cursor disappeared, but the L of LOAD, the character above the cursor, remained. This is different from the way (-LINE) works, as you'll see shortly. With (-LINE), all characters right of the cursor, plus the character above the cursor, are wiped out. Also, when (-LINE) is pressed, the cursor does not move. Of course, (BACK SPACE), with or without (SHIFT), does move the cursor.

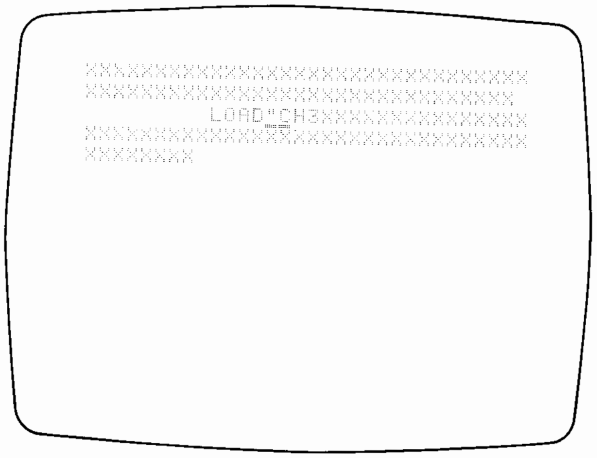
You'll get a better feel for these editing keys as you continue to use them throughout this course.

( ) 26. The third rule requires that the command be typed correctly. To obey this rule, you must add a pair of quotation marks around CH3, the name of the program you wish to load. Use (←) to move your cursor to the C of CH3. Now press (INS/RPL) to get INSERT mode with its double cursor. See Figure 52.



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         LOADCH3XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXX
```

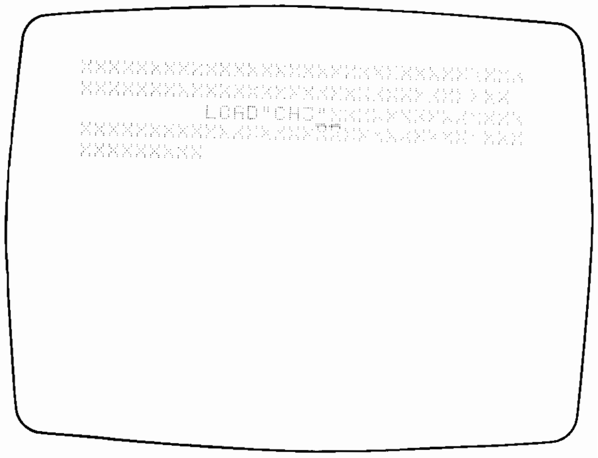
FIGURE 52
Screen after Step 26



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         LOAD"CH3XXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX
```

FIGURE 53

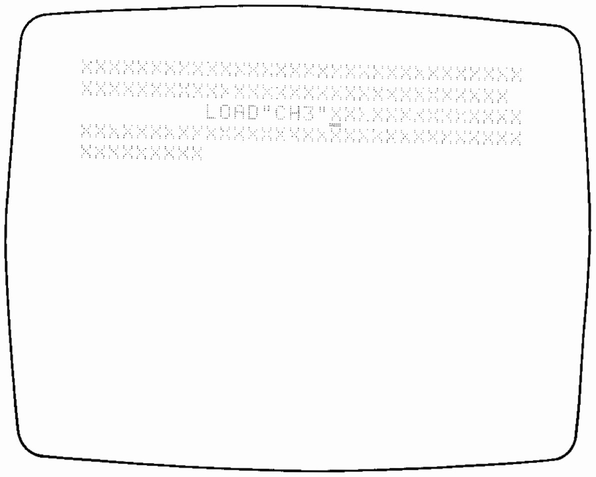
Screen after Step 27.



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         LOAD"CH3"XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX
```

FIGURE 54

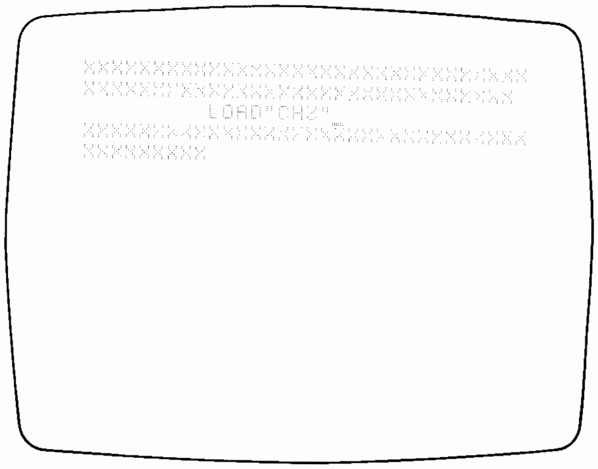
Screen after Step 28



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         LOAD"CH3"XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX
```

FIGURE 55

Screen after Step 29



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         LOAD"CH3"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX
```

FIGURE 56

Screen after Step 30

√

( ) 27. Press SHIFT + " to add the first quotation mark. See Figure 53. Notice that you pushed all the X characters located ahead of your double cursor forward one position.

( ) 28. Next, press → three times to move your double cursor under 3X. Pressing " (don't forget SHIFT) gives Figure 54.

( ) 29. Press INS/RPL to get REPLACE mode, Figure 55, and notice that the cursor is correctly placed for -LINE to wipe out all characters right of the command.

( ) 30. Press -LINE. Figure 56 shows the result. Only one command requirement left.

( ) 31. Let me explain what will happen when, five steps later (step 36), you satisfy the final requirement. If you've followed the steps up to here OK, and if you have the BASIC Training cartridge inserted, performing step 36 will cause the tape drive light to turn on. The screen will blank to conserve power, and the tape drive will run. After some seconds, the tape drive will turn off, as shown by the tape drive light turning off. At that time, your program will have been successfully copied from the tape, and it will then rest in the HP-85's memory, ready to serve you.

(✓) 32. Please read this carefully:

CAUTION: *Never remove a tape cartridge while the tape drive is running.* Make sure the yellow tape drive light is off before removing a cartridge. Otherwise, recorded material may be lost.

Also, for the same reason, *never switch the HP-85 OFF when the tape drive is running.*

( ) 33. Figure 57 shows the location of two important keys you'll soon be using, RUN and SCRATCH CONT.

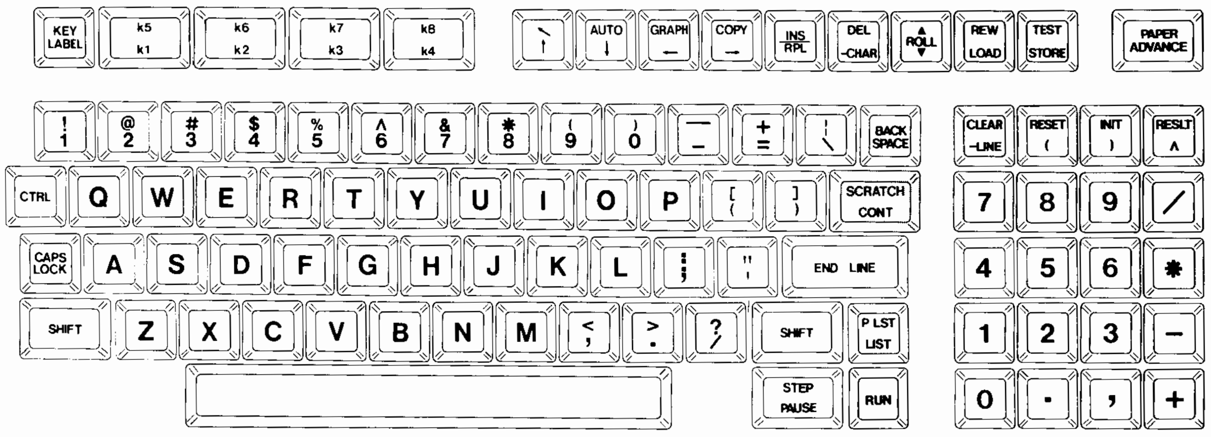


FIGURE 57

Locations of RUN and SCRATCH CONT

( ) 34. I'll be instructing you shortly from the screen and the printer. At that time, you'll often see: TO PROCEED, PRESS CONT.

( ) 35. When I ask you from the screen to press CONT (press the SCRATCH CONT key), you might, if you're like me, hit another key by mistake. Do not worry. Simply press CONT as though nothing happened, and all will be well. Unless you press RUN, that is. In that case, sit back and relax, because the program will start over.

( ) 36. Now, press END LINE. If you get an error message, you may have used your space bar rather than BACK SPACE or -LINE to remove one or more X characters from your screen. If you do get an error message, clear your screen (SHIFT + CLEAR -LINE) and repeat steps 14 through 30, then press END LINE again.

( ) 37. If you got no error message in step 36, your tape drive started when you pressed END LINE. When the tape drive light goes out, program "CH3" is loaded; that is, it's copied into the HP-85's memory.

( ) 38. In step 39 I'm going to ask you to start your "CH3" program but to ignore the screen's request to press CONT. I'm going to get you into and out of trouble again. I'll ask you to press another key that you might press by accident sometime instead of CONT. It will cause feet of paper to pour out of the printer with strange markings all over it. This will be the program listing for program "CH3." When this happens, do not be concerned. Just keep reading and following my instructions in this workbook, and all will be well.

(√) 39. Press RUN. Read the message on the screen, but *DO NOT* press CONT. Since "CH3" is already in the HP-85's memory, pressing RUN caused "CH3" to run.

( ) 40. Press SHIFT + P LST LIST. P LST stands for PRINT LIST. If you have patience, the printer will run for a long time. But you can stop it fast. *Press* CONT. Your trouble is cured.

( ) 41. Continue following the instructions on your screen. You'll be back in this workbook for the Summary and Review Test for Chapter 3.

Summary of Chapter 3

- **To insert a cartridge,** push it until the HP-85 clicks. Reference: Page 48, step 3.
- **For a command to be understood** by the HP-85, five conditions must be satisfied:
  - Last character position of preceding line must be blank.
  - Portion of line in front of command must be blank.
  - Command must be typed correctly.
  - Portion of line following command must be blank.
  - Finally, END LINE must be pressed.

Reference: Pages 3-49–3-50, step 12.

- LOAD: **A command**

  To get a program from tape into the HP-85's memory, press LOAD " [name of program] " END LINE or press L O A D " [name of program] " END LINE. " is obtained with SHIFT + ". LOAD represents the single key REW LOAD while L O A D represents the four letter keys. Pressing the single REW LOAD key or the four keys L O A D gives the identical result: The word LOAD is typed onto the screen.

Reference: Pages 3-51–3-55, steps 16-37.

- REWIND: **A command**

  Pressing REW (pressing SHIFT + REW LOAD) executes immediately the REWIND command. This command rewinds the tape.

Reference: Page 3-51, step 17 and program "CH3" display.

- **When tape drive is running**

  DO *NOT* remove tape cartridge.
  DO *NOT* turn the HP-85 OFF.

Reference: Page 3-55, step 32.

- **To run a program** stored in the HP-85's memory, press (RUN). Reference: Page 3-56, step 39.
- **To protect material recorded on a tape,** snap tape cartridge's RECORD tab in direction away from arrow. Reference: Program "CH3" display.
- **BASIC:** A language with words and grammar. Reference: Program "CH3" display.
- **BASIC statement:** An instruction for the HP-85 using BASIC words. Reference: Program "CH3" display.
- **BASIC program:** A group of statements designed to work together to perform a task. Reference: Program "CH3" display.
- **BASIC vocabulary words**

  DISP: Instructs the HP-85 to display message on screen.
  PRINT: Instructs the HP-85 to print message on printer.
  END: Ends program.

  Reference: Program "CH3" display; also printout, steps 3-20.
- **To prevent two programs from destroying each other,** clear the HP-85's memory before entering another program's statements. Reference: Program "CH3" printout, step 2.
- **The (END LINE) key**
  - Enters program statements into memory.
  - Executes commands.
  - Performs calculations.

  Reference: Program "CH3" printout, step 5, also Chapter 2, page 2-29, step 18.
- **The (LIST) key**

  Lists on screen program statements stored in the HP-85's memory.

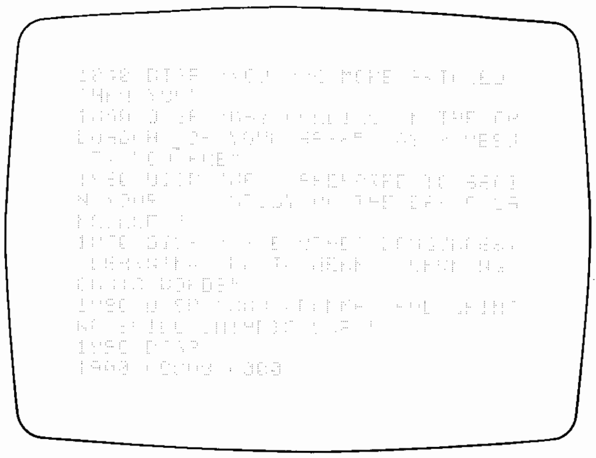
  Reference: Program "CH3" printout, step 13.
- **To change a BASIC statement,** edit or retype the statement using the same statement number and press (END LINE). Reference: Program "CH3" printout, steps 15-20.

Review Test for Chapter 3

Answer all questions, then see the answers on page 3-59.

1. You wish to get a program named "CH3" from your tape cartridge into the HP-85's memory. The correct

cartridge is properly inserted. You find your screen full of characters, but you know you can write over them. So you press: LOAD C H 3 to write over the beginning of line 4. Now your screen looks like this:



Without pressing LOAD C H 3 again, what must you do before this command will be accepted by the HP-85?

2. Faced with the same situation as in question 1, but this time allowing the command to be retyped, what is an easier way to get the HP-85 to accept this command?

3. How can you mechanically protect material recorded on your tape cartridge?

4. For a BASIC program to show a message on the screen, what BASIC word is needed?

5. What is an individual program instruction called?

6. What three major actions are performed by END LINE?

See the next page for the answers.

Answers to Review Test Questions for Chapter 3

1. To load the program:

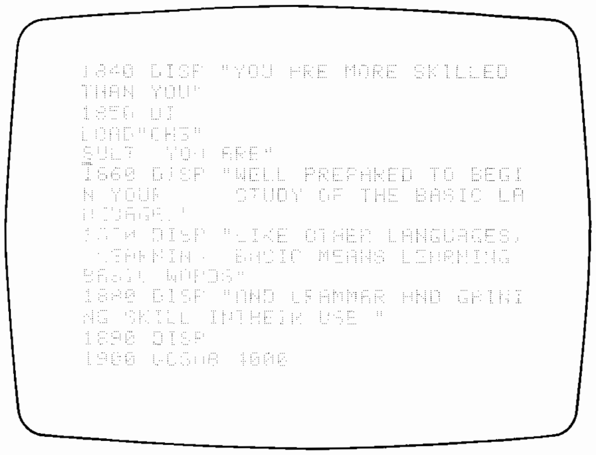
   a. Remove the "P" from the right end of the third line.

   b. Put quotation marks around the program's name.

   c. Remove all characters on the same line that are right of the command.

   d. Press END LINE.

   One way to satisfy the four actions listed above leaves the screen looking as shown below. The last character P and some other characters on line 3 were removed by pressing ↑ once, then pressing -LINE.



```
1840 DISP "YOU ARE MORE SKILLED
THAN YOU"
1850 DI
LOAD"CH3"
SULT  YOU ARE"
1860 DISP "WELL PREPARED TO BEGI
N YOUR       STUDY OF THE BASIC LA
NGUAGE."
1870 DISP "LIKE OTHER LANGUAGES,
 LEARNING  BASIC MEANS LEARNING
BASIC WORDS"
1880 DISP "AND GRAMMAR AND GAINI
NG SKILL IN THEIR USE "
1890 DISP
1900 GOSUB 1000
```

Reference: Pages 3-49–3-50, step 12.

2. a. Clear the screen by pressing CLEAR (by pressing SHIFT + CLEAR -LINE).

   b. Press LOAD " C H 3 " END LINE.

Reference: Page 3-52, step 21. Also Chapter 1, page 1-9, step 15.

3. Snap the RECORD tab away from the arrow direction. Reference: Program "CH3" display.

4. DISP. Reference: Program "CH3" display; also printout, steps 3–10.

5. Statement. Reference: Program "CH3" display; also printout, steps 3–7.

6. Execute commands, enter statements, and perform calculations. Reference: Program "CH3" printout, step 5.

Chapter 4

Write Your First BASIC Program

Preview

In Chapter 4, you will:

- Learn more about memory.
- Learn more about loading programs.
- Learn more about trouble killers.
- Learn more about spaces in BASIC statements.
- Learn how to add, change or remove a statement when writing a program.
- Learn about statement numbers.
- Practice using your new BASIC words, `DISP`, `PRINT`, `END`.
- Learn what's the same and what's different about statements and commands.
- Write, enter, run and list your first program.

( ) 1. If it's been several days since you studied Chapter 3, why don't you take the Review Test for Chapter 3 again to make your work with Chapter 4 easier?

( ) 2. HELP MESSAGE:

IF YOU NEED HELP

while running program "CH4," follow steps a, b and c below.

a. To stop unexpected action, press CONT.

b. If things are still strange, press RUN to start "CH4" over again.

c. If all else fails, perform steps 1 through 4:

1. Make sure tape drive is NOT running.
   The tape drive light should be OFF.
2. Remove BASIC Training cartridge.
3. Switch the HP-85 OFF, then ON.
4. Start Chapter 4 over again.

While following instructions printed by the HP-85, follow IF YOU NEED HELP messages on printout.

While writing, entering and running your "Name-Game" program, follow the IF YOU NEED HELP messages in text.

( ) 3. Insert your BASIC Training cartridge. Push your cartridge into the HP-85 until you hear the click.

( ) 4. If your screen is not clear, clear it (SHIFT + CLEAR -LINE), then press LOAD " C H 4 " END LINE.

( ) 5. When the tape drive light goes out, your "CH4" program has been copied into the HP-85's memory and the tape drive is off.

( ) 6. After you perform step 7, I will be instructing you from the HP-85's screen. When you finish the programming exercises that will be printed by program "CH4," you'll return to these pages to compose your first program.

( ) 7. Press RUN and follow the instructions on the screen.

( ) 8. Now that you've had more programming experience performing the tasks given you by program "CH4," it's time to make some general observations about programs, statements and commands.

( ) 9. The last, highest numbered statement in a program should be an END statement. Otherwise, the program may not run.

( ) 10. To write a statement designed to display a message, use the BASIC word DISP and enclose the message in quotation marks.

( ) 11. To write a statement designed to print a message, use the BASIC word PRINT and enclose the message in quotation marks.

( ) 12. Statements may be entered (typed in and END LINE pressed) in any numerical order.

( ) 13. Remember the five conditions that must be met for the HP-85 to understand a command? Those same conditions must also be satisfied for the HP-85 to accept a statement. Here they are again.

For the HP-85 to understand and accept a command or a statement you attempt to enter, these five conditions must be satisfied.

TO EXECUTE A COMMAND OR STATEMENT

a. Last character position of preceding line must be blank.

b. Portion of line in front of command or statement must be blank.

c. Command or statement must be typed correctly.

d. Portion of line following command or statement must be blank.

e. Finally, END LINE must be pressed.

( ) 14. Also remember: You get rid of all interfering characters at once simply by clearing your screen (SHIFT + CLEAR -LINE).

( ) 15. You may list your program statements whether or not your program is finished. You'll have a chance to try this soon.

( ) 16. BASIC programs may be as simple as two short statements.

( ) 17. Here are important differences between statements and commands, both of which are instructions to the HP-85:

STATEMENTS VS. COMMANDS

Question	Statements	Commands
Part of program?	YES	NO
Use line (statement) numbers?	YES	NO
When executed?	When program is run	Some immediately when key pressed (RUN) Some when END LINE pressed (LOAD)

( ) 18. Now it's time for the "Name-Game" Program.

IF YOU NEED HELP
with steps 19 through 24, see page H-8, No. 9 in the Supplement's HELP Section.

( ) 19. The programs you've entered so far you have, in fact, copied. Now you're going to write your first genuine program! Your program should print this message:

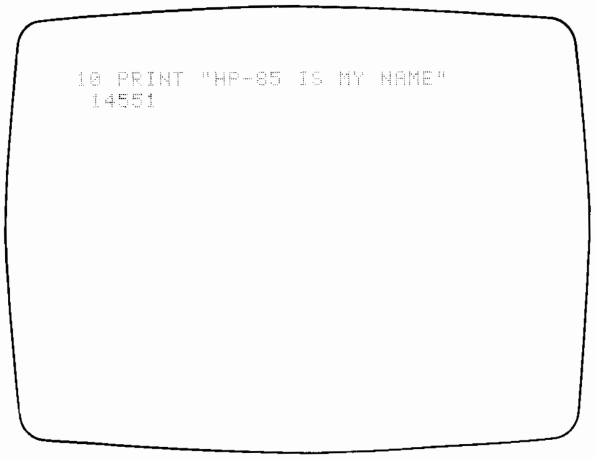
```
HP-85 IS MY NAME
AND SERVING YOU
IS MY GAME.
```

( ) 20. HINT: The first statement may be written as shown below.

```
10 PRINT "HP-85 IS MY NAME"
```

Enter this statement 10 (don't forget END LINE), then perform steps 21 through 24 before completing your program.

( ) 21. Now, show that even a single statement may be listed. Clear your screen. Then press LIST. Figure 58 shows the result.



```
10 PRINT "HP-85 IS MY NAME"
 14551
```

FIGURE 58
Screen after Step 21

Being able to list part of a program while you're writing it is a big help. Take advantage of this power when you write your programs.

( ) 22. You can print as well as display a listing. To print a listing, you press SHIFT + P LST LIST. P LST stands for PRINT LIST.

( ) 23. Print a listing of statement 10. Press SHIFT + P LST LIST and see on the printer the same as you saw earlier on the screen, Figure 58, except the number of "bytes" of memory remaining within the HP-85, 14551, is displayed, not printed.

( ) 24. Your program should have at least four statements. Now write that program, and good luck!

( ) 25. When you're finished, see one way to write the "Name-Game" program. A listing is shown on page H-9, No. 10 in the Supplement's HELP Section.

( ) 26. If you had trouble getting this program finished on your own, rest assured you're not the first one who has had trouble with a program. If you were unable to finish this program, why don't you enter the statements shown on page H-9, No. 10, then press RUN, LIST and P LST (SHIFT + P LST LIST). Remember, enter a statement means type in the statement with the statement number and press END LINE. See Help No. 9 on page H-8 in the Supplement for further assistance.

( ) 27. If you were successful, congratulations!

Summary of Chapter 4

- **HP-85 user memory = 14576 bytes.** At the time of market introduction, a 16K HP-85 had 14576 units, or bytes, of memory available for users.

  Reference: Program "CH4" display.

- **LOAD: A command**

  The LOAD command copies a program from the tape into user memory. Once loaded, the cartridge may be removed.

  Reference: "CH4" display.

- **TROUBLE KILLERS**

  METHOD 1—Gentlest—erases only one line from screen.

  a. Press SHIFT + BACK SPACE.

  b. Press -LINE (press CLEAR -LINE).

  c. Try executing command or entering statement (don't forget to press END LINE in either case). If trouble persists, try Method 2.

  METHOD 2—Easiest and often best.

  a. Clear screen (press SHIFT + CLEAR -LINE).

  b. Try executing command or entering statement. If trouble persists, try Method 3.

  METHOD 3—Erases entire user memory within the HP-85 and erases screen.

  a. Make sure tape drive is OFF.

  b. Turn the HP-85 OFF, then ON.

  c. Load any program you were working with and continue.

  d. Otherwise, start over.

  Reference: Program "CH4" display.

- **Spaces in statements**

  1. Spaces are always preserved in quoted messages by the HP-85.

  2. Spaces are often ignored otherwise. The HP-85 puts in its own spaces when statement is listed.

Reference: Program "CH4" display, also printout, steps 3-10.

- **Scratching memory means clearing memory.** Memory must be scratched before entering statements of a new program. One way to scratch memory is to switch the HP-85 OFF, then ON. Reference: Program "CH4" printout, steps 1 and 2.

- **To add, modify, or delete a statement** to or from a complete or incomplete program currently in user memory:

  1. To add: Use a new statement number, type the statement and press (END LINE).
  2. To modify:
     Either: Get statement on screen, either the original typing or a listing, edit it (while keeping the same statement number), and enter it (press (END LINE)).
     Or: Type a new statement using same statement number, and enter it (press (END LINE)).
  3. To delete: Type statement number only and press (END LINE).

  Reference: Program "CH4" printout, steps 13-29.

- **Statements may be entered in any numerical order.** The HP-85 will list and execute the lowest numbered first, then the next lowest, and so on, ending with the highest numbered statement.

  Reference: Program "CH4" printout, steps 19-24.

- `END`**: A BASIC word**

  A program should end with an `END` statement. It should be the highest numbered statement in the program.

  Reference: Page 4-61, step 9.

- `DISP`**: A BASIC word**

  To display a message, use `DISP` in a statement and enclose the message in quotes. Include the statement in a program, and run the program.

  Reference: Page 4-61, step 10.

- `PRINT`**: A BASIC word**

  To print a message, use `PRINT` in a statement and enclose the message in quotes. Include the statement in a program, and run the program.

  Reference: Page 4-61, step 11.

- **To execute a command or statement:**

  1. Last character position of preceding line must be blank.
  2. Portion of line in front of command or statement must be blank.
  3. Command or statement must be typed correctly.

4. Portion of line following command or statement must be blank.
5. Finally, (END LINE) must be pressed.

Reference: Page 4-61, step 13.

- **Statements vs. commands:**

  1. *Both* are instructions for the HP-85.
  2. *Statements* are parts of programs, they use line (statement) numbers, and are executed when the program is run.
  3. *Commands* are not parts of programs, they do not use line numbers, and they are executed either
     a. When a command key is pressed, like (RUN), or:
     b. When (END LINE) is pressed after a command like LOAD.

  Reference: Page 4-62, step 17.

- **The (LIST) key:**

  Displays listing of complete or incomplete program.

  Reference: Page 4-63, step 21.

- **The (P LST) key:**

  Prints listing of complete or incomplete program. ((SHIFT) + (P LST / LIST)).

  Reference: Page 4-63, step 22.

Review Test for Chapter 4

Both questions and answers are given by program "TEST4" on your BASIC Training cartridge. Load this program by pressing

(LOAD) (") (T) (E) (S) (T) (4) (") (END LINE)

If the HP-85 refuses to accept your LOAD command, review and use your trouble killers. See page 4-64.

When the tape drive light goes out, press (RUN). Then follow the instructions on the screen.

Notes

Chapter 5

Increase Your Control

Preview

In Chapter 5, you will:

- Learn how to start a program at some place other than the beginning.
- Learn how to quickly see any segment of your program instructions.
- Learn how to clear memory without switching OFF, then ON.
- Learn how to put the HP-85 in a "just-switched-on" condition without switching OFF, then ON.
- Get the last word on trouble killers.
- Get more editing practice.
- Learn how to put quotation marks into printed or displayed messages.
- Learn how to record a program on your tape cartridge.
- Learn what numbers you may use to number your program instructions.

( ) 1. Don't forget to take Chapter 4's Review Test (program"TEST4") again if you feel a refresher would be useful. See page 4-66.

( ) 2. HELP MESSAGE:

> IF YOU NEED HELP
> during this chapter, follow these steps:
>
> a. Look for the IF YOU NEED HELP messages in this workbook and in program "CH5."
>
> b. Review the Trouble Killers summarized on page 4-64.
>
> c. Review the editing tools starting on page 1-23.

( ) 3. Here are two similar commands you'll be using soon:

R U N [line number] END LINE

L I S T [line number] END LINE

( ) 4. When you press the RUN key, the program in the HP-85's memory will always start at the lowest numbered line (lowest numbered statement) in the program. There will be times when you will wish to start a program at a different, higher numbered line. Say you wanted to start at line 1000. Simply press the eight keys

R U N 1 0 0 0 END LINE

Since the RUN key is an immediate execution key, pressing the single key RUN gives you no chance to type in a line number. Pressing RUN is equivalent to pressing the four keys R U N END LINE. To begin program execution at a line number different from the lowest numbered line in the program, the three characters RUN must be typed in, followed by the line number, and finally END LINE must be pressed.

( ) 5. If you load a program from tape or enter a program statement by statement (line by line) from the keyboard, and later press the LIST key, a listing of the first part of your program will be displayed on your screen starting with the lowest line number. Let's assume this display shows lines 10 through 100. If LIST is pressed again, the program segment starting with the next line (say 110) will be displayed. Let's assume further that you have a big program in memory, with statements numbered 10, 20, 30, and so on up to 1500. If you wanted to display the listing of the program segment starting with line 1000, you could continue to press LIST, and list adjacent segments of your program, until you finally reached the segment which included line 1000.

A much easier way to do this is to press these 9 keys:

L I S T 1 0 0 0 END LINE

( ) 6. Soon you'll be using program "CH5." As you work with that program, I'll ask you to execute both

R U N [line number] END LINE

and L I S T [line number] END LINE.

( ) 7. Before you load and run program "CH5," I'd like you to learn a new command and a new key function described in steps 8 and 9. See Figure 59 for key locations.

( ) 8. The new command, SCRATCH, is executed by pressing these keys: SHIFT + SCRATCH CONT END LINE.

Pressing these three keys clears; that is, scratches the HP-85's user memory. As you see, SCRATCH is a typing aid like LOAD. Executing the SCRATCH command accomplishes the same thing as turning the HP-85 OFF, then ON, *except:*

a. The screen is *not* cleared.

b. The PRINTALL command is *not* cancelled. (You'll learn later about some other commands that are not cancelled by SCRATCH.)

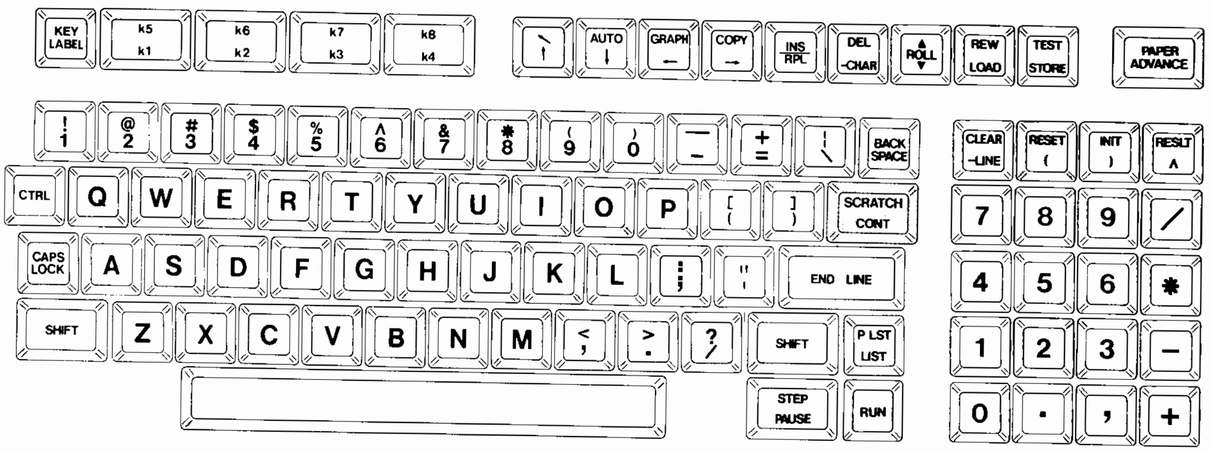


FIGURE 59

Key locations

( ) 9. The new key function is RESET (SHIFT + RESET ( ).

Pressing RESET puts the HP-85 in the same state as when the HP-85 is turned OFF, then ON, *except:* User memory is *not* scratched (not cleared).

RESET clears the screen and cancels PRINTALL (and some other commands you'll learn about later).

( ) 10. Here's a summary of what SCRATCH and RESET do:

SCRATCH COMMAND AND RESET KEY

COMMAND OR KEY	PROGRAM MEMORY	SCREEN	COMMANDS LIKE PRINTALL THAT PUT THE HP-85 INTO A NON-"WAKE-UP" CONDITION
SCRATCH	WIPED CLEAN	NO CHANGE	NO CHANGE
RESET	NO CHANGE	CLEARED	CANCELLED

( ) 11. So, the equivalent to switching the HP-85 OFF, then ON, is to

a. Execute SCRATCH (SHIFT + SCRATCH CONT END LINE), and

b. Press RESET (SHIFT + RESET ( ).

( ) 12. As you work with program "CH5" a little later, you'll execute SCRATCH and press RESET.

( ) 13. Now that you know about the SCRATCH command and the RESET key, it's time to improve the trouble killing keystrokes discussed in Chapter 4. The first two trouble killing methods are unchanged, and the final "if all else fails" method is still turning the HP-85 OFF, then ON.

Before using the ON-OFF switch, however, it's good practice to use the equivalent keystrokes, SCRATCH END LINE RESET. For one thing, pressing either SCRATCH END LINE or RESET while the tape drive is running causes no trouble. The tape drive will continue to run. However, if the HP-85 is turned off, a running tape drive will stop, which could cause the loss of valuable information.

( ) 14. Here is the final set of trouble killers:

METHOD 1—Gentlest—erases only one line from screen.

a. Press SHIFT + BACK SPACE.

b. Press -LINE (press CLEAR -LINE).

c. Try executing command or entering statement (don't forget to press END LINE in either case). If trouble persists, try Method 2.

METHOD 2—Easiest and often best.

a. Clear screen (press SHIFT + CLEAR -LINE).

b. Try executing command or entering statement. If trouble persists, try Method 3.

METHOD 3—Erases memory and clears screen. Does not affect tape drive.

a. Clear memory (press SHIFT + SCRATCH CONT END LINE).

b. Reset computer (press SHIFT + RESET ( ).

c. Load any program you were working with and continue.

d. Otherwise, start over. If trouble persists, try Method 4.

METHOD 4—Erases memory and clears screen. Stops running tape drive.

a. Make sure tape drive is OFF.

b. Turn the HP-85 OFF, then ON.

c. Load any program you were working with and continue.

d. Otherwise, start over.

( ) 15. Now it's time for some more editing practice. I've prepared some program statements that definitely need editing. When executed, these statements print a message on the printer. When you run program "CH5" in step 17, you'll see the sorry state of this message. After you read that message, see step 18 for more instructions.

( ) 16. If your BASIC Training tape cartridge is not inserted, insert it now into the HP-85

( ) 17. Now load and run program "CH5" and read the printed message. Do you remember how to execute LOAD and RUN? Follow these keystrokes:

LOAD " C H 5 " END LINE

When the tape drive light goes out, press RUN and read the printout.

( ) 18. After you complete this step, read steps 19 and 20. Step 21 will ask you to edit.

Now use one of your new commands:

Press L I S T 9 0 END LINE

( ) 19. Let me comment on a feature of this display before you begin editing. Even though the screen is 32 characters wide, the HP-85 will accept a BASIC statement as long as 95 characters. Look at line 100. This statement (including spaces, quotation marks, and the line number, 100) is 41 characters long. So characters 33 through 41 (`EW AWAY."`) appear on the next line.

However, note that the number of characters within the quotation marks in every statement is never more than 32. This means the text that gets printed by each statement will appear on one line. See the printout you got when you pressed RUN in step 17. (I'll introduce you later to `PRINT` and `DISP` statements having more than 32 characters within the quotation marks).

( ) 20. Please read this carefully:

Note: After editing a line, don't forget to press END LINE. Otherwise, your old, unrevised text will come right back to haunt you the next time you list or run your program.

Also, you may press END LINE regardless of where your cursor is on the line. If your cursor is under any character (including spaces and the statement number) in a displayed statement, and if END LINE is pressed, that version of the statement goes into memory.

IF YOU NEED HELP with step 21, see page H-9, No. 11 in the HELP Section for step-by-step editing of line 100.

( ) 21. Use your editing tools to straighten these lines out, and then I'll tell you in step 22 how to list the statements that print the rest of this message. You can then edit those statements as well.

IF YOU NEED HELP with step 22, clear your screen (SHIFT + CLEAR -LINE) and try again.

( ) 22. After I tell you how to list the remaining lines, I'll tell you in step 24 how to run all your corrected Iron Jaw statements. Now list the rest of these statements by pressing L I S T 1 1 0 END LINE.

( ) 23. Now edit lines 125 through 145.

IF YOU NEED HELP with step 24, try clearing your screen (SHIFT + CLEAR -LINE) and repeating step 24 again.

( ) 24. Next, run your corrected program segment by pressing

R U N 8 0 END LINE

( ) 25. Did some of your brilliant corrections mysteriously disappear? Perhaps you forgot to press END LINE after you finished correcting each line. If your results don't satisfy you, why don't you do some re-editing by completing steps 18 through 25 one more time? Repeat these steps as often as you like. Experienced programmers often edit many times before they're happy.

( ) 26. Is there an easy way to print or display quotation marks using PRINT or DISP statements? Program "CH5" has the answer. You do *not* have to load program "CH5" into the HP-85's memory. You already did that in step 17 above. All you need to do is start "CH5" at a line number I'll give you in the next step, then follow the instructions on the screen.

( ) 27. To start program "CH5" at line 280, press the 7 keys:

R U N 2 8 0 END LINE

Summary of Chapter 5

- **To start a program at any line number,** use R U N [line number] END LINE. Reference: Page 5-69, step 4.

- **To list on the screen any program segment,** use L I S T [line number] END LINE. Reference: Page 5-69, step 5.

- SCRATCH: **A command**

  To clear memory, use the SCRATCH command. Press SHIFT + SCRATCH CONT END LINE.

  Reference: Pages 5-69 and 5-70, steps 8 and 10.

- **The** RESET **key:**

  To put the HP-85 in a "just-switched-on" state, while *preserving memory,* press RESET (SHIFT + RESET ( ).

  Reference: Page 5-70, steps 9 and 10.

- **To simulate switching the HP-85 OFF, then ON;** that is, to put the HP-85 in a "just-switched-on" state, including *clearing memory,* execute SCRATCH and press RESET (press SHIFT + SCRATCH CONT END LINE plus SHIFT + RESET ( ). Reference: Page 5-70, steps 10 and 11.

- **TROUBLE KILLERS—final set**

  METHOD 1—Gentlest—erases only one line from screen.

  a. Press SHIFT + BACK SPACE.

b. Press -LINE (press CLEAR -LINE).

c. Try executing command or entering statement (don't forget to press END LINE in either case). If trouble persists, try Method 2.

METHOD 2—Easiest and often best.

a. Clear screen (press SHIFT + CLEAR -LINE).

b. Try executing command or entering statement. If trouble persists, try Method 3.

METHOD 3—Erases memory and clears screen. Does not affect tape drive.

a. Clear memory (press SHIFT + SCRATCH CONT END LINE).

b. Reset computer (press SHIFT + RESET ( ).

c. Load any program you were working with and continue.

d. Otherwise, start over. If trouble persists, try Method 4.

METHOD 4—Erases memory and clears screen. Stops running tape drive.

a. Make sure tape drive is OFF.

b. Turn the HP-85 OFF, then ON.

c. Load any program you were working with and continue.

d. Otherwise, start over.

Reference: Pages 5-70 through 5-71, steps 13 and 14.

- **Use the apostrophe (') to simulate quotation marks** in a printed or displayed message. Reference: Program "CH5," first display; also first printout, steps 3-5.

- **The TEST key**

To test the HP-85's circuits, press TEST (SHIFT + TEST STORE). TEST displays 5 lines on the screen, but the HP-85's memory is unaffected.

Reference: Program "CH5," second printout, steps 8 and 9.

- **CAUTION!** When you execute LOAD with a cartridge in the HP-85, memory is immediately scratched. If you execute LOAD when you wish to execute STORE, the program you intended to store will be lost. So:

DON'T MIX UP THE LOAD AND STORE KEYS!

Reference: Program "CH5," second printout, item 11.

- STORE: **A command**

To record a program on your tape, use the STORE command. Press (STORE) [name of program] (END LINE).

Reference: Program "CH5," second printout, step 12.

- **A stored program name may contain a maximum of 6 characters,** including spaces, but not including quotation marks. Reference: Program "CH5," second printout, step 13.
- **Line numbers from 1 to 9999 inclusive are acceptable.** Reference: Program "CH5," last display.
- **It's good practice to skip numbers between adjacent statements** (10, 20, 30, ... rather than 1, 2, 3 ...). Reference: Program "CH5," last display.

Review Test for Chapter 5

Answer all questions, then see the answers on page 5-76.

1. A program named TIGER uses statements numbered from 10 to 2000. If you wanted to start TIGER at statement number 500, write down every key you would press, including (SHIFT) (if pressed).
2. Suppose the following names of planets are proposed as program names. Which of these names may be used without change to record a program on a tape cartridge?

MERCURY	MARS
VENUS	JUPITER
EARTH	SATURN

3. You have just written a check balancing program, and you attempt to store it on your cartridge with a STORE "CHK BAL" command. When you enter this command, will the HP-85 accept it? If not, why not?
4. What substitute for a quotation mark may be used between the quotation marks of a DISP or PRINT statement?
5. Why is it good practice to skip numbers when numbering adjacent statements?
6. If you wanted to display a listing of the TIGER program starting at line 750, write down every key you would press to get the first group of statements displayed. If you would press (SHIFT), write it down also.
7. What keys, including (SHIFT) (if used) would you press to record the TIGER program on a tape cartridge?
8. What keys would you press to simulate turning the HP-85 OFF, then ON? If you would press (SHIFT), write it down also.

The answers are on the next page.

Answers to Review Test Questions for Chapter 5

1. [R] [U] [N] [5] [0] [0] [END LINE]. Reference: Page 5-69, step 4.

2. VENUS
   EARTH
   MARS
   SATURN

   The remaining two may not be used, since both MERCURY and JUPITER have seven letters. The maximum number of characters a program name may have between the quotation marks, including spaces, is six. Reference: Program "CH5," second printout, step 13.

3. The HP-85 will not accept the command because the name has seven characters, six letters and a space. Reference: Program "CH5," second printout, step 13.

4. Apostrophe ('). Reference: Program "CH5," first display; also first printout, steps 3-5.

5. It makes it easy to insert new statements between two original statements. Reference: Program "CH5," last display.

6. [L] [I] [S] [T] [7] [5] [0] [END LINE]. Reference: Page 5-69, step 5.

7. [STORE] [SHIFT] + [" '] [T] [I] [G] [E] [R] [SHIFT] + [" '] [END LINE]. Reference: Program "CH5," second printout, step 12.

8. [SHIFT] + [SCRATCH CONT] [END LINE] [SHIFT] + [RESET (].

   Pressing [RESET] first followed by executing the `SCRATCH` command would not simulate turning the HP-85 OFF, then ON, since the screen would not be clear. The screen would show: `SCRATCH`.

   To be precise, executing `SCRATCH` and pressing [RESET] would not have exactly the same effect as switching the HP-85 OFF, then ON, since no search for an "Autost" program would occur. Reference: Page 5-70, steps 10 and 11.

Notes

Chapter 6

Write Two Wordy Programs That Figure

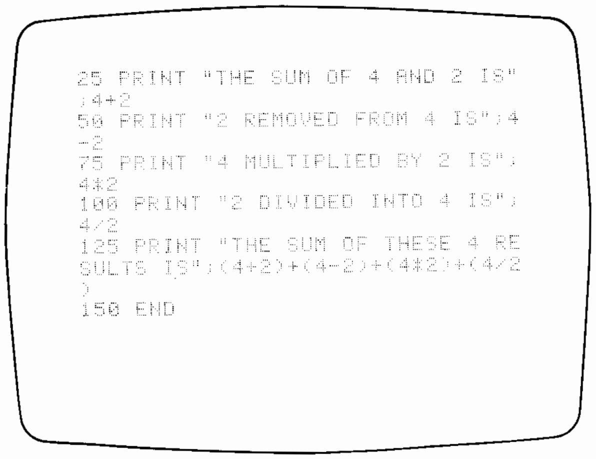
Preview

In Chapter 6, you will:

- Write, enter and run two programs.
- Enter and run two other programs.
- Learn how your program can generate a blank line.
- Learn how one statement can produce words, and also perform math calculations and show the results.
- Learn how a program can produce close spacing and wide spacing between quoted text and numbers.
- Learn how one statement can produce three lines of text and numbers.
- Learn how long an HP-85 statement can be.
- Discover the difference between a displayed listing and a printed listing.

( ) 1. The review test for Chapter 5, page 5-75, would be worthwhile taking again if you've been away from the course for a couple of days.

( ) 2. Now for those powerful `DISP` and `PRINT` statements I mentioned at the end of program "CH5." Can you predict what the following "Calculate" program will print when run?



```
25 PRINT "THE SUM OF 4 AND 2 IS"
;4+2
50 PRINT "2 REMOVED FROM 4 IS";4
-2
75 PRINT "4 MULTIPLIED BY 2 IS";
4*2
100 PRINT "2 DIVIDED INTO 4 IS";
4/2
125 PRINT "THE SUM OF THESE 4 RE
SULTS IS";(4+2)+(4-2)+(4*2)+(4/2
)
150 END
```

IF YOU NEED HELP with step 3, see page H-17, No. 20.

( ) 3. Execute `SCRATCH` and press RESET (unless you just turned the HP-85 on). You're going to enter the "Calculate" program, and you want to be sure the HP-85's memory is clear. Also, you want to be sure `PRINTALL` is not active.

( ) 4. Before entering "Calculate," look at the semicolon circled below in line 25.

```
25 PRINT "THE SUM OF 4 AND 2 IS"
;4+2
```

Look at the listing of "Calculate" in step 2 and notice that a semicolon is also used in a similar way in lines 50, 75, 100 and 125.

( ) 5. You'll learn later in this chapter why this semicolon is used to separate the words and mathematical expressions in these statements.

( ) 6. For now, just remember that it's important to use semicolons in `PRINT` and `DISP` statements to separate quoted words from mathematical expressions.

( ) 7. Remember, when entering a statement longer than 32 characters, keep typing until you reach the end of the statement, then press (END LINE). When entering such a greater-than-32 character statement, do *not* press (END LINE) immediately after you type the 32nd character. At that point, the HP-85 will automatically move your cursor to the beginning of the next line, inviting you to continue typing.

IF YOU NEED HELP with step 8, see page H-17, No. 21.

( ) 8. OK. Now enter the "Calculate" program, statement by statement. Remember (END LINE)!

( ) 9. Run your "Calculate" program, and check its output against the correct output shown in Figure 60.

```
THE SUM OF 4 AND 2 IS 6
2 REMOVED FROM 4 IS 2
4 MULTIPLIED BY 2 IS 8
2 DIVIDED INTO 4 IS 2
THE SUM OF THESE 4 RESULTS IS
 18
```

FIGURE 60

Output of "Calculate"

( ) 10. Read steps 11 and 12, then press keys when you reach step 13.

( ) 11. It's more fun to write than to only copy, so try your hand at the "Seven Come Eleven" program. Step 12 describes the program and step 13 asks you to write it.

( ) 12. When your program is run, the screen should display

```
SEVEN RAISED TO THE
ELEVENTH POWER IS [number]
```

Your program should calculate `7^11`, which is seven raised to the eleventh power, and the answer should replace [number].

IF YOU NEED HELP with steps 13 and 14, see page H-17, No. 22.

( ) 13. Write and enter your "Seven Come Eleven" program. Remember to scratch memory before you start. And remember the semicolon. Also END LINE.

( ) 14. Run your program, then print its listing.

( ) 15. To see the output of "Seven Come Eleven," turn to page H-18, No. 23.

( ) 16. Perhaps your version of "Seven Come Eleven" has an error or two. If so, list your program on the screen and edit the errors out of it. Remember END LINE. Then go back to step 14.

( ) 17. Now read steps 18, 19 and 20. Your fingers may relax until step 21.

( ) 18. Since your programmer's hat is now firmly fixed to your skull, try composing the "Binary Brain" program.

( ) 19. "Binary Brain" McCrunch, programming supervisor, and "Flying Fingers" Flanagan, chief calculator key tester for a local calculator concern, have a continuing contest involving the serial numbers on their paychecks. It works this way: The five digits of each serial number are multiplied together, and the one whose paycheck gives the largest number gets a free lunch from the other. To illustrate this contest, let's take a look at how last week's contest worked out. Binary Brain's serial number was 12345, while Flying Fingers' check showed 22245. Multiplying McCrunch's five digits together gave 1 * 2 * 3 * 4 * 5, or 120. Flanagan's digits, when multiplied together (2 * 2 * 2 * 4 * 5) gave 160. So Flying Fingers got a free lunch.

This week, Binary Brain's number is 62639 and Flying Fingers has 89741. At this very moment, they are in a heated discussion about whose multiplied number is greater. Each insists he gets the free lunch.

Your mission, should you choose to accept it, is to write a program that, when run, will show who is right. Your program should calculate these numbers. No fair calculating these numbers on your handy HP-85 and putting the answers into your program.

( ) 20. Your program's printed output should look like the following, with [number] replaced by the proper number in each case.

```
BINARY BRAIN'S NUMBER
IS [number]
FLYING FINGERS' NUMBER
IS [number]
```

IF YOU NEED HELP with steps 21 through 23, see page H-18, No. 24.

( ) 21. Write and enter your "Binary Brain" program into the HP-85. Remember to scratch, remember END LINE, and remember the semicolon.

( ) 22. Run your program.

( ) 23. Get a listing on the printer.

( ) 24. There is no ''best'' way to write a program. This is especially true when programs are longer and more complex. So each of the programs I'll show you throughout this course represents only one acceptable way to do the job.

Who is the final judge of program quality? You are.

( ) 25. To see one acceptable way to write ''Binary Brain,'' see page H-18, No. 25.

( ) 26. If you're unsatisfied with your version of ''Binary Brain,'' list it on the screen, edit your errors away, and run it. Then start at step 25.

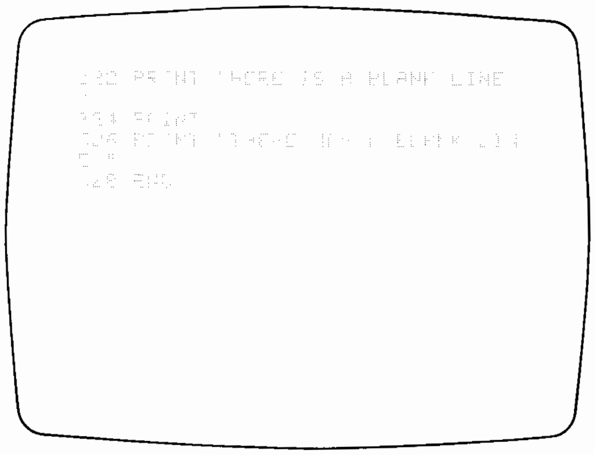
( ) 27. The next program you'll enter will demonstrate two new BASIC weapons. One allows you to ''print'' or ''display'' a blank line, and the other allows you to control spacing within one printed or displayed line using commas and semicolons. (I told you I'd get back to semicolons).

( ) 28. First—how do you instruct a program to put a blank line on the printer? Simple. This statement

(line number) `PRINT`

''prints'' a blank line when it's executed.

For example, when this program is run



```
10 PRINT "HERE IS A BLANK LINE
:"
20 PRINT
30 PRINT "THERE WAS A BLANK LIN
E."
40 END
```

it prints this:

```
HERE IS A BLANK LINE:

THERE WAS A BLANK LINE.
```

( ) 29. To get a blank line "displayed" on the screen, use this statement:

(line number) `DISP`

( ) 30. I'd like you to discover for yourself what commas and semicolons do. The following "Semicolon, Comma" program will demonstrate the action of each.§

IF YOU NEED HELP with steps 31 and 32, see page H-19, No. 26.

( ) 31. When you enter this program, enter the extra spaces (▲) where indicated using the space bar. You'll see why they're there when you run your program. (Remember: press [END LINE] after every statement—not after every 32 character line on the screen).

```
10▲PRINT▲"A▲SEMICOLON▲(;)▲GIVES▲
CLOSE▲▲▲▲▲SPACING,▲WHILE▲A▲COMMA
▲(,)▲▲▲▲▲▲SPREADS▲THINGS▲OUT."[END LINE]
20▲PRINT[END LINE]
30▲PRINT▲"A▲SEMICOLON▲DOES▲THIS:
"[END LINE]
```
(Note that the closing " folds onto the next 32 character line).
```
40▲PRINT▲"5";"10";"15"[END LINE]
50▲PRINT▲"WHILE▲A▲COMMA▲DOES▲THI
S:"[END LINE]
60▲PRINT▲"5","10","15"[END LINE]
70▲END[END LINE]
```

( ) 32. After you've entered your "Semicolon, Comma" program into the HP-85's memory:

a. Run it.

b. Clear your screen.

c. List your program on the screen.

d. Press [PAPER ADV] 3 or 4 times to generate some blank paper.

e. Copy your displayed listing.

f. List your program on the printer.

g. Press [PAPER ADV] to raise your printing above the tear off bar, and tear it off. Don't lose it. You'll be using it a little later.

( ) 33. To see how your version of "Semicolon, Comma" compares with mine, see the program output in Figure 61. If your output is different, get your editing tools, list your program on the screen, and fix it. Remember to press [END LINE] after you finish editing a statement, while your cursor is still on one of the lines of the statement. Pressing [END LINE] gets the revised version of that statement into memory, where it replaces the old version. Your revised listing should look like the displayed listing shown in Figure 61.

§ For more details on the exact spacing produced by commas and semicolons, see page 84 in your Owner's Manual.

( ) 34. Look at the printout you tore off the printer in step 32. My printout is shown in Figure 61. Notice the different format used for a displayed listing compared to a printed listing. In the displayed listing, the second and third lines of statement 10 are against the left margin—see A. In the printed listing, the extra lines for line 10 are indented to make the statement or line number 10 stand out—see B. Notice how the indentation forces the printed listing of line 10 to occupy four lines of printing compared to three for the displayed listing.

"Semicolon, Comma" program output

```
A SEMICOLON (;) GIVES CLOSE
SPACING, WHILE A COMMA (,)
SPREADS THINGS OUT.

A SEMICOLON DOES THIS:
51015
WHILE A COMMA DOES THIS:
5                     10
15
```

Displayed listing

Ⓒ Ⓐ Ⓓ

```
10 PRINT "A SEMICOLON (;) GIVES
CLOSE      SPACING, WHILE A COMMA
 (,)       SPREADS THINGS OUT."
20 PRINT
30 PRINT "A SEMICOLON DOES THIS:
"
40 PRINT "5";"10";"15"
50 PRINT "WHILE A COMMA DOES THI
S:"
60 PRINT "5","10","15"
70 END
 14367
```

Printed listing

Ⓑ

```
10 PRINT "A SEMICOLON (;) GIVES
     CLOSE      SPACING, WHILE A
   COMMA (,)       SPREADS THING
   S OUT."
20 PRINT
30 PRINT "A SEMICOLON DOES THIS
   :"
40 PRINT "5";"10";"15"
50 PRINT "WHILE A COMMA DOES TH
   IS:"
60 PRINT "5","10","15"
70 END
```

FIGURE 61

Printout generated by Step 32

( ) 35. Also notice the awkward use of the word "line" in step 34. Get used to it. Since "line," in BASIC, refers to a numbered statement, which can occupy more than one line of display or printing, learn to live with the dual use of "line."

( ) 36. Let me point out a useful feature of the displayed listing. See how C points to the first character of each of the three lines printed when statement 10 is executed. Statement 10 printed `A` for the first character of line one; it printed the `S` of `SPACING` for the first character of line two; and it printed the `S` of `SPREADS` for the first character of line three. When you first typed statement 10, you typed enough spaces (5) after `CLOSE` to put the `S` of `SPACING` directly under the `A`. Then, after you typed `(,)`, you typed in 6 spaces to put the `S` of `SPREADS` directly under the `S` of `SPACING`. In the future, when you type a multi-line statement, just press the space bar or (_) to put your cursor under the first character of the preceding line and start typing your next line.

( ) 37. Another thing about line 10. The final character, which is the closing quote mark—see D—is the 94th character of line 10. The HP-85 can handle only one more character in one "line." Remember:

The Maximum HP-85
"Line" is 95 Characters

( ) 38. The little "Semicolon, Comma" program had a lot to teach as shown in the Summary below. I hope you entered it successfully. Don't fret if you had to try a few times before entering the program correctly. It's not a trivial program, even though it's fairly short.

Summary of Chapter 6

- **One `PRINT` or `DISP` statement can include both quoted text and mathematical expressions,** separated by a semicolon or comma. When the statement is executed, the quoted material is shown unchanged, while the math expression is evaluated and the result is shown. Reference: Pages 78 and 6-79, steps 2-9.

- **To have a program generate a blank line,** use `DISP` or `PRINT` alone, like:

```
25 DISP
75 PRINT
```

Reference: Pages 6-81 and 6-82, steps 28 and 29.

- **`PRINT` and `DISP` statements can control spacing.** A semicolon (;) between quoted characters and numbers gives close spacing, while a comma (,) gives wide spacing.

  `;` close
  `,` wide

Reference: Pages 6-82 and 6-83, steps 30-34.

- **A `PRINT` or `DISP` statement can generate more than one line of text or text and numbers.** Reference: Pages 6-82 through 6-84, steps 30-36.
- **A displayed listing shows no indentation.** Reference: Page 6-83, step 34.
- **A printed listing shows indentation for all but the first line of a multi-line statement.** Reference: Page 6-83, step 34.
- **95 is the maximum number of characters an HP-85 statement may have.** Reference: Page 6-84, step 37.

Review Test for Chapter 6

Answer both questions, then see the answers on page 6-86.

1. The question is asked in the listing.

```
10 PRINT "THIS IS QUESTION 1."
20 PRINT
30 PRINT "I HOPE YOU GET IT RIG
   HT."
40 PRINT
50 PRINT "HOW MANY BLANK LINES
   WILL THIS  PROGRAM GENERATE?
   "
60 END
```

2. Match the program with the output.

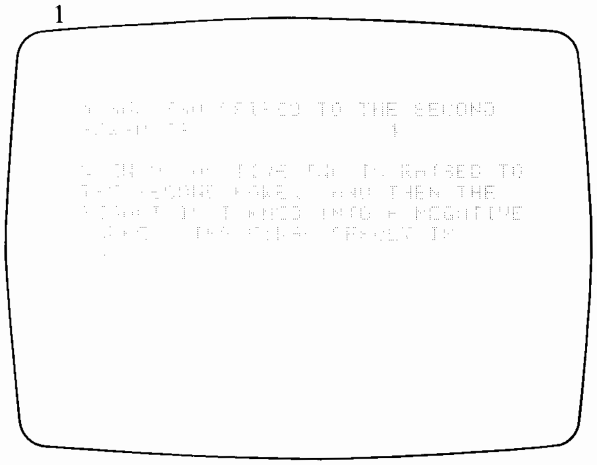
a

```
 5 DISP
10 DISP "MINUS TWO RAISED TO TH
   E SECOND  POWER IS",(-2)^2
15 DISP
20 DISP "WHEN A POSITIVE TWO IS
    RAISED TOTHE SECOND POWER,
   AND THEN THE"
30 DISP "RESULT IS TURNED INTO
   A NEGATIVENUMBER, THE FINAL
   RESULT IS ",-2^2
40 END
```

b

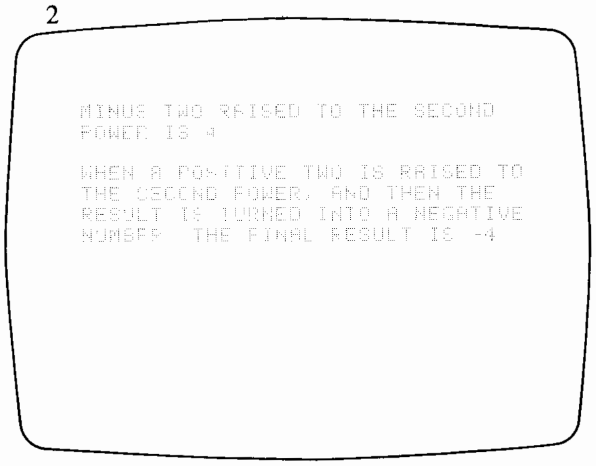
```
 5 DISP
10 DISP "MINUS TWO RAISED TO TH
   E SECOND  POWER IS";(-2)^2
15 DISP
20 DISP "WHEN A POSITIVE TWO IS
    RAISED TOTHE SECOND POWER,
   AND THEN THE"
30 DISP "RESULT IS TURNED INTO
   A NEGATIVENUMBER, THE FINAL
   RESULT IS ";-2^2
40 END
```

1



```
MINUS TWO RAISED TO THE SECOND
POWER IS                 4

WHEN A POSITIVE TWO IS RAISED TO
THE SECOND POWER, AND THEN THE
RESULT IS TURNED INTO A NEGATIVE
 NUMBER, THE FINAL RESULT IS
[illegible]
```

2



```
MINUS TWO RAISED TO THE SECOND
POWER IS 4

WHEN A POSITIVE TWO IS RAISED TO
THE SECOND POWER, AND THEN THE
RESULT IS TURNED INTO A NEGATIVE
NUMBER  THE FINAL RESULT IS -4
```

Answers to Review Test Questions for Chapter 6

1. Two.

   Reference: Page 6-81, step 28.

2. Program A gives output 1.

   Program B gives output 2.

   Reference: Pages 6-82 and 6-83, steps 30-34.

Notes

Chapter 7

Put Numbers Into Your Program While It's Standing Still

Preview

In Chapter 7, you will:

- Learn an important way to put numbers into your programs.
- Learn how a letter can represent a number or value in your program.
- Learn how the same letter can represent a variety of different numbers at different times, but never more than one number at a time.
- Learn about warning messages.
- Write a program to show off your new skills.

( ) 1. If your recollection of Chapter 6 has faded a bit since you finished it, why not read the Summary, page 6-84, and take the Review Test again, page 6-85?

While I won't make this ''review-the-last-chapter'' suggestion again during this course, keep the idea in mind. The better you know the material you've covered, the more fun you'll have with the new ideas I'll be giving you.

( ) 2. I want you to meet a noisy friend of mine, Mr. Loudmouth:

You'll be seeing a lot of old Loudmouth and his more agreeable companions in this chapter.

( ) 3. At the moment, he's trying to balance his checkbook, a task many of us have struggled with from time to time. Let's watch.

( ) 4. He knows his beginning balance, and he knows his deposit. He is also smart enough to know his new balance is given by his old balance plus his deposit. He thinks of it this way:

Let my New Balance = my Old Balance + my Deposit

( ) 5. Loudmouth has done this job a few times, so he drops off a couple of words:

Let Balance = Balance + Deposit

or LET B = B + D

( ) 6. That looks a little strange at first glance, but Loudmouth knows the B on the left represents his new balance, and the B on the right stands for his old balance:

$$\text{LET}\quad \underset{\substack{\uparrow\\ \text{new}\\ \text{balance}}}{B} \;=\; \underset{\substack{\uparrow\\ \text{old}\\ \text{balance}}}{B} \;+\; \underset{\substack{\uparrow\\ \text{deposit}}}{D}$$

( ) 7. If you feel comfortable with B = B + D, you have just floated gracefully over a major hurdle that has tripped up a large number of beginning programmers.

( ) 8. If this concept whizzed past your head before you were able to grab it, you'll have plenty of chances to hold it firmly with both hands before you finish Chapter 7.

( ) 9. Let's leave Mr. L and get serious. (For those of you concerned about his financial condition, he balanced out at $13.67). Chapter 7 continues on the tape cartridge, where I'll present this B = B + D concept in a more formal manner.

IF YOU NEED HELP with step 10, see page H-19, No. 27.

( ) 10. Insert your BASIC Training cartridge, execute `LOAD"CH7"` and `RUN`. Watch the screen for an addition to your BASIC vocabulary.

( ) 11. Welcome back! When you ran this "A + 2" program:

```
 25 LET A=1
 50 LET A=A+2
 75 PRINT "A=";A
100 END
```

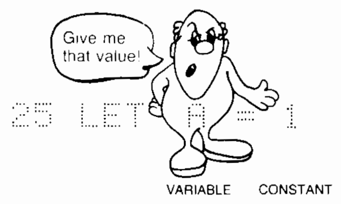
you should have seen this printed:

```
A= 3
```

( ) 12. Mr. Loudmouth is going to help me explain the workings of the "A + 2" program. Before we get into it, remember one thing about Mr. L. He has room in his brain for only one number at a time. If he gets a new number, the old one is forgotten. Don't ask him what his old checkbook balance was. He forgot that completely when he received his new balance.

( ) 13. Now, let's look at the "A + 2" program in detail.

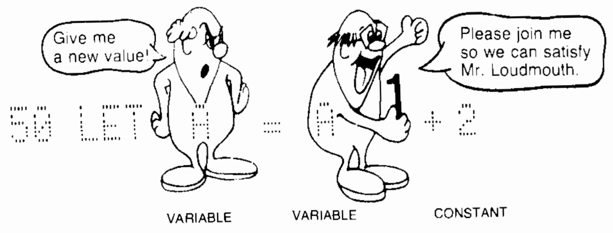
( ) 14.



```
25 LET A = 1
```

Here the variable A is assigned the value 1. The number 1 is called a constant. This is how a number may be put into your program.

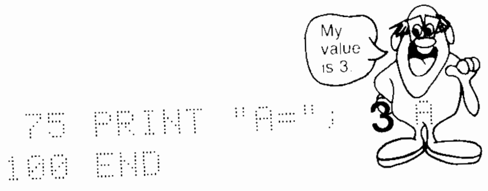
( ) 15.



```
50 LET A = A + 2
```

Now the variable A is assigned the OLD value of A plus the constant 2. The old value of A is 1 from statement 25. So now the variable A *has the new value 3.*

( ) 16.



```
75 PRINT "A=";A
100 END
```

Statement 75 prints the characters between the quotation marks A= followed by the present value of the variable A, which is 3. Therefore, statement 75 prints: A= 3.

( ) 17. Before you study the assignment statement further, take a closer look at variables and variable names.

( ) 18. **Variable:** In BASIC, the word "variable" means the same thing as it does in algebra. It's the name of something that can take one value or a succession of values in a problem or program, one value at a time. For instance:

Thing	Possible Variable Name	Possible Values
A girl's height in centimeters	H	160,170
Area of a circle in square centimeters	A	4.32, 17.3
Price of gasoline in dollars	P	1.27, 2.31

( ) 19. **Variable Name:** In addition to H, A and P above, A in the "A + 2" program is a variable name. So are S and D in your "MONEY" program. We called any square on the chess board S, and the number of silver dollars on a square D. The HP-85 offers you 286 variable names: A—Z plus A0—A9, B0—B9, C0—C9, etc. Examples are C, W7, X3, D0, M. A valid variable name is any letter alone or any letter followed by any single numeral.

( ) 20. To help you recognize a valid variable name, I've prepared another quiz that uses the BASIC Training tape cartridge. Let me tell you how the quiz works before you get into it. The screen will show 20 collections of letters and numbers, one after the other. Some are valid variable names. Your job will be to identify the good ones. When you start the quiz, the screen will show 1. AB?. If you believe AB is a valid variable name, you would enter T (press T END LINE). If you believe AB is not a valid variable name, you would enter F. If you're wrong, the screen will explain why the other answer is correct. Whether right or wrong, you will then be asked to enter your answer for the next question. At the end of the quiz, you'll receive your score. Good luck.

IF YOU NEED HELP with step 21, see page H-20, No. 29

( ) 21. This Variable Names Quiz is in program "CH7." If this program is still in the HP-85's memory, simply execute RUN2000. Otherwise, execute LOAD"CH7", then RUN2000.
Don't forget to press END LINE after you type either T or F.

( ) 22. I hope you aced your quiz. Now for some important truths about the assignment statement:

ASSIGNMENT STATEMENT
VITAL RULES AND REGULATIONS

(Courage! I'll give you some program examples later to help you understand these.)

- When L, or any variable, is on the LEFT of the = symbol:

  1. The variable always stands alone.

     NOT ALLOWED! ⟶ `10 LET L*2=4`

     OK ⟶ `10 LET L=4/2`

  2. The variable on the LEFT always gets a new value, equal to the number on the right.

     `10 LET L=4/2` means L is now 2.

     ★ ★ ★

     `10 LET A=1` means A is now 1.

     $\downarrow$

     `20 LET A=A+2` means A is now 1+2 or 3.

     ★ ★ ★

     `10 LET E=5` means E is now 5.

     `20 LET E=8` means E is now 8.

     E's earlier value, 5, is gone.

- When A, or any variable, is on the RIGHT of the = symbol, it always has an old value, which is used to get a single number on the RIGHT of the = symbol.

```
10 LET A=1
          ↓
20 LET A=A+2
```

This variable, A, has an old value, 1 (from statement 10), which is used to get a single number, 3, on the right of the = symbol.

- If the SAME variable DOES appear on both sides of the = symbol, its OLD value is LOST.

```
10 LET A=1
          ↓
20 LET A=A+2
```

Since A DOES appear on both sides of the = symbol, its OLD value, 1, is LOST. This OLD value is replaced by a NEW value, 3.

- If a variable appears ONLY on the RIGHT of the = symbol, its OLD value is KEPT.

```
10 LET A=1

20 LET B=A+2
```

Since A appears ONLY on the RIGHT of the = symbol, its OLD value, 1, is KEPT. That is, the value of A is still 1.

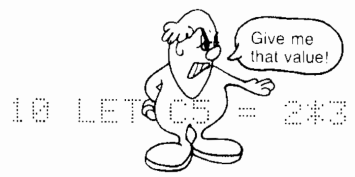
( ) 23. Let's look at some more programs using assignment statements.

( ) 24. Program 1:

```
10 LET C5=2*3
20 LET X=C5
30 DISP "X =";X
40 DISP "C5 =";C5
50 END
```

What's happening?

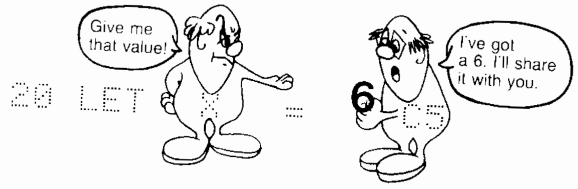
( ) 25.



```
10 LET C5 = 2*3
```

The variable C5 is assigned the value 2*3, which is 6. This statement first calculates 2*3, then does the assignment.

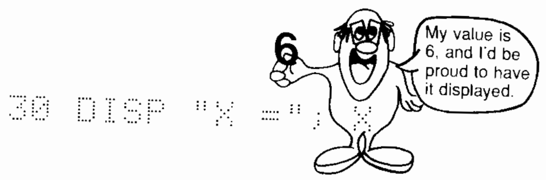
( ) 26.



```
20 LET X = C5
```

Since C5 appears only on the right, it KEEPS its value. At the same time, it SHARES its value 6, with the variable X.

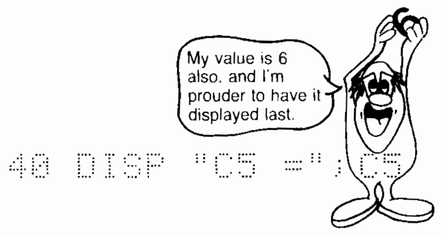
( ) 27.



```
30 DISP "X =";X
```

Now the message X = 6 is displayed on the screen. The variable X got its value, 6, in statement 20.

( ) 28.


6
My value is 6 also, and I'm prouder to have it displayed last.

```
40 DISP "C5 =";C5
```

Next, the message C5 = 6 is displayed. The variable C5 was assigned its value in statement 10, and C5 kept its value in statement 20.

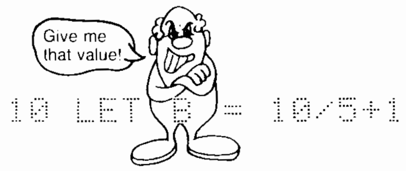
( ) 29. SUGGESTION: To understand an assignment statement more easily, it often helps to read it from right to left. You first determine the number on the right, and then assign that number to the variable on the left of the = symbol.

( ) 30. Program 2:

```
10 LET B=10/5+1
20 LET C=B*2
30 DISP "C DIVIDED BY B IS";C/B
40 END
```

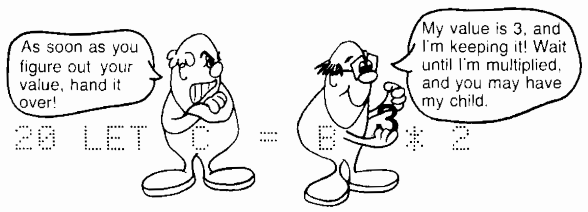
Let's take a closer look.

( ) 31.



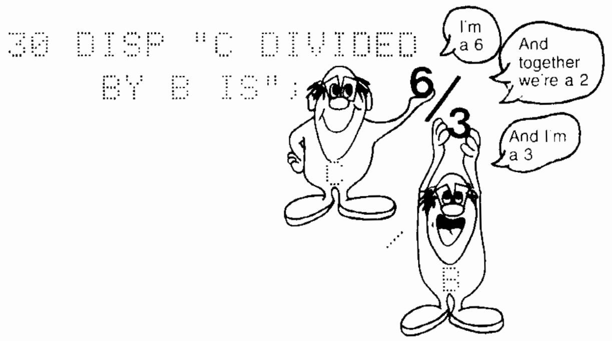
The calculation is performed first, producing the number 3. This number is then assigned to the variable B.

( ) 32.


As soon as you figure out your value, hand it over!
My value is 3, and I'm keeping it! Wait until I'm multiplied, and you may have my child.
20 LET C = B * 2
3

Again, the calculation is performed first, then the resulting value, 6, is assigned to the variable C. Note that the variable B KEEPS its value.

( ) 33.



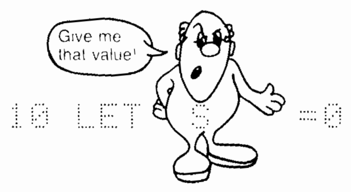
This display statement includes a calculation. As in all PRINT and DISP statements, these variables, C and B, KEEP their values.

( ) 34. Program 3:

```
10 LET S=0
20 LET A=4
30 LET S=S+A
40 DISP "A =";A
50 DISP "S =";S
60 END
```

This time, try to predict what happens when each statement is executed before I tell you.

( ) 35.



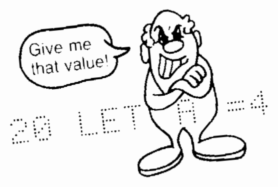
```
10 LET S =0
```

The variable S is assigned the value 0. A variable cannot be used in a program unless it is first given some value—unless it is **initialized**. Zero is a popular value used to initialize variables. If your program does not initialize a variable, the HP-85 will initialize it to zero, give you a WARNING MESSAGE:

```
Warning 7 on line 30 : NULL DATA
```

and continue the program. Note that the message tells you where the variable is used. NULL DATA means "uninitialized numeric variable." I'll tell you more about warning messages later in this chapter.

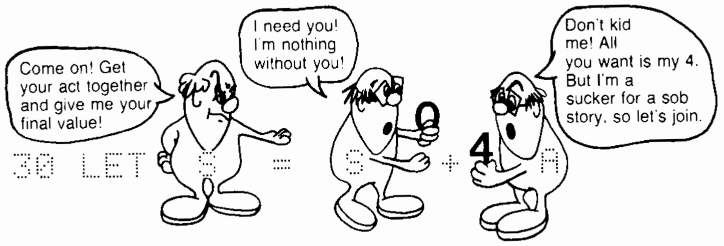
( ) 36.



```
20 LET A =4
```

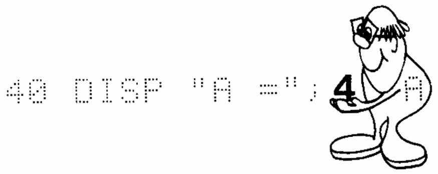
Here, the variable A is assigned the value 4.

( ) 37.


Come on! Get your act together and give me your final value!
I need you! I'm nothing without you!
Don't kid me! All you want is my 4. But I'm a sucker for a sob story, so let's join.
30 LET S = S + A
0
4

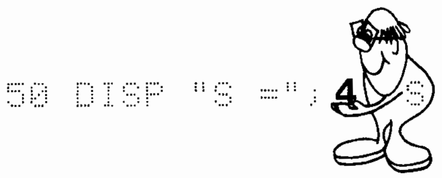
The variable S appears on both sides of the = symbol, so its old value, 0, is lost. However, A appears only on the right, so its value, 4 is kept. The calculation, S+A or 0 + 4, is performed, and then the single number, 4, is assigned to the variable S.

( ) 38.



The message: A = 4 is displayed. A's value was assigned in statement 20 and kept in statement 30.

( ) 39.



Statement 50 displays the message: S = 4. The variable S was assigned value 0 in statement 10, but statement 30 changed its value from 0 to 4.

( ) 40. There are two general ways variable values may be printed or displayed:

PRINTING OR DISPLAYING VARIABLE VALUES

With Text You've seen these statements already.	**Without Text** Here are the same statements rewritten.
`75 PRINT "A=";A` prints: `A= 3` Reference: Page 7-89	`75 PRINT A` prints: `3`
`30 DISP "X =";X` displays: `X = 6` Reference: Page 7-93	`30 DISP X` displays: `6`
`40 DISP "C5 =";C5` displays: `C5 = 6` Reference: Page 7-93	`40 DISP C5` displays: `6`
`30 DISP "C DIVIDED BY B IS";C/B` displays: `C DIVIDED BY B IS 2` Reference: Page 7-95	`30 DISP C/B` displays: `2`

( ) 41. Let's take a look at one more program before you learn more about warning messages and write another program.

"B + X" Program

```
10 LET X=5
20 LET B=7+2
30 LET D4=B+X
40 PRINT "D4 =";D4
50 END
```

What would statement 40 print?

I'll give you a statement by statement analysis so you can check your answer.

( ) 42. `10 LET X=5` — The value 5 is assigned to the variable X. So now X equals 5.

( ) 43. `20 LET B=7+2` — The sum of 7 and 2 is assigned to the variable B. Now B equals 9.

( ) 44. `30 LET D4=B+X`

The sum of the old values for B and X is assigned to the variable D4.
From statement 10, X=5
From statement 20, B= 9
So statement 30 makes D4= 14

( ) 45. `40 PRINT "D4 =";D4`

The output from this program is: D4 = 14.

If you figured out by yourself what statement 40 would print, GREAT! If not, you might read through Chapter 7 again to get these critical ideas well in mind. The assignment statement is one of the foundations of the BASIC language.

( ) 46. You'll soon be writing another program, but first let me finish my remarks about warning messages. These are a special class of error messages. A beep is heard, the message is displayed, and the program continues to run. So a warning message is even friendlier than the normal error message, which stops the program. A warning message reports that the HP-85 has assumed a value for a variable or for a calculation to prevent a program halt. The value assumed and the line number where the trouble occurred are both displayed, as well as a description of the trouble.

( ) 47. Error and warning messages are sometimes called diagnostics because they help you diagnose your trouble. In general, they have these meanings:

ERROR MESSAGE: ''Sorry, I don't understand you, so I've come to a screeching halt. Please try again. I'll do my best to obey.''

WARNING MESSAGE: ''I don't completely understand you, but I'm assuming you mean [one of several things, depending on the message]. I'm going ahead using my assumption, but I'm letting you know so you can stop me if I'm wrong.''

Appendix E, starting on page 303 in your Owner's Manual, lists all error messages. The first eight are the warning messages. The ''default'' value is the value assumed by the HP-85.

( ) 48. Now about that program. Its name is ''23 Skidoo.'' (''Skidoo'' has nothing to do with the program, but ''23'' just doesn't do it by itself).

Try your hand at writing, entering into the HP-85, running and listing on the printer a program that does the following:

a. Assigns the value 5 to a variable.
b. Multiplies that variable by 8.
c. Divides the result by 2.
d. Adds 3 to the new result.
e. Assigns this final result to a second variable.
f. Prints the name and value of the second variable.

( ) 49. To see my version of "23 Skidoo," see page H-20, No. 30. Don't worry if your program is different from mine. If your program prints the correct output, it's OK. Of course, you can use any variable names you wish.

( ) 50. If you were unable to get your "23 Skidoo" program to print the correct value for the second variable, do not despair, you're not alone. Before moving on, however, study my version to make sure you understand how to write such a program. As the course proceeds, your programming opportunities will become more challenging, so it's worthwhile to get these early programs well in mind.

Summary of Chapter 7

- `LET`: **A BASIC word**

  `LET` identifies an assignment statement.

  Reference: Program "CH7" display.

- **Variable:** The name of something that can take one value or a succession of values in a program, one value at a time. Reference: Page 7-91, step 18.

- **Variable name:** Any letter or any letter followed by any single numeral. Examples: `C`, `M5`, `P`, `B7`, `A`. Reference: Page 7-91, step 19.

- **Assignment Statement:** Assigns a value to a variable. The single variable on the left of the = symbol is assigned the single number represented by everything to the right of the = symbol. If the same variable name appears on each side of the = symbol, its old value is lost. In statement `50` below, `A`'s old value is replaced by a new one. If a variable name appears only on the right (`C` below), its value is shared to arrive at a new value for `A`, but `C` retains its previous value.

```
50 LET A=A/2+C*5
```

  Reference: Pages 7-91 through 7-93, step 22.

- To **initialize** a variable means to assign the variable the first, or initial value it has in the program. Before a variable can be used in a program, it must be initialized. Reference: Page 7-96, step 35.

- **Warning message:** In certain error situations, the HP-85 will assume a value for a variable or for the result of a calculation, and continue program execution. A beep is heard, the number of the first executed statement using this variable is displayed, and program execution continues.

  Reference: Page 7-96 step 35; also page 7-99, step 47.

Review Test for Chapter 7

This test is on your BASIC Training cartridge. Your instructions will be printed by the program. To start the test, load "TEST 7" and run it. Remember how to do this? Here's how:

With your cartridge inserted, press LOAD " T E S T 7 " END LINE.

When the tape drive stops, press RUN.

May you get 100%.

Review Test for Chapters 1-7

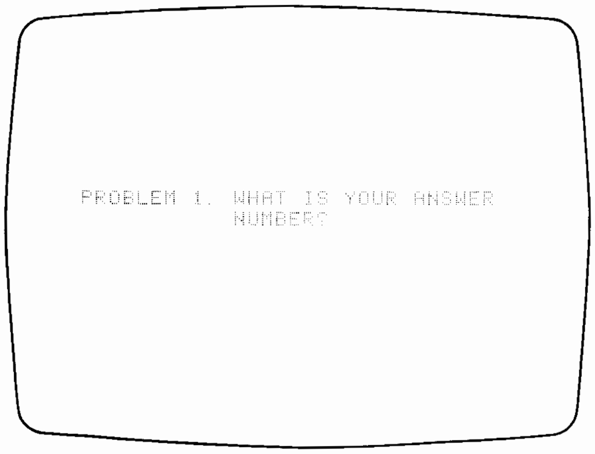
( ) a. This test uses both this workbook and the HP-85. The workbook gives you 17 problems, each with one blank. Responding to your orders, the HP-85 will print an answer list for you. Your job, for each problem, will be to choose an item from the answer list that makes the problem a true statement, and then enter the *number* of that item into the HP-85.

Please, do not type the answers into the HP-85, just read them. All the HP-85 wants is your answer ***numbers.***

If you enter the correct answer number, you'll be suitably praised. If you enter the wrong answer number, or just press END LINE, you'll see on the printer the correct answer and an explanation. In either case, you will automatically be asked to enter your answer number for the next problem.

( ) b. To give you a better idea of how this test works, I'll walk you through the first two problems. Step c is for reading only. Keep your fingers off the keys until step d.

( ) c. I will ask you soon to load and run a program. When you do, your answer list will be printed automatically, and you will see this message on the screen:



```
PROBLEM 1. WHAT IS YOUR ANSWER
NUMBER?
```

IF YOU NEED HELP with step d, see page H-20, No. 31

( ) d. OK. Now insert your BASIC Training cartridge, then execute `LOAD"REVIEW"` and `RUN`.

Here's problem 1:

1. When this statement is executed,

```
10 PRINT "DOG"
```

the printout is ________ .

( ) e. Since this is an example, I'll give it to you free. The answer is not `TSETSE FLY`, as you may have thought, but `DOG`. The printed answer list shows that the answer number for `DOG` is `4`.

IF YOU NEED HELP with step f, see page H-20, No. 32.

( ) f. Enter the answer number `4`, get your reward, and see the HP-85 ask you for your answer number for problem 2.

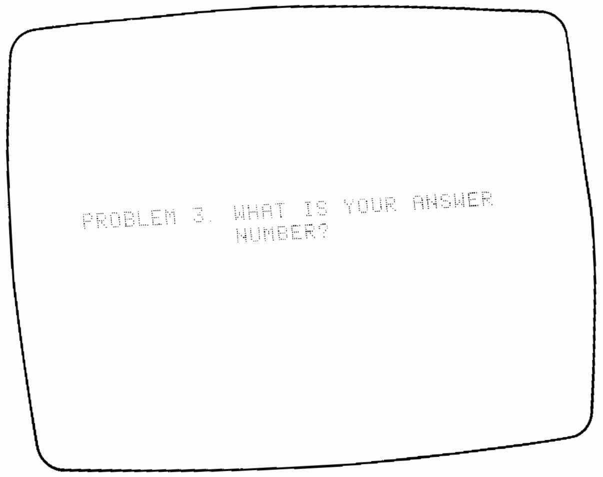
Here is problem 2:

2. To clear your screen, press the ____________ key(s).

( ) g. You know the answer to this one is SHIFT + CLEAR -LINE, so find

```
SHIFT+CLEAR/-LINE - -  30
```

on the printed answer list. Enter `30`, get your pat on the head, and then see:



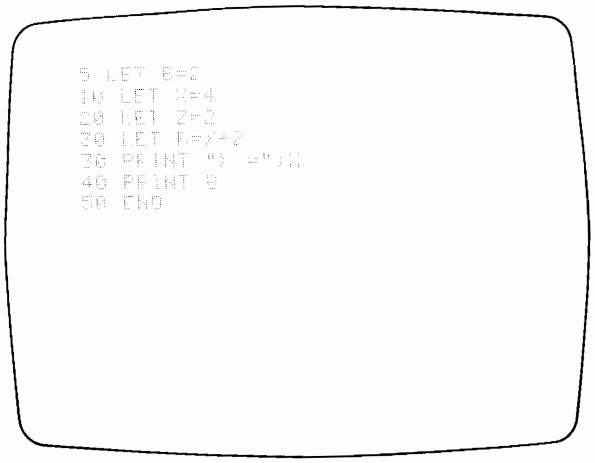
```
PROBLEM 3. WHAT IS YOUR ANSWER
           NUMBER?
```

This starts the real test. Read problem 3 in step h, choose an item from the answer list that puts the right word in the blank, find its answer number in the right hand column, and enter this number. Then continue, problem by problem. May success be yours!

( ) h. Here are the problems that count towards your score:

Note: If you enter an incorrect answer number or you press only (END LINE), I'll give you the correct answer number and an explanation.

3. You type the following into your HP-85 and press (END LINE) after each line.



```
5 LET B=2
10 LET X=4
20 LET Z=2
30 LET B=X^Z
35 PRINT "X =";X
40 PRINT B
50 END
```

Statement 40 will print the number ________ .

4. The following is added to an HP-85 program, and (END LINE) is pressed after each line is typed.

```
290 PRINT "THIS QUIZ IS EASY."
290
```

As a result, statement 290 is ______________ .

5. A statement must have a(an) ________, while a command doesn't use one.

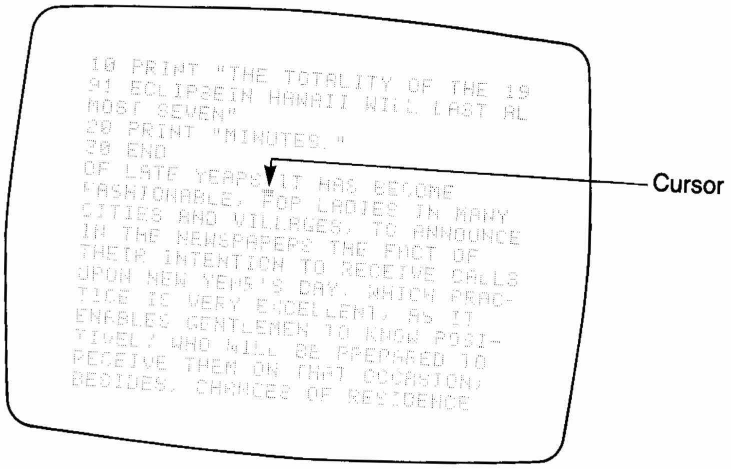
6. When you enter a statement or execute some commands, the ________ key is the last one you press.

7. When the BASIC word LET appears in a statement, that statement is a/an ______________ statement.

8. The highest number you may use for a statement number is ________ .

9. A (An) __________ within quotation marks in a command or statement is always preserved by the HP-85. Outside quotation marks, it is often ignored.

10. You have just commanded the HP-85 to load program "CH7" from your cartridge, and instead of loading, you get an error message. Before you pressed LOAD " C H 7 " END LINE, your screen looked like this:



```
10 PRINT "THE TOTALITY OF THE 19
91 ECLIPSEIN HAWAII WILL LAST AL
MOST SEVEN"
20 PRINT "MINUTES."
30 END
OF LATE YEARS IT HAS BECOME
FASHIONABLE, FOR LADIES IN MANY
CITIES AND VILLAGES, TO ANNOUNCE
IN THE NEWSPAPERS THE FACT OF
THEIR INTENTION TO RECEIVE CALLS
UPON NEW YEAR'S DAY. WHICH PRAC-
TICE IS VERY EXCELLENT, AS IT
ENABLES GENTLEMEN TO KNOW POSI-
TIVELY WHO WILL BE PREPARED TO
RECEIVE THEM ON THAT OCCASION;
BESIDES, CHANCES OF RESIDENCE
```

To avoid the error, and to preserve the three statements on the screen, the first key(s) you should have pressed before pressing LOAD is (are) ________ .

11. The BASIC word(s) ______ should always be used in the highest numbered statement in any program.

12. To understand an assignment statement, it helps to read the statement from ________.

13. In an assignment statement, the thing on the left of the = sign is always a/an ________.

14. In the following program, the cursor in statement 10 should be replaced by what character to produce the output shown? Note that this problem concerns statement 10, not statement 20.

```
10 PRINT "FRIENDLY:M"_"F"
20 PRINT "UNFRIENDLY:M"_"F"
30 END
```

Desired output:

```
FRIENDLY:MF
UNFRIENDLY:M          F
```

15. You've finished programming your 50 statement masterpiece, with statements numbered 1 to 50. You now realize you must add a new statement at the very beginning of your program. So the number of statements you must enter or change is ________.

16. Let's do number 15 over again. This time you've used statement numbers `10`, `20`, `30`, ... `490`, `500`. Now you must enter or change ________ statement(s).

17. In an assignment statement, a/an ________ is always on the right of the = sign.

Chapter 8

Put Numbers into Your Program While It's Running

Preview

In Chapter 8, you will:

- Enter a savings account program that asks for your starting balance, your interest rate, and the length of time your savings will be held, and then tells you your ending balance.
- Learn how to answer a program when it asks for numbers.
- Learn how NOT to answer a program when it asks for numbers.
- Learn how you can write a program that asks for one number or several numbers at a time.
- Learn how to determine the values of a running program's variables at any time from the keyboard.
- Learn how to delete with one command a group of statements from a program.
- Learn how to list only part of a program on the printer or screen.
- Learn about another kind of math power you can use in your program.
- Learn how the HP-85 tells you your cartridge tab is in the protect position.

Note: Since this chapter is rather long, I've given you two places where you may conveniently interrupt your studies for several hours or several days. These break points are page 8-110, step 15 and page 8-121, step 49. I'll tell you about these break points when you reach them.

( ) 1. Chapter 8 continues on your BASIC Training cartridge. When you finish with the program, you will continue Chapter 8 below. Now to see an important new BASIC word, `LOAD "CH8"` and press (RUN).

( ) 2. Now that you've entered your "Savings" program, play with it. I suggest below some values you might enter for your present balance, interest rate, and number of years held. Remember, enter means type in and press (END LINE). For instance, to enter a present balance of $500, press (5)(0)(0)(END LINE).

If you've entered "Savings" correctly, you should get the final balance figures shown in the table.

FIVE RUNS OF "SAVINGS" PROGRAM

Run No.	Present Balance B	Interest % I	No. Years Held Y	Final Balance
1	500	6.25	1	531.25
2	325.14	7.25	5	461.38
3	50000	8.75	100	219733648.56
4	.10	6.5	350+ 7/12	387540110.74
5	1000	10	500	4.96984196731E23

( ) 3. Notice, in run no. 3, how the $219 million balance you're leaving your great grandson was displayed entirely on the next line, as shown below. The PRINTALL output shown allows all of run 3 to be seen at one time. There is no need for you to execute PRINTALL.

```
          SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
50000

ENTER S&L INTEREST (%).
?
8.75

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
100

AFTER 100 YEARS YOUR ORIGINAL
$ 50000 WILL GROW TO $
 219733648.56 .
```

PRINTALL output of Run 3

The HP-85 is thoughtful enough to avoid splitting a number between two lines.

( ) 4. Run 4, shown below, is even more interesting:

```
               SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
.10

ENTER S&L INTEREST (%).
?
6.5

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
350+7/12

AFTER 350.583333333
YEARS YOUR ORIGINAL
$ .1 WILL GROW TO $
 387540110.74
```

PRINTALL output of Run 4

Besides showing how a tiny dime can grow into a mighty fortune for a whole town full of descendants to fight over, it shows other things as well:

a. You can enter a number smaller than one (.10).

b. You can enter 350 years 7 months as 350 + 7/12.

c. Since the number of years cannot be expressed exactly as a decimal, the HP-85 gives the number its maximum 12 digit treatment.

d. Since the rest of line 140, `YEARS YOUR ORIGINAL` cannot fit on the remainder of the line following the display of `AFTER 350.583333333`, the `YEARS YOUR ORIGINAL` is displayed entirely on the next line. When the HP-85 is asked to display or print the value of a variable or the result of a calculation in this manner, it avoids splitting either the number or any words associated with it.

e. The final zero of your 10 cent present balance was dropped when line 150 was executed:

```
$ .1 WILL GROW TO $
 387540110.74
```

The HP-85 can produce the final zero to give you `$ .10`, but this requires two BASIC words beyond the scope of this course, `PRINT USING` and `IMAGE`. Your Owner's Manual covers these starting on page 161.

( ) 5. Here is run 5:

```
                SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
1000

ENTER S&L INTEREST (%).
?
10

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
500

AFTER 500 YEARS YOUR ORIGINAL
$ 1000 WILL GROW TO $
 4.96984196731E23
```

What happened to the final balance? Did the HP-85 have a seizure? No, but the incredible size of your balance caused the HP-85 to shift gears, and display the number in a form of numerical shorthand called exponential notation. This is the only chapter in which I'll mention exponential notation in this course, and you'll have no Review Test questions on it. The `23` following the `E` at the end of this final balance number is the exponent of 10 when the number is written like this:

```
4.96984196731*10^23
```

or, using the symbols of algebra:

$$4.96984196731 \times 10^{23}$$

How does this number look without the exponent? Like this:

496,984,196,731,000,000,000,000

which results when the decimal point is moved right 23 places. So your distant descendants will share almost five hundred sextillion dollars.

( ) 6. You've already stored or recorded your "MONEY" program on your BASIC Training tape cartridge back in Chapter 5. Now let me help you store "Savings." First of all, you need a new name, since the HP-85 will accept no more than six characters including spaces as the name of a stored program. As you recall, these six or less characters must be enclosed in quotation marks when you use the `STORE` or `LOAD` commands.

( ) 7. Let's name your stored savings and loan program "SAVING."

( ) 8. Are you ready for another scheduled blunder? This time you're going to try to execute `STORE"SAVING"` with the cartridge tab in the protect position. First, check your BASIC Training cartridge to make sure the tab is moved opposite to the arrow.

( ) 9. The tape drive will probably run when you follow the keystrokes in step 10. The tab protects against recording, but the HP-85 is still able to move the tape.

( ) 10. With your cartridge protected against recording, insert it into the HP-85. You'll use the [TEST STORE] key just right of [REW LOAD]. [STORE], like [LOAD], is a typing aid.

Press [STORE] ["] [S] [A] [V] [I] [N] [G] ["] [END LINE].

The tape drive will probably run, then you'll hear a beep, and see:

```
STORE"SAVING"
Error 60 : WRITE PROTECT
```

So the tab does protect against recording, or, as the computer people say, against "writing."

( ) 11. Take your cartridge out, slide the tab to RECORD, and reinsert your cartridge.

( ) 12. Now move your cursor to the line on which you typed `STORE"SAVING"` and press [END LINE]. When the tape drive light goes out, "SAVING" is recorded on your tape.

( ) 13. Before you do anything else, remove your cartridge and

Move Your Cartridge Tab Against The Arrow to Protect Against Accidental Erasing.

You may leave your cartridge out of the HP-85 if you wish.

( ) 14. You have only copied "SAVING" from memory onto the tape, not transferred it. "SAVING" is still in memory. Press [RUN] to confirm this. Now that it's running, try to make more than 500 sextillion dollars.

( ) 15. You'll be doing more with "SAVING" in step 16, but if you wish to interrupt your study for a few hours or a few days, this is a good place. You may switch the HP-85 OFF without losing "SAVING." When you start again, simply execute `LOAD"SAVING"` and begin at step 16 (although you might wish to do a little reviewing to get up to speed). At step 49 in this chapter, page 8-121, I'll give you another break point.

( ) 16. If you're coming back from a break, make sure "SAVING" is loaded into the HP-85's memory. Now let's look at your first `INPUT` statement in detail. When

```
50 INPUT B
```

is executed during the running of your program, these things happen:

a. A `?` is displayed on the screen.

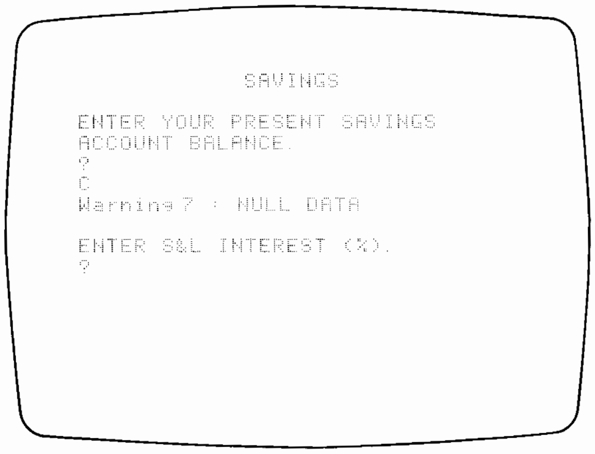
b. Your program waits for you to type in a number and to press END LINE.

The HP-85 has difficulty understanding any other response, as you shall see.

( ) 17. For the next several steps, please press keys only when I tell you to. Press RESET, then press RUN.

( ) 18. Instead of entering a number for your present savings account balance, enter a letter. Press C END LINE.

The HP-85 is uncertain. It's warning message:



```
              SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
C
Warning 7 : NULL DATA

ENTER S&L INTEREST (%).
?
```

which you've seen before in Chapter 7, says two things:

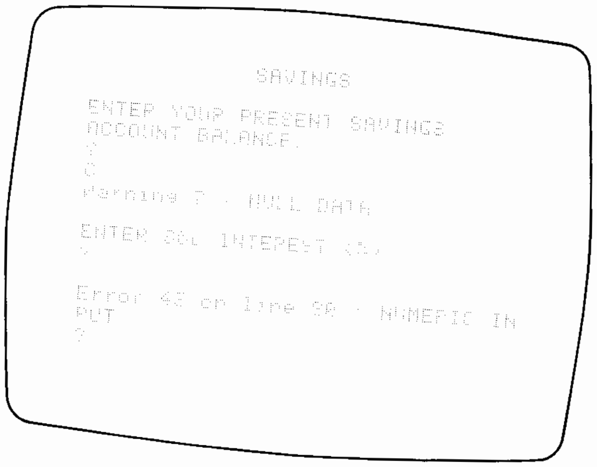
a. Your program has continued. Note that you're now asked to enter S & L interest.

b. You did not enter a value for present savings account balance. In this situation, the HP-85 assumes you want a value of zero and it assigns zero to your variable. You now have a zero savings account balance. Sorry about that.

What's happening?

The letter C is a valid variable name. Even though you haven't mentioned C in your program, the HP-85 accepts it, with a warning, and assigns the value 0 to it. In computer talk, you used a variable name in your program before it was initialized; that is, before it was given an initial value.

( ) 19. Now your program asks you your S & L interest. This time, press only END LINE. Do not type any letter or number. Now the HP-85 is a little more serious:



```
            SAVINGS
ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
0
Warning 7 : NULL DATA
ENTER S&L INTEREST (%).
?

Error 43 on line 80 : NUMERIC IN
PUT
?
```

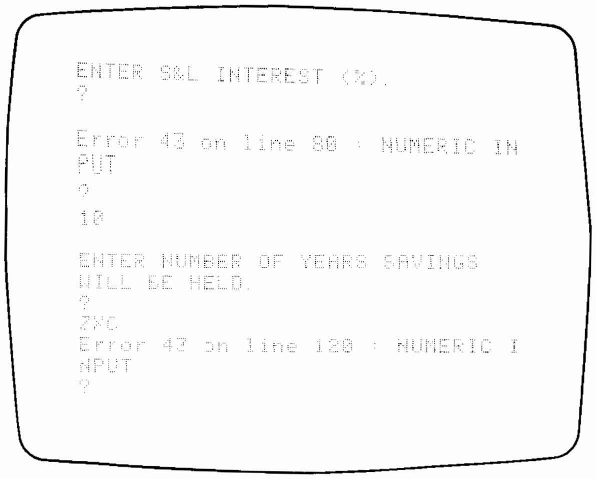
This message says: I didn't like that input. Try again.

Notice the question mark inviting you to give it another shot. Also notice that the HP-85 tells you the line number, `80`, where the problem occurred.

This time, make the HP-85 happy by entering a number, say 10%. Press (1) (0) (END LINE).

( ) 20. For the number of years savings will be held, press (Z) (X) (C) (END LINE).

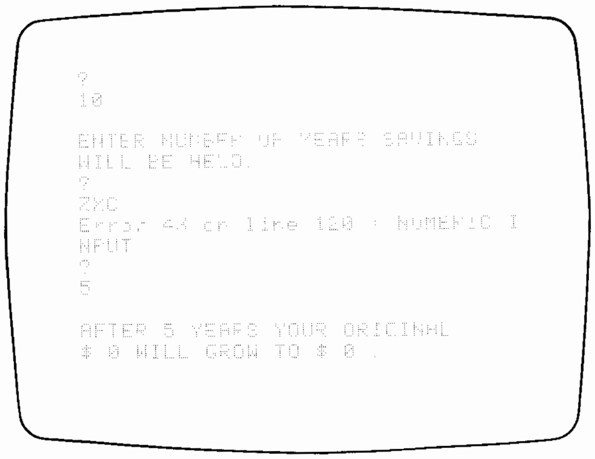
The line number has changed, but the message is the same:



```
ENTER S&L INTEREST (%).
?

Error 43 on line 80 : NUMERIC IN
PUT
?
10

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
ZXC
Error 43 on line 120 : NUMERIC I
NPUT
?
```

ZXC is not a valid variable name, so the HP-85 asks you to try again.

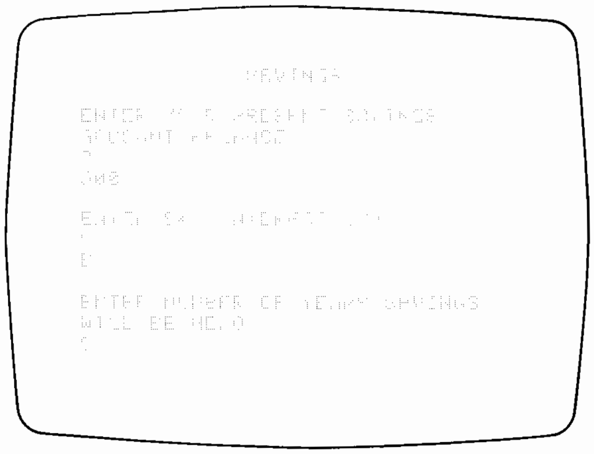
( ) 21. Try 5 years, and learn that when you start with nothing, you get nothing:



```
?
10

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
ZXC
Error 44 on line 120 : NUMERIC I
NPUT
?
5

AFTER 5 YEARS YOUR ORIGINAL
$ 0 WILL GROW TO $ 0 .
```

( ) 22. Clear your screen.

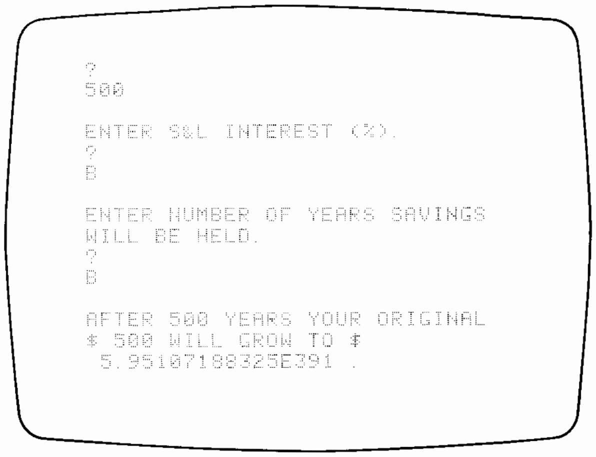
( ) 23. Since that's clearly an unsatisfactory result, run "SAVING" again. Start with $500 as your present savings account balance. For an interest rate, enter `B`, another valid variable name. Your screen should now look as shown below (except your screen will, of course, also display a cursor).



```
               SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE
?
500

ENTER % INTEREST RATE
?
B

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD
?
```

Where is `Warning 7 : NULL DATA`? Has the HP-85 grown tired of giving you warning and error messages?

( ) 24. Enter `B` again for the number of years savings will be held, and see:



```
?
500

ENTER S&L INTEREST (%).
?
B

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
B

AFTER 500 YEARS YOUR ORIGINAL
$ 500 WILL GROW TO $
 5.95107188325E391 .
```

( ) 25. Look at that final balance! 500 years in the future, your descendants will own at least half of the entire galaxy! That number is about 6 followed by 391 zeros!

Obviously, the HP-85 is generous, but is it sane?

( ) 26. The HP-85 is sane and, as always, obedient. When you entered `B` for S&L interest, the HP-85 checked to see if a value had been assigned to the variable `B`. Indeed it had. You entered `500` as your present savings account balance, and the corresponding `INPUT` statement, line 50, assigned `500` to `B`. So when you entered `B` in response to the next two `INPUT` statements (in response to the next two question marks), you were, in fact, entering 500% interest and 500 years for the length of time your savings were to be held.

( ) 27. Let me summarize what can happen if you enter a valid or invalid variable name in response to the question mark of an `INPUT` statement instead of the number the HP-85 expects:

a. You enter a *valid* variable name.

1. The variable name you entered has *not* been assigned a value. The HP-85 displays:

```
Warning 7 : NULL DATA,
```

assigns zero to the variable, and continues executing your program.

2. The variable name you entered *has* been assigned a value. The HP-85 takes this previously assigned value and assigns it to the variable used in your `INPUT` statement.

b. You entered an *invalid* variable name. The HP-85 displays:

```
Error 43 on line— : NUMERIC INPUT
?
```

The new ? the HP-85 displays invites you to try again to respond to the same INPUT statement.

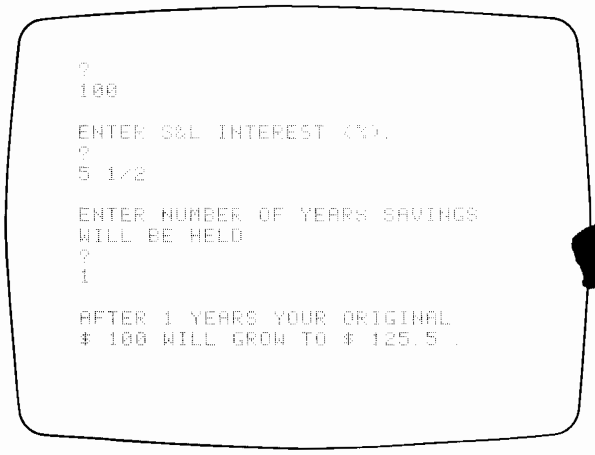
( ) 28. How about another try at running "SAVING"? First, clear your screen.

( ) 29. Start with $100, and for S&L interest, try 5½ instead of 5.5 (remember, ▲ means insert one space):

5 ▲ 1 / 2 END LINE

The HP-85 took it without even a burp! Maybe mixed numbers are OK after all (but don't bet on it).

Enter 1 for the number of years, and see:



```
?
100

ENTER S&L INTEREST (%).
?
5 1/2

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD
?
1

AFTER 1 YEARS YOUR ORIGINAL
$ 100 WILL GROW TO $ 125.5 .
```

You have picked one good savings and loan association. Your $100 grew to $ 125.5[§] in just one year! Before you spend it all, however, let's look again at that percentage interest.

The HP-85 generally ignores spaces, so it sees

5 1/2 as 51/2.

Dividing 51 by 2 gives an interest rate of 25.5%, which explains the impressive $125.50 return after one year. As you recall, you can avoid this problem by pressing 5 + 1 / 2 END LINE or 5 . 5 END LINE.

[§] The HP-85 can produce the final zero to give you $ 125.50, but this requires two BASIC words beyond the scope of this course, PRINT USING and IMAGE. Your Owner's Manual covers these starting on page 161.

( ) 30. Here is a new key you'll be using later in step 36 where I'll discuss how to recover from wrong inputs. The key is PAUSE. See Figure 62 for its location.

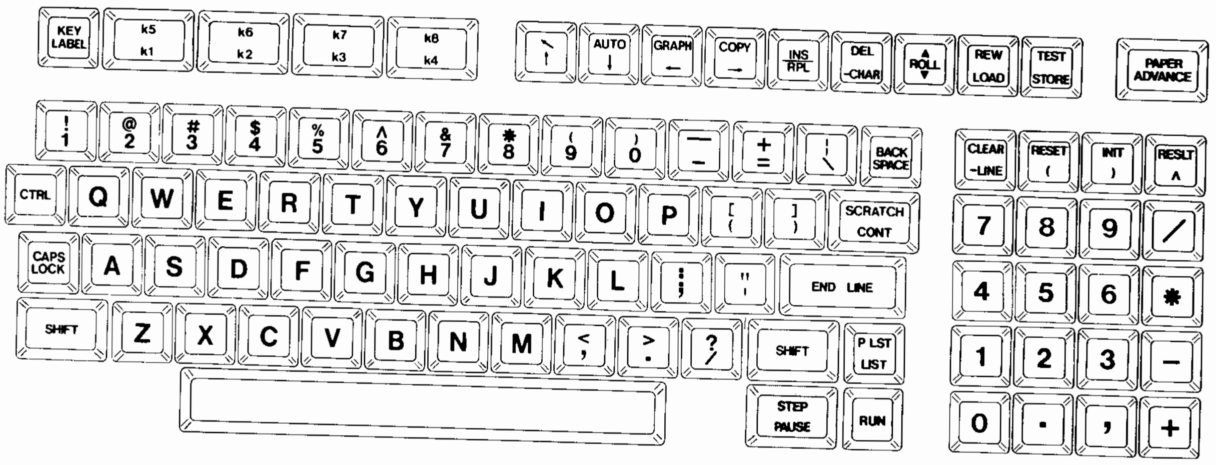


FIGURE 62

Location of PAUSE

( ) 31. PAUSE is closely related to CONT. You can pause a running program at any time by pressing PAUSE, and start it again at the same place by pressing CONT. When PAUSE is pressed, you hear a beep, and the program goes into a stand-by condition. When the program goes into stand-by, the HP-85 goes into calculator mode, and the keyboard becomes alive. You could type text and do calculations as you did in Chapters 1 and 2.

( ) 32. It's time for some definitions, one new (input loop) and two old.

Program mode: The HP-85 is in program mode whenever it is running a program.

Calculator mode: At all other times, except when turned off, the HP-85 is in calculator mode, allowing calculations, text writing and program writing.

Input loop: After a program has executed an `INPUT` statement, but before the entry or input has been made, the program is said to be in an **input loop.** If no entry is made, an input loop will last as long as the HP-85 and power last, or until the program is paused.

( ) 33. **To Change from Program to Calculator Mode:**

A way that always works: Press STEP PAUSE. A beep will sound, indicating that the running program has been paused. Another way that works *except during input loop:* Press almost any key. A beep will sound, indicating that the program has been paused. The pressed key's action will also occur, except when RUN is pressed.

Since pressing PAUSE always works, and performs no other action, I suggest the following:

Safety rule: To change from program mode to calculator mode, press PAUSE unless you're sure of what you're doing.

( ) 34. **DANGER!**

After you pause a running program, you can, by mistake, delete lines from the program you just paused, which could easily destroy the program in memory. Since the keyboard is alive after a program is paused, you are free to edit the program that was just paused. If you type a number which is the same as a line number in the program, and then press (END LINE), you have deleted that line.

( ) 35. PAUSE: **A BASIC Word**

A program may be paused using the PAUSE statement. However, no beep is heard when a PAUSE statement is executed. Whenever you saw TO PROCEED, PRESS CONT displayed by one of my programs, a PAUSE statement had just been executed. As soon as you pressed (CONT), my program began executing at the statement number immediately following the PAUSE statement.

( ) 36. Now—**How to Recover from Wrong Inputs**

a. Error realized *before* (END LINE) is pressed. No problem. Simply erase your mistake with (BACK SPACE), and try again.

b. Error realized *after* (END LINE) is pressed. The safest thing is to press (PAUSE) to halt your program (you'll hear a beep) and run your program again. (Whenever you press (PAUSE) while a program is running, the HP-85 tells you the program has halted by saying "beep.")

( ) 37. **Determining Values of Program Variables**

Follow this example in Figure 63.

Let's use "SAVING" one more time. Clear your screen and press (RUN). Start with $7000 at 8.06% interest. Now, before entering the number of years, you'd like to assure yourself that your program is using the correct values for B and I. This is done in calculator mode by typing the variable name and pressing (END LINE).

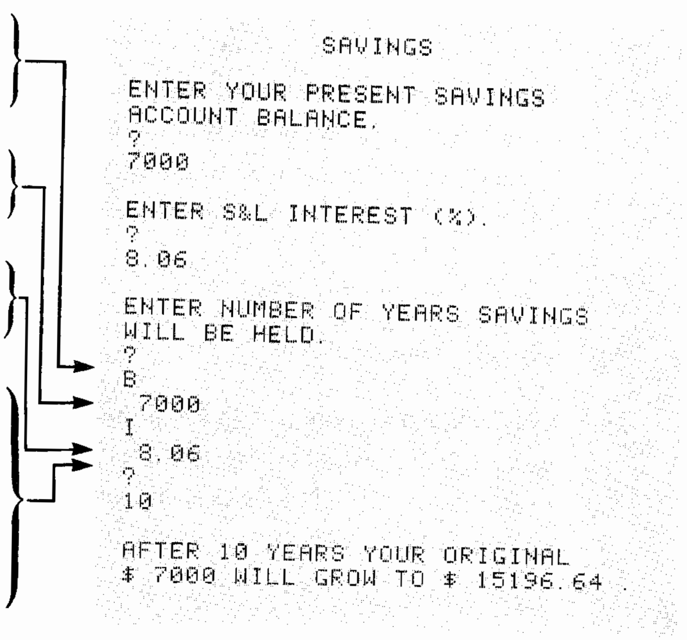
After responding to the first two input requests with 7000 and 8.06, press (PAUSE) to get into calculator mode. Next press (B)(END LINE) and see B's value, 7000, displayed. Similarly, press (I)(END LINE) and see 8.06. Since the correct values for B and I are being used, start your program just where it halted by pressing (CONT). Respond to the new question mark by entering 10 years, and see your ending balance of $ 15196.64.

PAUSE pressed here, after ? was displayed. The HP-85 now in calculator mode. Program execution halted.

Current value of B displayed by pressing B END LINE.

Current value of I displayed by pressing I END LINE.

CONT pressed to put the HP-85 back in program mode. Program execution resumed. ? is again displayed by "number of years" INPUT statement, but "Enter number of years" message not repeated. Value of 10 for Y is entered, and final balance calculated.



```
              SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE.
?
7000

ENTER S&L INTEREST (%).
?
8.06

ENTER NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
B
 7000
I
 8.06
?
10

AFTER 10 YEARS YOUR ORIGINAL
$ 7000 WILL GROW TO $ 15196.64 .
```

FIGURE 63

PRINTALL printout of "SAVING" showing how values of variables being used by program can be displayed during a pause.

( ) 38. DELETE: **A Command**

You'll be modifying "SAVING" soon, and this new command will be a big help. Figure 64 shows the location of a typing aid for this command.

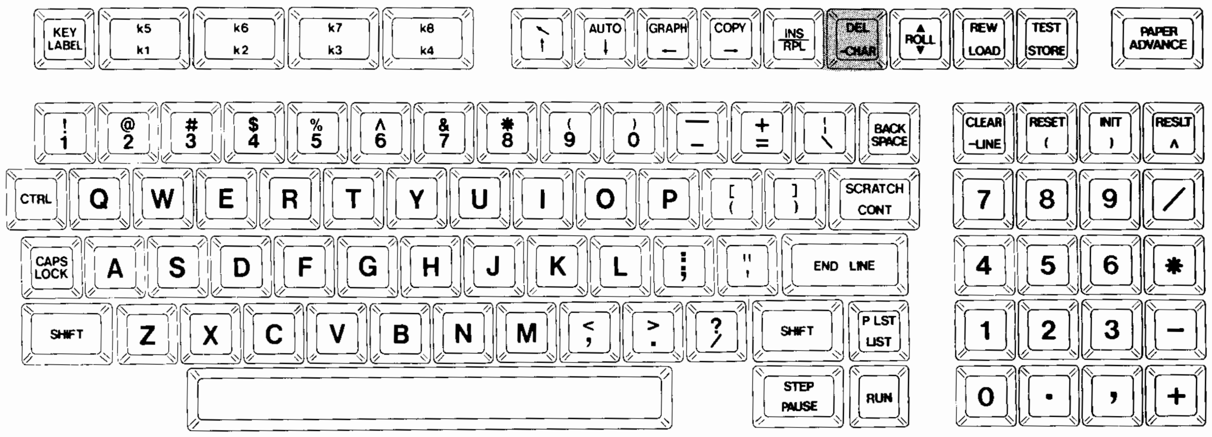


FIGURE 64

Location of shifted typing aid DEL

Say you wanted to delete one line, for instance, line 50. You would press

$\boxed{\text{SHIFT}} + \boxed{\text{DEL -CHAR}}\boxed{5}\boxed{0}\boxed{\text{END LINE}}$

Line 50 could also be deleted by pressing:

$\boxed{5}\boxed{0}\boxed{\text{END LINE}}$

Of course, the program having the unwanted line 50 must be in the HP-85's memory at the time, but the listing need not be displayed.

( ) 39. The real power of the DELETE command is exercised when you want to delete a group of lines, say 50 through 100 inclusive.

All that's needed is to press:

$\boxed{\text{DEL}}\boxed{5}\boxed{0}\boxed{,}\boxed{1}\boxed{0}\boxed{0}\boxed{\text{END LINE}}$

and the deed is done. You'll be deleting a group of lines from "SAVING" in step 41.

( ) 40. **Multiple Inputs**

So far, the INPUT statements you've used ask for only one value at a time. A program like "SAVING" can be shortened by making one INPUT statement do the work of three. First, I'll ask you to modify "SAVING" to combine your three INPUT statements into one, and then I'll give you some points to remember about multiple inputs.

( ) 41. By no coincidence, the lines I'd like you to remove from "SAVING" to modify it for multiple inputs are 50 to 100 inclusive. So press $\boxed{\text{SHIFT}} + \boxed{\text{DEL -CHAR}}\boxed{5}\boxed{0}\boxed{,}\boxed{1}\boxed{0}\boxed{0}\boxed{\text{END LINE}}$.

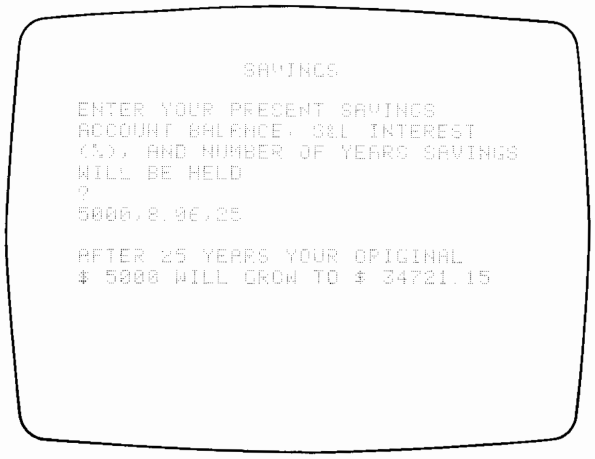
( ) 42. After completing that short task, list "SAVING" and change line 40 as shown in Figure 65. Don't forget to clear the remainder of your line ($\boxed{\text{-LINE}}$) before pressing $\boxed{\text{END LINE}}$. When you change line 40, you'll type over line 110. This will not change line 110. Prove this, after entering your new line 40, by listing "SAVING" again and seeing line 110 reappear unharmed. Now enter a new line 42 (see Figure 65), and finally make the important changes Figure 65 shows for line 120.

```
10 DISP "              SAVINGS"
20 DISP
30 DISP "ENTER YOUR PRESENT SAVI
NGS"
40 DISP "ACCOUNT BALANCE, S&L IN
TEREST"
42 DISP "(%), AND NUMBER OF YEAR
S SAVINGS"
110 DISP "WILL BE HELD."
120 INPUT B,I,Y
130 DISP
140 DISP "AFTER";Y;"YEARS YOUR O
RIGINAL"
150 DISP "$";B"WILL GROW TO $";
INT(B*(1+I/100)^Y*100+.5)/100;".
"
160 END
```

FIGURE 65

PRINTALL of "SAVING" CRT listing modified for multiple input

( ) 43. Clear your screen, then run this new "SAVING" and respond to the ? by pressing 5 0 0 0 , 8 . 0 6 , 2 5 END LINE. See Figure 66 for the output.



```
              SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE, S&L INTEREST
(%), AND NUMBER OF YEARS SAVINGS
WILL BE HELD
?
5000,8.06,25

AFTER 25 YEARS YOUR ORIGINAL
$ 5000 WILL GROW TO $ 34721.15
```

FIGURE 66

Execution of "SAVING" modified for multiple input

( ) 44. Notice these points about multiple inputs:

a. You enter the exact number of values required by the INPUT statement (line 120), and separate entered values by commas.

b. The values must be entered in the same order as they appear in the INPUT statement.

c. In the INPUT statement, variable names are separated by commas.

d. To recover from input error, all values must be entered again.

( ) 45. In the next step you will deliberately violate points a. and d. Follow your troubles in Figure 67.

( ) 46. Run "SAVING" again. First respond to the enter request by typing in only one value, 100, followed by END LINE. The HP-85 checks your work and discovers that your INPUT statement, line 120, wants three values. It then refuses to accept your single value and invites you to try again. This time enter the remaining two values, 5 and 1. Again the HP-85 does not accept any value. Don't despair! Try again by entering all three values. Now the HP-85 rewards you with the final balance, $105.

Attempt to enter one value—not accepted. ⟶

Attempt to enter remaining 2 values—neither accepted. ⟶

To recover, must enter correct number of values. ⟶

```
             SAVINGS

ENTER YOUR PRESENT SAVINGS
ACCOUNT BALANCE, S&L INTEREST
(%), AND NUMBER OF YEARS SAVINGS
WILL BE HELD.
?
100
Error 43 on line 120 : NUMERIC I
NPUT
?
5,1
Error 43 on line 120 : NUMERIC I
NPUT
?
100,5,1

AFTER 1 YEARS YOUR ORIGINAL
$ 100 WILL GROW TO $ 105 .
```

FIGURE 67

PRINTALL execution of "SAVING" showing multiple input errors

( ) 47. **CONCLUSION: To correct a multiple input error, you must re-enter all values, not just one or two or a few.**

( ) 48. You won't need this multiple input version of "SAVING" anymore, so do *not* store your program. If you do, using the same name "SAVING," you'll lose the version now stored (recorded) on tape. You'll need that stored version later.

( ) 49. Here's another good place for a break.

( ) 50. If you've just come back from a break, welcome!

( ) 51. **How To Move the "?" of INPUT**

When an INPUT statement is used to assign values to variables, a message is almost always displayed or printed telling the user what values to enter. Most often this message appears just before the INPUT statement, and many times this message can be a question. In such cases, the question mark generated by the INPUT statement can be easily moved to the end of the message. Figure 68 shows an example of how a conventional INPUT statement can be converted into one that moves the question mark. The trick is to put a semicolon at the end of the message preceding the INPUT statement. This semicolon goes *outside* the final quotation mark. Also, no period is used at the end of the message.

Program segment. Conventional DISP statement.

```
200 DISP "ENTER YOUR DIMENSIONS.
"
210 INPUT A,B,C
```

Program segment output. INPUT "?" displayed on separate line.

```
ENTER YOUR DIMENSIONS.
?
```

Same program segment. DISP statement modified to put INPUT "?" at end of DISP message. Period removed, semicolon put after final quotation mark.

```
200 DISP "WHAT ARE YOUR DIMENSIO
NS";
210 INPUT A,B,C
```

Program segment output. INPUT "?" follows displayed message.

```
WHAT ARE YOUR DIMENSIONS?
```

FIGURE 68
How to move "?" displayed by INPUT statement to end of message displayed by preceding DISP statement.

( ) 52. **PROBLEM:**

Load your "SAVING" program and revise it in a similar manner to that shown in Figure 68 to make all INPUT messages questions and to move each INPUT "?" to the end of each question. Run your revised program. By the way, did you remember to remove your periods? Compare your revised listing and output with mine. See pages H-21 and H-22, No. 35.

( ) 53. When you're satisfied with your revision, move your cartridge tab to RECORD, and store your revised program using the same name "SAVING." When your execute STORE"SAVING" you will destroy your previously recorded version, but that's fine, this version's better. Also, you'll be using this revised version in Chapter 9.

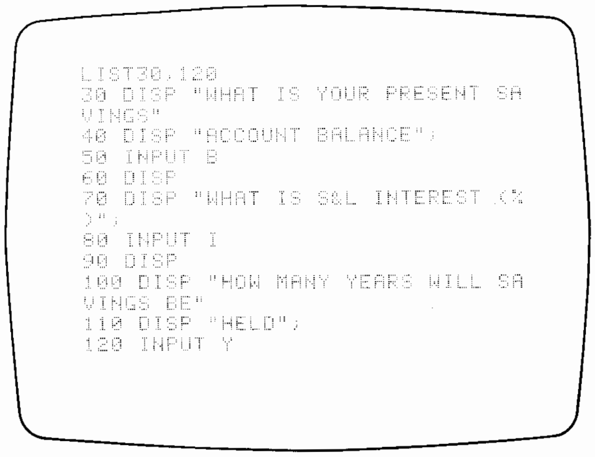
IMPORTANT: After storing "SAVING," move your cartridge tab to the protect position (opposite direction from the arrow).

( ) 54. **More LIST and PLIST Power**

Just as DELETE can delete a group of lines, so can the LIST and PLIST commands list portions of a program. Here's how you can list lines 30 through 120 of "SAVING":

Press (L) (I) (S) (T) (3) (0) (,) (1) (2) (0) (END LINE)

If your "SAVING" is like mine, you should see these ten statements on your screen:



```
LIST30,120
30 DISP "WHAT IS YOUR PRESENT SA
VINGS"
40 DISP "ACCOUNT BALANCE";
50 INPUT B
60 DISP
70 DISP "WHAT IS S&L INTEREST (%
)";
80 INPUT I
90 DISP
100 DISP "HOW MANY YEARS WILL SA
VINGS BE"
110 DISP "HELD";
120 INPUT Y
```

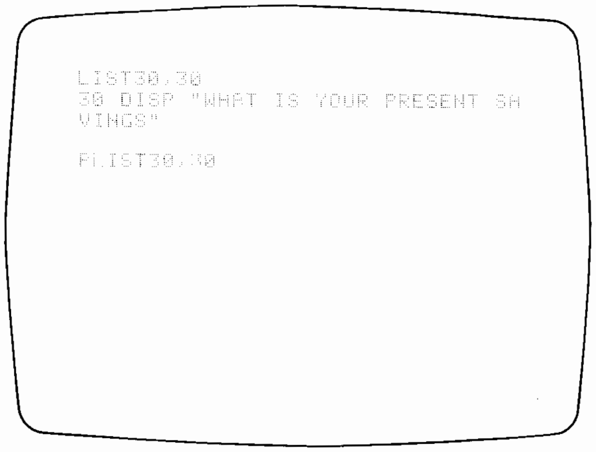
( ) 55. To print the same program segment:

Press P L I S T 3 0 , 1 2 0 END LINE

Now your printer should show the same ten lines.

( ) 56. Note that you must press P L I S T, not the P LST abbreviation on the key cap.

( ) 57. You can also display or print just one line by typing the same line number both before and after the comma, then pressing END LINE. To get practice, reproduce the examples shown below:



```
LIST30,30
30 DISP "WHAT IS YOUR PRESENT SA
VINGS"
PLIST30,30
```

```
30 DISP "WHAT IS YOUR PRESENT S
   AVINGS"
```

( ) 58. Finally, you can display or print all the lines of a program starting with any line by pressing

L I S T [line number] END LINE

or

P L I S T [line number] END LINE

Say you had a program in the HP-85's memory with statements numbered 10 to 2000, using the standard interval of 10 between statement numbers. If you pressed

L I S T 5 0 0 END LINE

the screen would fill with program statements starting with number 500. To get the next screenful, you would press the single key P LST LIST.

This version of the `LIST` command (L I S T [line number] END LINE) is perhaps the most often used. When you're writing a program, you often wish to see one or more screenfuls of statements starting with a particular line number. To print the same statements starting with 500, you would press

P L I S T 5 0 0 END LINE

The HP-85 would start with line 500, and print all the remaining statements in the program. Pressing any key would stop the printout.

( ) 59. **More About Functions**

The `INT` function used in "SAVING" is a typical function. You supply a number and `INT` will give you the largest integer less than or equal to that number. The number is called the argument, and the function gives you the value. For example:

function ——► `INT(4*.7)` gives `2` ◄—— value

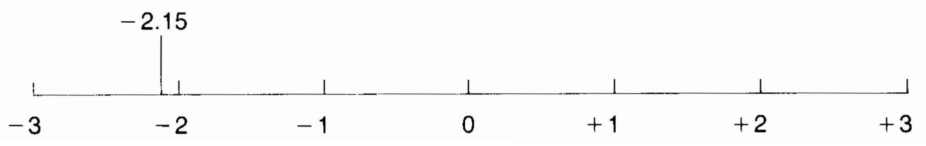
argument

As statement 150 in "SAVING" shows, an argument can be a complicated expression, but it always boils down to a single number. The function then acts on that single number to give the value. Looking again at our example:

`INT(4*.7)` boils down to `INT(2.8)` which gives `2`

( ) 60. What about an `INT` function with a negative argument, like `INT(-2.15)`?

It will help to look at a number line.



The largest integer less than or equal to −2.15 is −3. Whether the argument X is positive or negative, you can always evaluate INT(X) by putting X on the number line and moving left to the next integer.

( ) 61. **Integer Quiz**

Fill in all the blanks below, then check your answers against mine shown at the bottom of page 8-127. Remember, to find the INT value of an argument, place the argument (number) on the number line above, and move left to the first integer. That first integer is the value you're looking for.

a. INT (2.57) =

b. INT (−1.01) =

c. INT (−.52) =

d. INT (1.97) =

e. INT (−1.97) =

f. INT (.11) =

Summary of Chapter 8

- INPUT: **A BASIC word**

  INPUT identifies an INPUT statement.

  Reference: Program "CH8" display.

- **Input statement:**

  Examples:

```
35  INPUT R
80  INPUT X,Y,Z
```

  When executed, statements 35 and 80 each display a "?" on the screen, and the HP-85 goes into an **input loop.** The program continues to run, but it idles, like a car at a red light. The next statement beyond 35 (or whatever number the INPUT statement has) is not executed until the light turns green. The user turns the light green by entering a number or numbers in response to the displayed "?." The INPUT statement's variable(s) (R or X, Y and Z in the examples) is (are) assigned the value(s) entered by the user. This number entry and value assignment turns the light green, and program execution continues.

  Reference: Program "CH8" display, page 8-110, step 16, and page 8-120, step 44.

- **To enter a mixed number** like 3⅝, press [3] [+] [5] [/] [8] [END LINE]. Reference: Program "CH8" display and page 8-115 step 29.

- `INPUT` **warnings and errors**

  - If `INPUT` wants a number, and you enter a *valid* variable name, and

    a. if entered variable name has *not* been assigned a value, the HP-85 enters 0, displays

    ```
    Warning 7 : NULL DATA
    ```

    and continues program execution.

    b. if entered variable name *has* been assigned a value, the HP-85 enters that value, and continues program execution.

  - If `INPUT` wants a number, and you enter an *invalid* variable name, or press only (END LINE), the HP-85 will display:

    ```
    Error 43 on line - : NUMERIC
    INPUT
    ?
    ```

    After the error message is shown, a `?` is displayed, inviting you to try to enter your number again.

  Reference: Pages 8-111 through 8-115, steps 18-27.

- **How to recover from input errors**

  Error realized *before* (END LINE) is pressed—before entry is completed.

  Erase wrong characters ((SHIFT) + (BACK SPACE)) and try again.

  Error realized *after* (END LINE) is pressed—after entry is completed.

  SAFETY RULE: Start over. Press (PAUSE), then press (RUN).

  Reference: Page 8-117, step 36.

- **Multiple inputs**

  - Enter exact number of values required by `INPUT` statement, separated by commas.
  - Values must be entered in same order as they appear in `INPUT` statement.
  - To recover from input error, all values must be entered again.

  Reference: Pages 8-119 through 8-121, steps 40-47.

- **To move `?` displayed by `INPUT` statement** to the end of the previous `DISP` or `PRINT` statement, type semicolon as last character of `DISP` or `PRINT` statement, after final quotation mark and just before (END LINE).
  Reference: Page 8-121, step 51.

- `Error 60 : WRITE PROTECT`

  This error is displayed when an attempt is made to execute a `STORE` command with the cartridge RECORD tab in the protect position. To recover, remove cartridge, move the tab to "RECORD," insert cartridge, and try `STORE` command again.

  Reference: Page 8-110, steps 8-13.

- `STORE`: **A command**

  Copies program in memory onto a tape. The program in memory is unaffected. Reference: Page 8-110, steps 11-14.

- **Program mode:** Whenever the HP-85 is running a program, it is in program mode. Reference: Page 8-116, step 32.

- **Calculator mode:** Whenever the HP-85 is *not* running a program, unless turned off, the HP-85 is in calculator mode, allowing calculations, text writing and program writing. Reference: Page 8-116, step 32.

- **Input loop:** While an `INPUT` statement waits for an entry, the program is in an input loop. Reference: Page 8-116, step 32.

- **The (PAUSE) key**

  Pressing (PAUSE) not only halts a running program, but also sounds a beep and puts the HP-85 into calculator mode.

  Reference: Page 8-116, steps 30 and 31.

- **Pressing most keys during running program, EXCEPT DURING INPUT LOOP, halts program** and puts the HP-85 into calculator mode. Also, the action of the pressed key will occur except when (RUN) is pressed. Reference: Page 8-116, step 33.

- **SAFETY RULE: Pause a running program by pressing (PAUSE)** rather than some other key. Reference: Page 8-116, step 33.

ANSWERS TO INTEGER QUIZ

a. INT (2.57) = 2
b. INT (−1.01) = −2
c. INT (−.52) = −1
d. INT (1.97) = 1
e. INT (−1.97) = −2
f. INT (.11) = 0

- PAUSE: **A BASIC word**

  When a PAUSE statement is executed, the effect is identical to pressing PAUSE, except no beep is heard.

  Reference: Page 8-117, step 35.

- **CAUTION! When a program is paused, a line of the program can be accidently deleted** by typing a line number and pressing END LINE. Reference: Page 8-117, step 34.

- **To determine value of program variable while program is running:**

  Press PAUSE [variable name] END LINE to see current value displayed.

  Then press CONT to resume program operation.

  Example: To see value of program variable E while program is running, press PAUSE E END LINE.

  Reference: Page 8-117, step 37.

- **The** DEL **key**

  This typing aid displays the word DELETE when pressed.

  Reference: Page 8-118, step 38.

- DELETE: **A command**

  Deletes one or a group of statements from program in memory. For instance, to delete line 75, press

  SHIFT + DEL -CHAR 7 5 END LINE

  To delete a group of statements from a program in memory, say 110 through 225 inclusive, press

  SHIFT + DEL -CHAR 1 1 0 , 2 2 5 END LINE

  Reference: Pages 8-118 and 8-119, steps 38 and 39.

- LIST: **A command**

  Lists program or program segment on screen.

  Reference: Pages 8-122–8-124, steps 54, 57 and 58.

- PLIST: **A command**

  Lists program or program segment on printer.

  Reference: Pages 8-122–8-124, steps 54-58.

- **Review of Chapter 4 LIST and PLIST material**

  - **To display a listing of all program statements in memory,** press LIST to get first screenful of statements, then press LIST again to get next screenful of statements, and so on.

  - **To print a listing of all program statements in memory,** Press SHIFT + P LST LIST. To stop printout, press PAUSE.

- **To list one statement:**

  On screen:
  Press L I S T [line no.] , [line no.] END LINE

  On printer:
  Press P L I S T [line no.] , [line no.] END LINE

  Example: To list line 35 on the printer, press

  P L I S T 3 5 , 3 5 END LINE

  Reference: Page 8-123, step 57.

- **To list a group of statements, but not the whole program:**

  On screen:
  Press L I S T [first line no.] , [last line no.] END LINE

  On printer:
  Press P L I S T [first line no.] , [last line no.] END LINE

  Example: To list lines 100 through 140 inclusive on the screen, press

  L I S T 1 0 0 , 1 4 0 END LINE

  Reference: Pages 8-122 and 8-123, steps 54-56.

- **To list all program lines starting with a particular line number:**

  On screen:
  Press L I S T [line no.] END LINE to get first screenful of statements, then press LIST to get each succeeding screenful of statements.

  On printer:
  Press P L I S T [line no.] END LINE. To stop printout before last program line, press PAUSE.

Example: To list a screenful of lines starting with line 340, press

L I S T 3 4 0 END LINE

To get additional screenfuls of higher numbered lines, press LIST repeatedly.

Reference: Page 8-124, step 58.

- **Function**

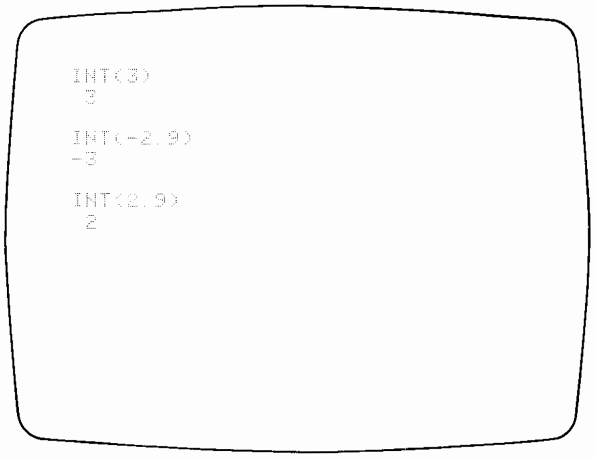
When executed, a function does something to the number within ( ) and produces another number. A function has three parts. Using `INT(X)` as an example:

$$\begin{tikzcd} \text{function} \arrow[r] & \texttt{INT(X)} \text{ gives } Y & \text{value} \arrow[l] \\ & \text{argument} \arrow[u] & \end{tikzcd}$$

Reference: Program ''CH8'' display, and page 8-124, step 59.

- `INT(X)`: **A function**

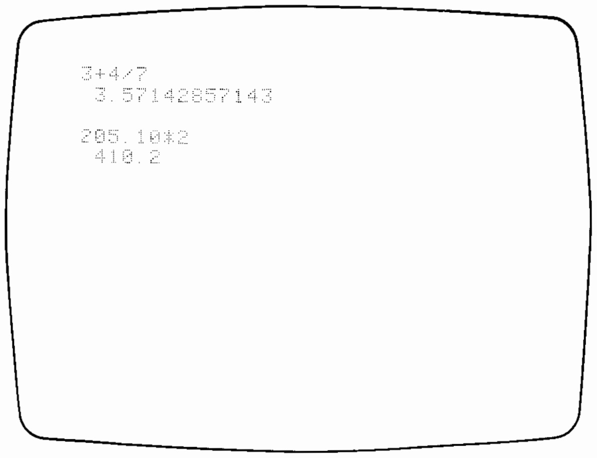
Gives the largest integer equal to or less than X. Examples:



```
INT(3)
 3
INT(-2.9)
-3
INT(2.9)
 2
```

Reference: Program ''CH8'' display, and pages 8-124 and 8-125, steps 59-61.

- **The HP-85 expresses an inexact decimal as a 12 digit number.** (If the 12th digit of an inexact decimal is zero, the final zero(s) is (are) normally dropped.) If a number is exact, the HP-85 normally drops trailing zeros. Examples:



```
3+4/7
 3.57142857143

205.10*2
 410.2
```

Reference: Page 8-108, step 4; also page 8-115, step 29.

Review Test for Chapter 8

The answers are on page 8-133, immediately following this Review Test.

1. Say you're running a program, and it displays the following:

   ```
   HOW OLD ARE YOU?
   ```

   You press these keys: 2 6 ▲ 1 / 2 END LINE
   (space)

   The HP-85 now believes you are how old?

2. You run the same program again, only this time, in response to `HOW OLD ARE YOU?`, you press W 7 END LINE. The HP-85 beeps and displays:

   ```
   Warning 7 : NULL DATA
   ```

   Now how old does the HP-85believe you to be?

3. Let's look at a partial listing of this same program:

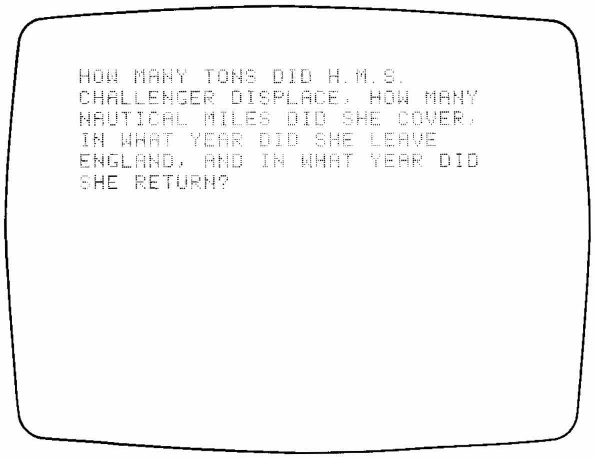
```
120 LET M=1000
130 LET Z=600
140 LET P=3
150 DISP "HOW OLD ARE YOU";
160 INPUT A
```

You run it again, and when `HOW OLD ARE YOU?` is displayed, you press (Z)(END LINE). How old does the HP-85 believe you to be this time?

4. Say you are 26½ years old. When this same program is run, you make an incorrect entry when responding to `HOW OLD ARE YOU?`. While the program continues to run, you realize your mistake. You wish to correct your error in a safe, sure way. What are the first two keys you should press?

5. In December 1872, the steam corvette H.M.S. Challenger, 2306 tons displacement, left England on what became one of the most important voyages of discovery in the history of oceanography. She traversed 68,890 nautical miles in the Atlantic, Pacific and Southern oceans, and returned to England in May 1876.

   You are running a program on the HP-85 testing your knowledge of the Challenger expedition. Here is one of the displayed questions:



```
HOW MANY TONS DID H.M.S.
CHALLENGER DISPLACE, HOW MANY
NAUTICAL MILES DID SHE COVER,
IN WHAT YEAR DID SHE LEAVE
ENGLAND, AND IN WHAT YEAR DID
SHE RETURN?
```

   What keys should you press to answer this question correctly?

6. The Challenger test question in problem 5 uses these variable names:

   T means tons displacement

   Y1 means year of departure

   Y2 means year of return

   M means nautical miles covered

   Write the `INPUT` statement relating to this Challenger question. To agree with my answer, use line number 100. No need to write the `DISP` statements associated with this question. Just write the `INPUT` statement.

7. You have answered the Challenger question shown above, and later, as the program is running, you wish to review the answer you gave for the number of tons displaced by the Challenger during her famous voyage. What keys do you press to review this number and to start the program running again where you stopped it?

8. You're writing a program describing how to build a true perpetual motion machine. You have some concern over a part of your description, contained in statements 150 through 325 inclusive. You are now in calculator mode. You wish to list these statements on the printer. What keys do you press?

9. After reviewing the statements you printed in problem 8, you realize they must be removed. To remove them from your program, what keys do you press?

Answers to Review Test Questions for Chapter 8

1. 130.5 years old

   Reference: Program "CH8" display and page 8-115, step 29.

2. 0 years old

   Reference: Page 8-111, step 18 and page 8-114, step 27.

3. 600 years old

   Reference: Pages 8-113 and 8-114, steps 23-27.

4. PAUSE RUN

   Reference: Page 8-117, step 36.

5. 2 3 0 6 , 6 8 8 9 0 , 1 8 7 2 , 1 8 7 6 END LINE

   Reference: Page 8-120, step 44.

6. `100 INPUT T,M,Y1,Y2`

   Reference: Page 8-120, step 44.

7. PAUSE T END LINE CONT

   Reference: Page 8-117, step 37.

8. P L I S T 1 5 0 , 3 2 5 END LINE

   Reference: Pages 8-122 and 8-123, steps 54-56.

9. SHIFT + DEL CHAR 1 5 0 , 3 2 5 END LINE or
   D E L E T E 1 5 0 , 3 2 5 END LINE

   Reference: Pages 8-118 and 8-119, steps 38-39.

Chapter 9

Plan Your Program

Preview

In Chapter 9, you will:

- Learn about four simple questions whose answers can help you write programs.
- Improve your "SAVING" program to include the effect of inflation.
- Learn how to chart the flow of your program before you write it.
- Learn how to easily renumber your program's statements.
- Learn how you can order your `DISP` statements to print and your `PRINT` statements to display. Also learn why anyone would wish to give such strange orders.
- Learn one way to make your programs friendly.
- Learn how to write messages that are meant only for the eyes of one reading your listing—they neither print nor display.
- Write a program for "Pail-Face" Trudgebottom.
- Say "Adieu!" to a good friend, `LET`, and "Allo!" to a trimmer, more popular assignment statement.

Just as each of us writes in his own way, so does each of us plan differently. A planning routine that works fine for one may not be all that great for another. However, it might be wise to start with the scheme I'll give you, and then modify it to suit your own way of working.

Program Planning Questions

Before writing your first statement, answer these questions:

1. What answers do I want?
2. What things do I know?
3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?
4. How can BASIC and the HP-85 help me find answers?

Say you were faced with the task of writing your "SAVING" program printed by program "CH8." Here's an acceptable set of answers to the program planning questions:

1. What answers do I want?

   I want to know the ending balance in a savings account after a certain number of years. I want to get this answer for a variety of interest rates, and a number of different years. Also, I'd like the ending balance rounded to the nearest cent.

2. What things do I know?

   I know the formula needed to calculate ending balance.

   B = starting balance
   I = annual percentage rate of interest
   Y = number of years savings held

   $$\text{Ending balance} = \text{INT}(B(1 + I/100)^{Y} \times 100 + 0.5)/100$$

3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?

   I'll solve the formula using chosen values for B, I and Y.

4. How can BASIC and the HP-85 help me find answers?

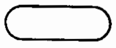
   `DISP` and `INPUT` statements will be useful. I'll have the HP-85 display requests for B, I and Y values, then display the answer. `INPUT` statements will allow me to enter these values into the program. I'll plug the formula directly into the `DISP` statement which gives the answer.

After you've gained some programming experience, you'll write a program as simple as "SAVING" without writing these answers out on paper. But you'll still answer these questions, or ones like them, even though you may not be aware of it. When you first learn to drive a car, you're very conscious of every driving question and answer. Later, ordinary driving becomes second nature.

As you program, you'll form programming habits just as a driver forms driving habits. On the other hand, no matter how experienced you become, some programming tasks will profit from careful planning just as an experienced race driver, to be successful, must plan well.

Flowcharts

An important planning tool is the flowchart. As the name suggests, a flowchart charts the flow of a program. Figure 69 shows the flowchart for "SAVING." The two kinds of symbols used in all my flowcharts are:

Start or END 

Task, like a calculation, a `DISP` or `PRINT`, or an `INPUT`

The arrows connecting the symbols in a flowchart show the direction of program execution. If no arrow is shown, program flow moves from the top of the flowchart towards the bottom.

The first few flowcharts I'll show you will give the program's line numbers that correspond to each task (or to END). These line numbers will help you see the relation between the flowchart and its program.

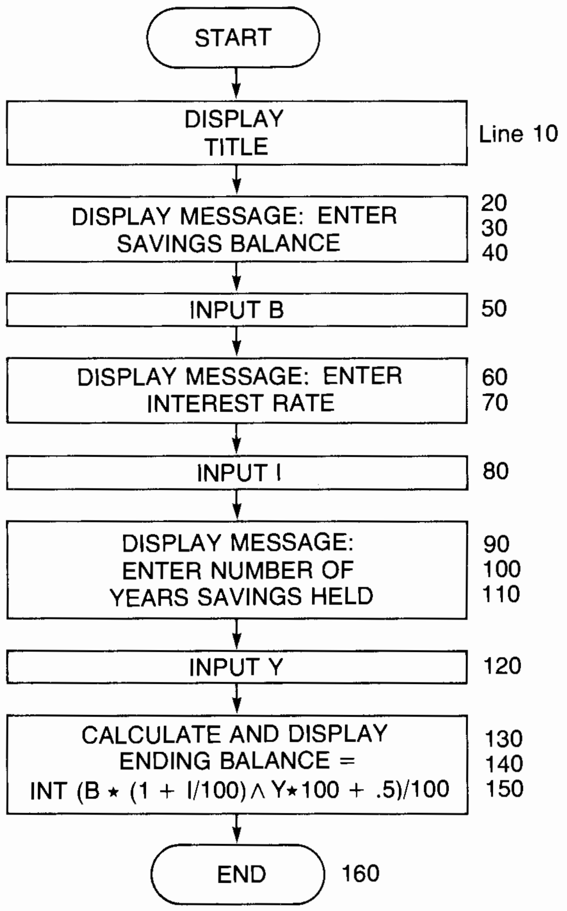

START
DISPLAY TITLE
Line 10
DISPLAY MESSAGE: ENTER SAVINGS BALANCE
20 30 40
INPUT B
50
DISPLAY MESSAGE: ENTER INTEREST RATE
60 70
INPUT I
80
DISPLAY MESSAGE: ENTER NUMBER OF YEARS SAVINGS HELD
90 100 110
INPUT Y
120
CALCULATE AND DISPLAY ENDING BALANCE = INT (B ⋆ (1 + I/100)∧Y⋆100 + .5)/100
130 140 150
END
160

FIGURE 69

Flowchart for "SAVING"

Flowcharts make it easy to:

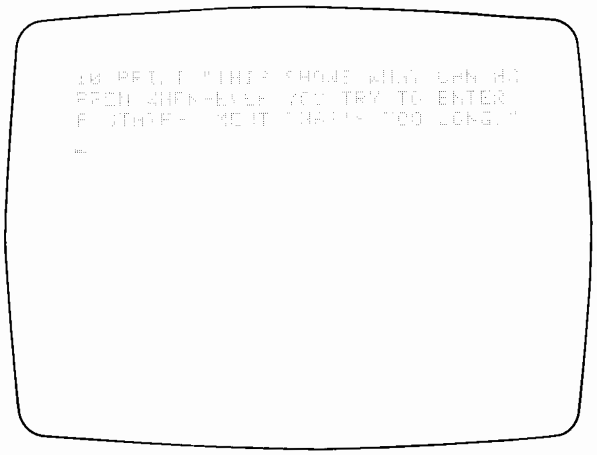
1. Plan program
2. Change program plan
3. Explain program to others (and later to yourself)

As the programmer, you're the one to decide how detailed your flowchart should be. It's your tool—use it in a way that helps you most.

EXAMPLE: Entering Long Statements

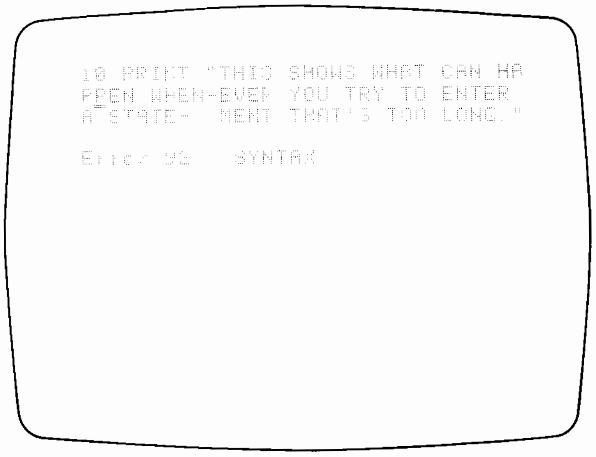
You'll soon be improving your ''SAVING'' program, and in the process, you may choose to write some statements over two screen lines long. Writing such long statements can get you into a new kind of trouble that you're going to experience right now.

Type *but do not enter* the following statement:



```
10 PRINT "THIS SHOWS WHAT CAN HA
PPEN WHEN-EVER YOU TRY TO ENTER
A STATE-  MENT THAT'S TOO LONG."
```

Now press END LINE and see:



```
10 PRINT "THIS SHOWS WHAT CAN HA
PPEN WHEN-EVER YOU TRY TO ENTER
A STATE-  MENT THAT'S TOO LONG."

Error 92 : SYNTAX
```

The HP-85 is confused, but why? Notice that your statement contains three full lines of characters, including spaces, or 3 × 32 = 96 characters. As you recall, the maximum number of characters the HP-85 allows in one statement is 95. So 96 is one too many, resulting in an error when END LINE is pressed.

Whenever you try as shown above to enter a statement over 95 characters long, the HP-85 will give you

```
Error 92 : SYNTAX
```

and put your cursor under the second character of the second line of your statement.

PROBLEM: Write the "Inflation" program

As some of us have learned to our sorrow, a savings account increases in money but not necessarily in buying power. Following the steps below, modify your ''SAVING'' program to include the effect of inflation. A flowchart for ''Inflation'' is shown in Figure 70.

1. Load your ''SAVING'' program.
2. Print a listing of your ''SAVING'' program for later reference.
3. See statements 60, 62 and 64 in Figure 71 on page 9-139. These generate the enter message and the `INPUT` statement for inflation rate.

4. Add these three statements (60, 62, 64) to your "SAVING" program.

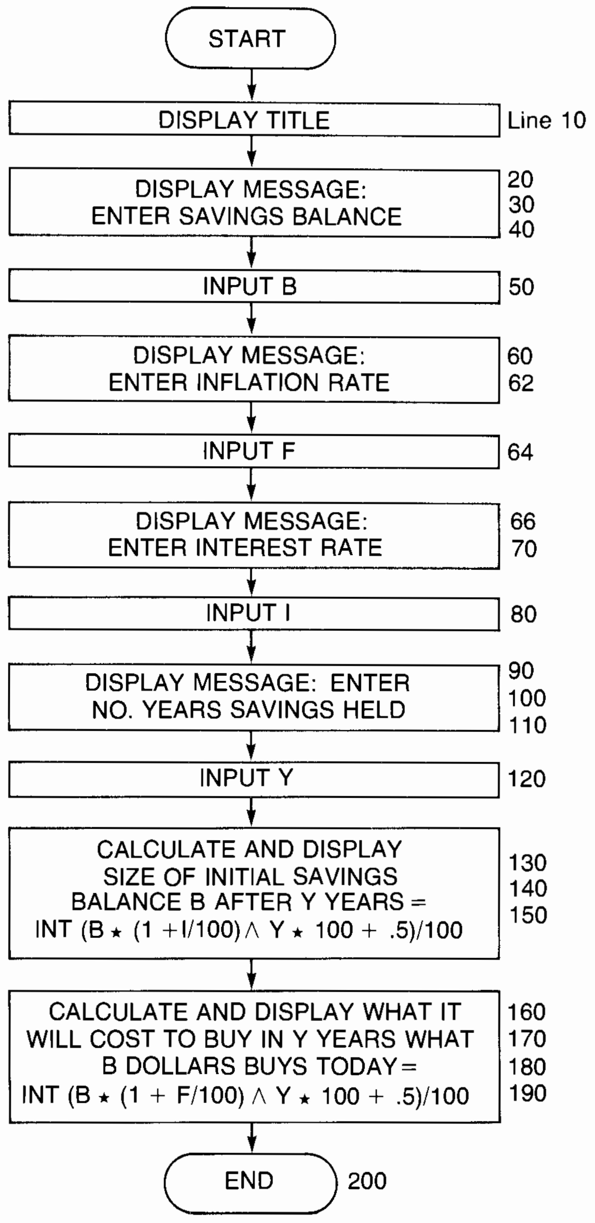

START
DISPLAY TITLE
Line 10
DISPLAY MESSAGE: ENTER SAVINGS BALANCE
20
30
40
INPUT B
50
DISPLAY MESSAGE: ENTER INFLATION RATE
60
62
INPUT F
64
DISPLAY MESSAGE: ENTER INTEREST RATE
66
70
INPUT I
80
DISPLAY MESSAGE: ENTER NO. YEARS SAVINGS HELD
90
100
110
INPUT Y
120
CALCULATE AND DISPLAY SIZE OF INITIAL SAVINGS BALANCE B AFTER Y YEARS = INT (B ⋆ (1 +I/100) ∧ Y ⋆ 100 + .5)/100
130
140
150
CALCULATE AND DISPLAY WHAT IT WILL COST TO BUY IN Y YEARS WHAT B DOLLARS BUYS TODAY= INT (B ⋆ (1 + F/100) ∧ Y ⋆ 100 + .5)/100
160
170
180
190
END
200

FIGURE 70

Flowchart for "Inflation"

```
10 DISP "           INFLATION"
20 DISP
30 DISP "WHAT IS YOUR PRESENT S
   AVINGS"
40 DISP "ACCOUNT BALANCE";
50 INPUT B
60 DISP
62 DISP "WHAT IS INFLATION RATE
    (%)";
64 INPUT F
66 DISP
70 DISP "WHAT IS S&L INTEREST (
   %)";
80 INPUT I
90 DISP
100 DISP "HOW MANY YEARS WILL SA
   VINGS BE"
110 DISP "HELD";
120 INPUT Y
130 DISP
140 DISP "AFTER";Y;"YEARS YOUR O
   RIGINAL"
150 DISP "$";B;"WILL GROW TO $";
   INT(B*(1+I/100)^Y*100+.5)/10
   0;" "
```

FIGURE 71

Partial listing for "Inflation"

5. The additional, new formula used by the "Inflation" program is shown at the bottom of the flowchart, Figure 70, just above END.

IF YOU NEED HELP with step 6, refer to my complete listing for "Inflation." See page H-22, No. 36.

```
              INFLATION

WHAT IS YOUR PRESENT SAVINGS
ACCOUNT BALANCE?
500

WHAT IS INFLATION RATE (%)?
10.5

WHAT IS S&L INTEREST (%)?
5.25

HOW MANY YEARS WILL SAVINGS BE
HELD?
5
```

Given by lines 140 and 150, Figure 71 }→

```
AFTER 5 YEARS YOUR ORIGINAL
$ 500 WILL GROW TO $ 645.77
```

Write additional statements that will give this kind of output }→

```
AFTER 5 YEARS IT WILL COST
$ 823.72 TO BUY WHAT $ 500
BUYS TODAY.
```

FIGURE 72

PRINTALL output of "Inflation" program

6. Using the "Inflation" flowchart (Figure 70), partial listing (Figure 71) and output (Figure 72), complete your version of "Inflation." To help you write your statements using information on the flowchart (Figure 70), study the relation between the flowchart and listing for "SAVING." The "SAVING" flowchart is Figure 69, and you printed its listing in step 2 above.
7. Run your "Inflation" program using the data shown in Figure 72 (500, 10.5, 5.25, 5), and check your output against Figure 72. You should get $ 645.77 and $ 823.72.
8. Run "Inflation" using any numbers you choose and as often as you like.

REN Command

REN for renumber is a very useful command. It is not programmable. It is used to renumber any program in the HP-85's memory. Pressing (R)(E)(N)(END LINE) numbers the first line 10 and gives an interval of 10 between each line, to give the familiar 10, 20, 30, 40, ... series of line numbers. You can also specify the beginning line number and the interval, if you wish. For instance,

(R)(E)(N)(5)(0)(,)(2)(5)(END LINE)

gives 50 for the first line number, and 75, 100, 125, 150, 175, ... for the rest.

Use REN to renumber your "Inflation" program statements as 10, 20, 30, 40, ... and so on. List "Inflation" to see if the HP-85 understood you.

If you wish to preserve your "Inflation" program, remove your cartridge, move the tab to RECORD, and store "Inflation" using the same name "SAVING" you used before. The old "SAVING" will be lost, but this new version includes all the abilities of the old "SAVING." Don't forget to move your cartridge tab back to the protect position.

"Inflation" is still in the HP-85's memory, so play around with `REN` some more. Renumber your program `1000`, `1050`, `1100`, `1150`, `1200`, ... etc. Press (LIST) again to check.

One way `REN` is useful is during extensive program modifications. Say you wanted to add 15 lines between lines 50 and 60. You could renumber using an interval of 20 and get room for your 15 lines plus 5 extra for possible additions.

I'd like you to do some more experimenting with "Inflation." I've got two new, related BASIC words and commands to give you, and a good way to find out what they do is to use them.

Here they are:

CRT IS 1: A BASIC Word and Command.
CRT IS 2: A BASIC Word and Command.

Now clear your screen and execute the `CRT IS 2` command by pressing

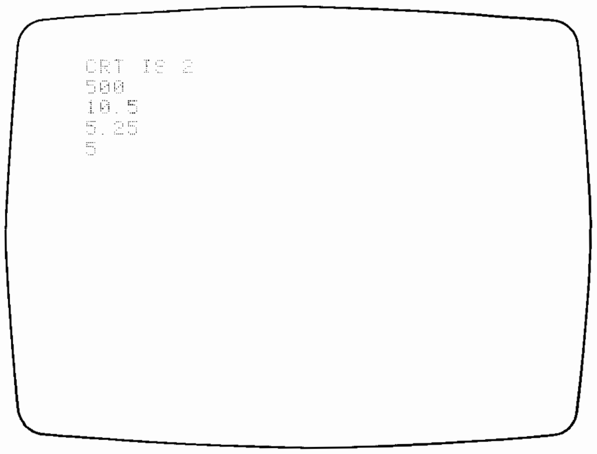
(C) (R) (T) ▲ (I) (S) ▲ (2) (END LINE)

(As usual, the spaces are optional).

Next, run your "Inflation" program and see the first question, including the question mark, printed rather than displayed. The only characters that will be displayed will be those you type in from the keyboard.

Now enter `500` as your present balance, and continue, using a `10.5`% inflation rate, a `5.25`% interest, and `5` years. Figure 73 shows what your printout and screen should look like when you finish.

```
           INFLATION

WHAT IS YOUR PRESENT SAVINGS
ACCOUNT BALANCE?

WHAT IS INFLATION RATE (%)?

WHAT IS S&L INTEREST (%)?

HOW MANY YEARS WILL SAVINGS BE
HELD?

AFTER 5 YEARS YOUR ORIGINAL
$ 500 WILL GROW TO $ 645.77

AFTER 5 YEARS IT WILL COST
$ 823.72 TO BUY WHAT $ 500
BUYS TODAY.
```



```
CRT IS 2
500
10.5
5.25
5
```

FIGURE 73

"Inflation" output after `CRT IS 2` is executed

To cancel CRT IS 2, execute CRT IS 1 by pressing (C)(R)(T)(I)(S)(1)(END LINE) (or (C)(R)(T) ▲ (I)(S) ▲ (1)(END LINE)).

Now, using the same figures, run "Inflation" again, and see the same display as shown in Figure 72.

Here's what's going on. The numbers "1" and "2" used in CRT IS 1 and CRT IS 2 are called "output codes." The code for the screen (CRT) is 1, and the code for the printer is 2. To summarize:

Output Device	Output Code
SCREEN or CRT	1
PRINTER	2

With CRT IS 2 executed, any operation (except typing and editing) that normally puts characters on the screen will, instead, print these characters. Examples of such operations are the execution of DISP statements, the HP-85's execution of error messages, the result of calculations, and the execution of the LIST command. With CRT IS 2 executed, all these operations will print, not display.

There are two other similar BASIC words and commands you should know:

PRINTER IS 1: A BASIC Word and Command

PRINTER IS 2: A BASIC Word and Command

If a program containing PRINT statements were run with PRINTER IS 1 executed, all PRINT statements would output to the screen. Also, if PLIST were executed, the program listing would be displayed, not printed.

When the HP-85 is first turned on, CRT IS 1 and PRINTER IS 2 are automatically in force. Executing either of these commands (or executing statements using these BASIC words) would have no effect on the HP-85's operation. Their only action is to cancel either CRT IS 2 or PRINTER IS 1.

What good are these four commands? A fair question. First of all, *these commands are programmable*. An example of one use for PRINTER IS 1 and PRINTER IS 2, when used as BASIC words in a program, is to cause a PRINT statement to print its message at one point in a program and to display the same message at another point in the program. By using some additional BASIC words you'll meet later, the PRINT statements defining the message need appear only once in the program. When PRINTER IS 1 is executed in the program, the PRINT message will appear on the screen. Then, when PRINTER IS 2 is executed, the same PRINT message will be printed on the paper. You've seen one example of this technique in the way

```
     HEWLETT-PACKARD
GETTING DOWN TO BASIC
```

has appeared on the CRT and on the printer in the same program.

These four commands or BASIC words are summarized below:

Command or BASIC Word	Function
CRT IS 2	Cancels CRT IS 1 All normally displayed characters are printed, except for typed characters.
CRT IS 1	Cancels CRT IS 2 All normally displayed characters are displayed.
PRINTER IS 1	Cancels PRINTER IS 2 All normally printed characters are displayed.
PRINTER IS 2	Cancels PRINTER IS 1 All normally printed characters are printed.

If the HP-85 has recently run a program containing CRT IS 2 or PRINTER IS 1 statements, or if these commands have recently been executed, another program using DISP or PRINT statements can give some strange results when run. Unexpected results can also occur when LIST or PLIST is executed. You saw an example of the strange results CRT IS 2 can produce when you ran your ''Inflation'' program a few paragraphs back.

An easy way to make sure both CRT IS 2 and PRINTER IS 1 are not active is to reset the HP-85 (press SHIFT + RESET). As you recall, pressing RESET puts the HP-85 into its ''wake-up'' condition, except memory is not changed.

I've just mentioned the fact that CRT IS 1, CRT IS 2, PRINTER IS 1, and PRINTER IS 2 are all BASIC words and commands. It turns out that most BASIC words are also commands (and conversely, most commands are also BASIC words). An easy way to find out if a BASIC word is also a command is to use it without a line number. For instance, type PRINT "COMMAND" and press END LINE. Your printer instantly leaps into action. Both DISP and PRINT are commands, to name only two.

Friendly Programming

Here is a programming option you may wish to adopt in some situations. It takes advantage of the fact that *the NORMAL and CLEAR commands are both programmable.* To emphasize these facts:

NORMAL: A BASIC Word and Command

CLEAR: A BASIC Word and Command

As you've seen, users can put the HP-85 in a state somewhat removed from its "wake-up" condition. As a programmer, you can insure that your program will run on an HP-85 which acts as though the user had just pressed RESET before pressing RUN. This set of statements will do it:

`10 CRT IS 1`	(needed only if your program contains `DISP` statements)
`20 PRINTER IS 2`	(needed only if your program contains `PRINT` statements)
`30 NORMAL`	(needed only if your program contains `DISP` statements)
`40 CLEAR`	

Of course, any line numbers will do for these statements, provided they're lower than the line numbers you use for your `DISP` and `PRINT` statements.

Such friendly programming might cause someone using your program to meet frustration. Here's how a user of your friendly program might become frustrated and how such frustration can be conquered.

Say your program starts as follows:

```
10 CRT IS 1
20 PRINTER IS 2
30 NORMAL
40 CLEAR
50 DISP "THIS PROGRAM DESIGNS A
   N AUTHEN- TIC PERPETUAL MOTI
   ON MACHINE."
60 DISP "THIS MACHINE IS AN ENE
   RGY MUL-  TIPLIER. ENERGY OU
   TPUT IS 100"
70 DISP "TIMES GREATER THAN ENE
   RGY INPUT."
 :
```

A user of your program wishes your `DISP` statements to be printed as well as displayed. So she executes the `PRINTALL` command before starting your program. She is frustrated to see the printer remain silent. All your `DISP` statements appear only on the screen. Do you know why? The answer involves your line 30.

```
30 NORMAL
```

As soon as statement 30 is executed, her `PRINTALL` command is cancelled. How does she easily conquer her frustration? She conquers it with a new BASIC word, the symbol `!` or `REM`.

REM: A BASIC Word, or
!: A BASIC Word

Either form of this BASIC word means "remark." A remark is a message that appears only in a program listing.

It is neither displayed, printed nor calculated. It is intended to be read by one who looks at the listing. A remark is used to label a group of related statements, to give the purpose for a group of statements or of an entire program, to give the program's author, to show the program's name, to give credit to sources which inspired the program, or to give any other comment the programmer wishes to make.

The symbol ! allows a remark to be added at the end of an otherwise complete statement. For instance:

```
        135 LET F=W*45-A ! F IS FORCE
```

All characters following the ! symbol are taken by the HP-85 to be a remark. All characters including ! and " may be used in a remark, and all spaces are preserved just as in the quoted text of a PRINT or DISP statement. Here are a few more remarks:

```
        5 REM "PERPETUAL MOTION PROGRAM"
        :
        50 LET A=0 ! THIS PROGRAM
           DESIGNS AN AUTHENTIC PERPET-
           UAL MOTION MACHINE.
        :
        70 ! THE MACHINE'S OUTPUT LIMIT
           IS ONE BILLION MEGAWATTS.
```

Note the use of quotation marks in the remark in statement 5.

Now let's get back to the frustrated user of your program. She can easily preserve the action of her PRINTALL command by changing your NORMAL statement into a remark; that is:

from `30 NORMAL`
to `30!NORMAL`

When she wishes to change back she need only delete the ! symbol from statement 30 and re-enter the statement into the HP-85 (press END LINE).

Note that REM can be used only to *begin* a statement devoted *entirely* to a remark, while ! can be used in two ways:

1. To begin a statement devoted entirely to a remark. (This is the only way REM may be used.)
2. To devote the last part of a line to a remark where the first part is used for any other kind of BASIC statement. An example of this use for ! is:

```
        50 LET A=0 ! THIS PROGRAM
           DESIGNS AN AUTHENTIC PERPET-
           UAL MOTION MACHINE.
```

EXAMPLE: " 'Mole Mitt' Morrison" Program

"Mole Mitt" Morrison, a strip mining engineer, is one of those fortunate people whose work is his hobby. He loves to dig. His new home in Suburbia is his pride and joy, but it lacks one feature strongly favored by his family and peculiarly suited to his own skill. It lacks a swimming pool. Mole Mitt lacks something else: money. "Who," Mole Mitt asks, "is better equipped than I to dig my pool, thereby saving a bundle? No one," he answers. So, trusty shovel in hand, Mole Mitt begins.

Let's see just how big a hole Mole Mitt is digging for himself. How long will it take him to dig his pool?

Program Description Each shovelful of dirt will have certain dimensions, it will take a certain average time to dig one shovelful of dirt, and the finished pool will have certain dimensions. Dividing the pool volume by the volume of dirt removed by each shovelful will give the number of shovelfuls required to dig the pool. Multiplying the average time to dig one shovelful by the required number of shovelfuls will give the total digging time required. Certainly Mole Mitt will not dig 24 hours a day, 7 days a week. His schedule will be 8 hours a day, 5 days a week, with 5 weeks off a year for holidays, vacation and sickness. Combining this information, we get the number of hours available per year for digging. Finally, dividing the total digging time required by the digging time available per year gives the number of years, or fraction of a year, this money saving project will take.

Important Formulas

I use the `INT()` function in some of these formulas to avoid the display of 12 digit numbers.

1. V = W * L * D

   Where V = volume of pool in cubic yards

   W = width of pool in yards

   L = length of pool in yards

   D = average depth of pool in yards

2. S1 = INT (46656/(A * B * C))

   Where S1 = shovelfuls of dirt in one cubic yard

   A = width of shovelful of dirt in inches

   B = length of shovelful of dirt in inches

   C = average depth of shovelful of dirt in inches

   Note: There are 46,656 cubic inches in one cubic yard.

3. S2 = V * S1

   Where S2 = shovelfuls of dirt needed to excavate pool

4. T2 = T1 * S2

   Where T2 = seconds needed to excavate pool

   T1 = seconds needed to shovel one shovelful of dirt, including time for coffee breaks, rest periods, wheelbarrow time, etc.

5. T3 = INT (T2/3600)

   Where T3 = hours needed to excavate pool

6. T4 = INT (T3/8)

   Where T4 = days needed to excavate pool

7. T5 = INT ((T4/5/47 * 10) + .5) / 10

   Where T5 = years needed to excavate pool

Answers to Program Planning Questions At the beginning of Chapter 9, on page 134, you were introduced to the four program planning questions. You saw how these questions might be answered by one planning the "SAVING" program. Now take a look at the way these four questions might be answered by a writer of the "Mole Mitt" program.

1. What answers do I want?

   I want to find out how long it will take Mole Mitt to dig his pool. I want this answer in seconds, hours, days and years. I want this answer for a variety of pool sizes, various dimensions for a shovelful of dirt, and different intervals of time needed to shovel one shovelful of dirt.

2. What things do I know?

   I know the various formulas shown above.

3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?

   I'll choose pool dimensions and shovelful of dirt dimensions, as well as the time interval needed to shovel one shovelful of dirt. Then I'll solve the formulas.

4. How can BASIC and the HP-85 help me find answers?

   `DISP` and `INPUT` statements will be useful. I'll have the HP-85 display a request for the pool dimensions in yards, W, L and D, then the program will calculate and display V, the pool's volume in cubic yards. Next, the HP-85 will display a request for the shovelful of dirt dimensions in inches, A, B, and C, and the number of shovelfuls in one cubic yard, S1, will be calculated. Following this, the total number of shovelfuls, S2, will be calculated and displayed. Now the HP-85 will display a request for T1, the number of seconds needed for one shovelful. Finally, T2, T3, T4 and T5, the seconds, hours, days and years needed to excavate the pool, will be calculated and displayed.

Construction of Flowchart

As you can see, the construction of the flowchart, Figure 74, follows directly from the answer to programming question 4. Compare this answer, sentence by sentence, with each element of the flowchart.

Listing for "Mole Mitt"

Figure 75 shows the listing for "Mole Mitt." Enter this program and run it.

If you want a suggestion, you might try a 100 by 50 by 2 yard pool (think big!), an 8 by 8 by 4 inch shovelful of dirt, and a 30 second time interval to shovel one shovelful. Check Figure 75 to see how long it would take poor Mole Mitt to dig such a pool.

Dimensions of pool in yards

W = Width
L = Length
D = Average depth
V = Pool volume in cubic yards

Dimensions of shovelful of dirt in inches

A = Width
B = Length
C = Average depth

T1 = Seconds needed to shovel one shovelful of dirt

S1 = Shovelfuls of dirt in one cubic yard

S2 = Shovelfuls of dirt needed to excavate pool

Time needed to excavate pool:

T2 : In seconds
T3 : In hours
T4 : In days
T5 : In years

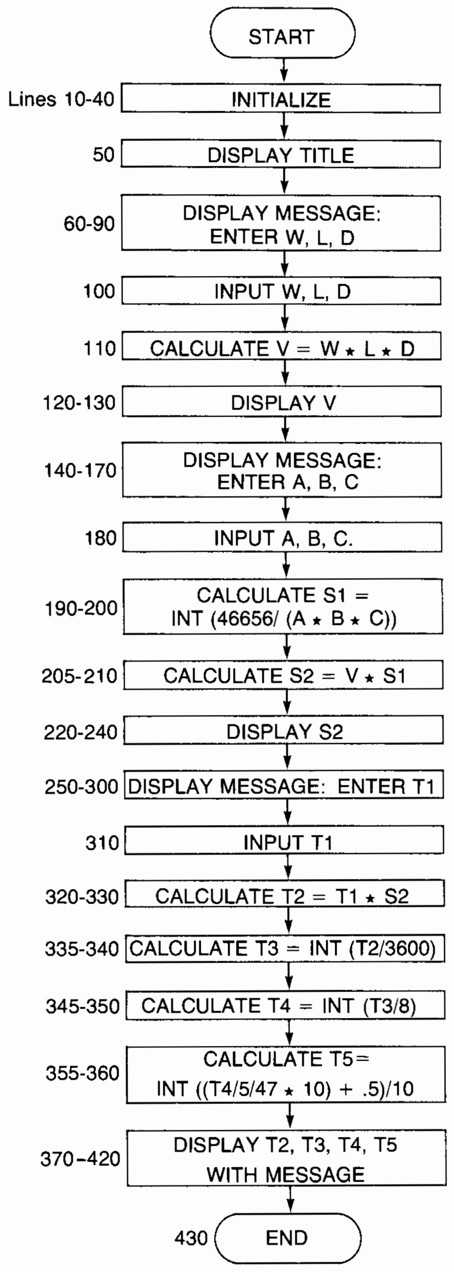


FIGURE 74
Flowchart for "Mole Mitt"

```
10 ! "MOLE" PROGRAM
20 CRT IS 1
30 NORMAL
40 CLEAR
50 DISP "            MOLE"
60 DISP
70 DISP "ENTER WIDTH, LENGTH AN
   D AVERAGE"
80 DISP "DEPTH OF SWIMMING POOL
    IN YARDS."
90 DISP "SEPARATE NUMBERS BY CO
   MMAS."
100 INPUT W,L,D
110 LET V=W*L*D
120 DISP
130 DISP "POOL VOLUME IS";V
140 DISP
150 DISP "ENTER THE WIDTH, LENGT
    H AND"
160 DISP "AVERAGE HEIGHT OF A SH
    OVELFUL"
170 DISP "OF DIRT IN INCHES."
180 INPUT A,B,C
190 DISP
195 ! S1=SHOVELFULS OF DIRT
          IN ONE CUBIC YARD.
200 LET S1=INT(46656/(A*B*C))
205 ! S2=SHOVELFULS OF DIRT
          NEEDED TO EXCAVATE
          POOL.
210 LET S2=V*S1
220 DISP "THE NUMBER OF SHOVELFU
    LS OF DIRT"
230 DISP "NEEDED TO EXCAVATE THE
     POOL IS"
240 DISP S2;"."
250 DISP
260 DISP "ENTER TIME IN SECONDS
    REQUIRED"
270 DISP "TO SHOVEL ONE SHOVELFU
    L OF DIRT."
280 DISP "INCLUDE ENOUGH TIME TO
     COVER"
290 DISP "COFFEE BREAKS, WHEELBA
    RROW WORK,"
300 DISP "ETC."
310 INPUT T1
320 DISP
325 ! T2=SECONDS NEEDED TO
          EXCAVATE POOL.
330 LET T2=T1*S2
335 ! T3=HOURS NEEDED TO
          EXCAVATE POOL.
340 LET T3=INT(T2/3600)
345 ! T4=DAYS NEEDED TO
          EXCAVATE POOL.
350 LET T4=INT(T3/8)
355 ! T5=YEARS NEEDED TO
          EXCAVATE POOL.
360 LET T5=INT(T4/5/47*10+.5)/10
370 DISP "TIME NEEDED TO EXCAVAT
    E POOL:"
380 DISP
390 DISP "   SECONDS:   ";T2
400 DISP "   HOURS:     ";T3
410 DISP "   DAYS:      ";T4
420 DISP "   YEARS:     ";T5
430 END
```

```
TIME NEEDED TO EXCAVATE POOL:

   SECONDS:    54600000
   HOURS:      15166
   DAYS:       1895
   YEARS:      8.1
```

FIGURE 75

Listing for "Mole Mitt" and length of time it would take Mole Mitt to complete his pool project.

Comments on Program Planning and Writing

If you're like most people, you will not construct your final flowchart on your first day. Even after you have finally answered your questions to your satisfaction and have drawn an elegant flowchart, you'll probably go back and change both once or several times after you begin writing your statements. On the other hand, you may find it unneccessary to answer the programming questions or construct a flowchart, at least for some programs. Please use the program writing system that works best for you.

As the job gets more difficult, and as the desire for perfection gets stronger, the more trial and error, scrapping, and starting over will occur.

Tape Cartridge Catalog—the `CAT` command

You'll soon write a program to help Mole Mitt's neighbor with his pool project. But first, learn more about HP-85 tape cartridges. You can easily see the contents, called the directory or catalog, of any HP-85 cartridge by executing the `CAT` command. To print the catalog for your BASIC Training cartridge, first make sure your cartridge is inserted into the HP-85. Since the `CAT` command normally displays rather than prints the catalog, execute now the `PRINTALL` or `CRT IS 2` command so that your catalog will be printed when you execute `CAT`. Finally, execute `CAT` by pressing (C)(A)(T)(END LINE).

Your cartridge will run as the HP-85 seeks the catalog at the front of the tape. Then the catalog will be printed. If you executed `PRINTALL` rather than `CRT IS 2`, your catalog will also be displayed. It should look like Figure 76.

Notice your `MONEY` and `SAVING` programs in the left hand `NAME` column.

The next column, `TYPE`, shows that all but one file is a program file. The single data file contains the information used by the `KEYS`, `KEYS2` and `KEYS3` programs to draw the HP-85's keyboard on the screen.

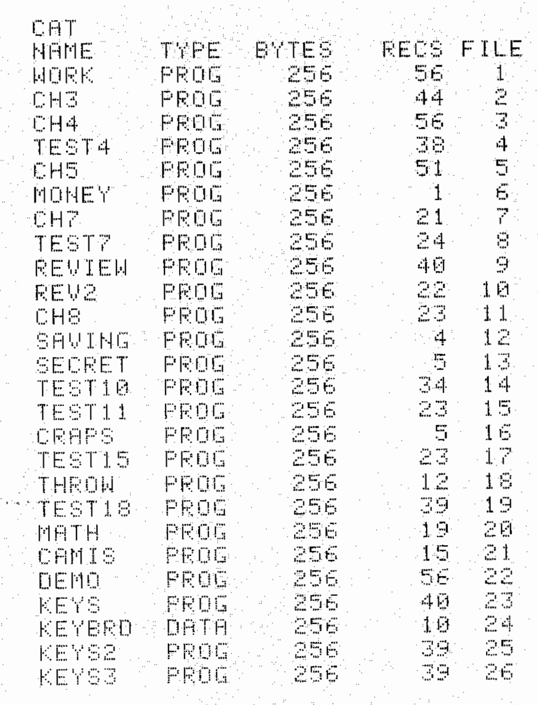

```
CAT
NAME     TYPE   BYTES   RECS FILE
WORK     PROG     256     56    1
CH3      PROG     256     44    2
CH4      PROG     256     56    3
TEST4    PROG     256     38    4
CH5      PROG     256     51    5
MONEY    PROG     256      1    6
CH7      PROG     256     21    7
TEST7    PROG     256     24    8
REVIEW   PROG     256     40    9
REV2     PROG     256     22   10
CH8      PROG     256     23   11
SAVING   PROG     256      4   12
SECRET   PROG     256      5   13
TEST10   PROG     256     34   14
TEST11   PROG     256     23   15
CRAPS    PROG     256      5   16
TEST15   PROG     256     23   17
THROW    PROG     256     12   18
TEST18   PROG     256     39   19
MATH     PROG     256     19   20
CAMIS    PROG     256     15   21
DEMO     PROG     256     56   22
KEYS     PROG     256     40   23
KEYBRD   DATA     256     10   24
KEYS2    PROG     256     39   25
KEYS3    PROG     256     39   26
```

FIGURE 76

Catalog of BASIC Training tape cartridge

These three program files and one data file belong to the third major part of the BASIC Training Application Pac, The HP-85 Keyboard. It is introduced on page 316 in this workbook.

The `DEMO` program, cataloged just ahead of `KEYS`, is associated with the second major part of this pac, An HP-85 Demonstration. See page 314 for its introduction.

All the remaining programs are part of this Getting Down to BASIC course.

The third column on your catalog, `BYTES`, and the fourth column, `RECS` (short for records) are closely connected. Your Owner's Manual explains records on pages 180 and 181. For this course, it's enough to say that the approximate length of a program on this cartridge, in bytes, is given by multiplying the number in the `BYTES` column, 256, by the number in the `RECS` column. You'll find that no program on this cartridge uses more than 14,576 bytes, since a larger program would not fit in the memory a standard HP-85 has available for a user. For instance, `CH4` has 56 records, or $56 \times 256 = 14{,}336$ bytes. Actually, a little less than 14,336, since the 56th record is not full.

The final column, `FILE`, simply gives an identifying number to each file.

Tape Cartridge Capacity

You'll be writing a number of programs as you continue this course. Since you may wish to store your completed programs for later use, you might like to know how many more programs you can store on this cartridge.

An HP-85 cartridge can hold no more than 42 files and no more than about 800 records (204,800 bytes). I can't give you an exact number of records or bytes since the actual number of centimeters of tape used by one record depends on the material contained in the record. Also, the length of tape contained in a cartridge varies somewhat from cartridge to cartridge. When your BASIC Training cartridge left our factory, it contained 739 records and 26 files. So you have about 60 records or 16 files left. As you store your programs, chances are you'll run out of records before you use all 42 available file numbers. In any event, don't worry about exceeding the capacity of your cartridge. If you try to store a program that tries to use unavailable records, this friendly error message will be displayed:

```
Error 65 : END OF TAPE
```

If you try to store file 43, you'll see this message:

```
Error 61 : >42 FILES
```

In either case, your program would remain unharmed in the HP-85's memory.

At some time you might wish to store a program on another cartridge, especially if your BASIC Training cartridge is full. If this other cartridge is brand new, you'll have to prepare it to accept programs. This is called "initialization," and is explained on page 175 in your Owner's Manual under Directory Set-Up.

CAUTION: The cartridge initialization procedure erases all files from the catalog of a used cartridge, making all previously recorded files unavailable.

Any particular program name can be used only once on any one cartridge. For instance, your BASIC Training cartridge cannot accept two programs named "MONEY." If you attempted to store a second "MONEY" program, that new program would be stored, but your Chapter 5 "MONEY" program would be lost. So inspect your catalog before storing a finished program to make sure the program name you intend to use has not been used already.

The Work File

The first program on your cartridge is named "WORK." It is a workspace provided as a temporary storage place for any unfinished program you might be working on. If you wish to store an unfinished version of a program, execute `STORE"WORK"` and the partial program will be stored.

When you store an unfinished program, don't be surprised to hear a beep, and to see an error message on your screen after your partial program is stored and the tape drive stops. The HP-85 is simply telling you that there is something wrong with your program. Not exactly overwhelming news, since you knew the program was unfinished when you stored it. Don't let the beep and message (or beeps and messages) concern you. Your partial program will be safely stored.

To get your program back into the HP-85 for further writing, execute `LOAD"WORK"`. No matter what program you're working on, to store it temporarily, always use the name "WORK." Each time you store an unfinished program using the name "WORK," you'll erase the previous contents of the WORK file. So don't use the name "WORK" to store your final version of a program you wish to keep.

A Word About Example and Problem Programs

You'll be seeing many programs described during this course that I'll offer as examples, and many more that I'll ask you to write. Your tape cartridge does *not* contain most of these programs. Rather, listings of these programs will be in this workbook and in the Supplement.

However, a few of these programs are on your cartridge. I'll tell you clearly when to load such an example program, or when to load my version of a program I've asked you to write. If you try to load a program that is not on the cartridge, you'll see this error message displayed:

```
        Error 67 : FILE NAME
```

If you see this error message, you'll know the program you've asked for is not on the cartridge.

PROBLEM: Write the " 'Pail-Face' Trudgebottom" Program

Right next door to Mole Mitt Morrison lives "Pail-Face" Trudgebottom. If Mole Mitt was born with a shovel in his hand, Pail-Face greeted the world with a water pail clutched in his tiny right fist.

Pail-Face has learned of your analysis of Mole Mitt's swimming pool project, and asks you to analyze a pool project of his own. While Mole Mitt decided to save money by excavating his pool with his trusty shovel, Pail-Face, having no hose, plans to save money on his pool by filling it using his faithful pail.

Pail-Face's pool is the same size as Mole Mitt's, and it's further along. In fact, his will be ready for water at the same time Mole Mitt's excavation is scheduled to start. Pail-Face wants to know if he should bet his neighbor that his pool will be filled with water before Mole Mitt finishes his dig. Trudgebottom has asked you to calculate how long it would take him to fill his pool using his pail.

Program Description Trudgebottom's pail has a certain effective capacity, perhaps 2 gallons. It will take a certain average time to fill, transport and empty one pailful of water. The finished pool will have certain dimensions. Dividing the pool capacity by the pail capacity will give the number of pailfuls of water needed to fill the pool. Multiplying the average time to empty one pailful by the required number of pailfuls will give the total pail filling and emptying time required. Trudgebottom's work schedule is the same as Morrison's, 8 hours a day, 5 days a week, with 5 weeks off a year for holidays, vacation and sickness. Combining this information, we get the number of hours available per year for pail work. Finally, dividing the total pail time required by the pail time available per year gives the number of years, or fraction of a year, this thrifty project will take.

Important Formulas

1. V = W * L * D

   Where V = volume of pool in cubic yards

   W = width of pool in yards

   L = length of pool in yards

   D = average depth of pool in yards

2. P2 = V * 201.97/P

   Where P2 = pailfuls of water needed to fill pool

   V = volume of pool in cubic yards

   P = usable volume of pail in gallons

   Note: There are 201.97 gallons in one cubic yard.

3. T2 = T1 * P2

   Where T2 = minutes needed to fill pool

   T1 = minutes needed for one pail filling and emptying round trip

4. T3 = INT(T2/60)

   Where T3 = hours needed to fill pool

5. T4 = INT(T3/8)

   Where T4 = days needed to fill pool

6. T5 = INT((T4/5/47 * 10) + .5)/10

   Where T5 = years needed to fill pool

Your Turn Try your hand at answering the four programming questions:

1. What answers do I want?
2. What things do I know?
3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?
4. How can BASIC and the HP-85 help me find answers?

To see one way the program planning questions can be answered, see page H-23, No. 37 .

When you're satisfied with your answers, draw your flowchart.

My flowchart for this program appears on page H-24.

Finally, write, run and list on your printer a program that will help Pail-Face decide whether or not to suggest a little wager to Mole Mitt. Your program should offer the same kind of choices as ''Mole'' does. That is, you should have three `INPUT` statements. Also, put some suitable remarks into your program. In fact, your program should be rather similar to ''Mole,'' although a little simpler, since there's one less formula. Good luck.

If you wish to store your unfinished program and continue your programming several hours or days later, insert your BASIC Training cartridge, prepare to ignore beeps and error messages (you know your program is not finished), and execute `STORE"WORK"` (STORE " W O R K " END LINE).

When you wish to resume programming, execute `LOAD"WORK"` and go to it.

The listing of my version of ''Pail'' is on pages H-23 and H-24.

Goodby LET

The BASIC word `LET` need not appear in assignment statements. `LET` will be missing from all programs I'll show you from now on, and chances are most BASIC programs you'll see elsewhere will omit `LET` as well.

Remember, whenever you see [statement no.][variable name] = ... you're looking at an assignment statement. Like:

```
 50  I=5000
  :
 90  I=.9*I
  :
140  C=(INT(N)+1)*449999+100000
```

The expression on the right of the = symbol always represents a single number, no matter how complicated it looks.

Summary of Chapter 9

- **Program planning questions**

  1. What answers do I want?
  2. What things do I know?
  3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?
  4. How can BASIC and the HP-85 help me find answers?

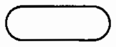
  Reference: Pages 9-134–9-135.

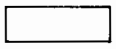
- **Flowcharts help to:**

  - Plan program
  - Change program plan
  - Explain program to others and later to yourself.

  Reference: Pages 9-135–9-136.

- **Flowcharting symbols**

  Start of program or END statement 

  Task 

  Reference: Page 9-135.

- REN: **A command**

  Renumbers statements of a program. To renumber starting at 10 and using an interval of 10, press

  R E N END LINE

  To renumber starting at any number and using any interval, press

  R E N [starting number] , [interval] END LINE

  Example: To start numbering at 100 and using an interval of 20, press

  R E N 1 0 0 , 2 0 END LINE

  Reference: Pages 9-140–9-141.

- **Output device codes**

Output Device	Output Code
Screen (CRT)	1
Printer	2

Reference: Pages 9-141–9-142.

- **Four BASIC words**

BASIC Word	Function
CRT IS 2	Cancels CRT IS 1 All normally displayed characters are printed, except for typed characters.
CRT IS 1	Cancels CRT IS 2 All normally displayed characters are displayed.
PRINTER IS 1	Cancels PRINTER IS 2 All normally printed characters are displayed.
PRINTER IS 2	Cancels PRINTER IS 1 All normally printed characters are printed.

Reference: Pages 9-141–9-143.

- **To easily cancel** CRT IS 2 **and** PRINTER IS 1**, reset the HP-85** (press SHIFT + RESET). Reference: Page 9-143.

- **Most BASIC words are also commands and most commands are also BASIC words.** Reference: Page 9-143.

- NORMAL: **A BASIC word**

  Cancels PRINTALL. Also cancels some other commands covered in later chapters.

  Reference: Pages 9-143–9-144.

- CLEAR: **A BASIC word**

  Clears the screen.

  Reference: Pages 9-143–9-144.

- **Friendly programming**

The following set of statements, when executed, cancels commands which might otherwise cause the program to execute in undesirable ways. The statement numbers used and the order of these statements are not critical, except they should execute before any `DISP` or `PRINT` statement.

`10 CRT IS 1`	(needed only if your program contains `DISP` statements)
`20 PRINTER IS 2`	(needed only if your program contains `PRINT` statements)
`30 NORMAL`	(needed only if your program contains `DISP` statements)
`40 CLEAR`	

Reference: Pages 9-143–9-145.

- `REM` **or** `!`**: A BASIC word**

Either form means "remark." A remark is a message that appears only in a program listing. It is neither displayed, printed nor calculated. It is a message for a reader of the program listing.

Reference: Pages 9-144–9-145.

- **To temporarily remove a line from a program, use** `!`. To make the line active again, simply delete the `!` and reenter the statement. Example:

`30 NORMAL` (active)
`30!NORMAL` (inactive)
`30 NORMAL` (active)

Reference: Pages 9-144–9-145.

- `CAT`: **A command**

Pressing $\boxed{C}\boxed{A}\boxed{T}\boxed{\text{END LINE}}$ displays the catalog of the tape cartridge currently inserted into the HP-85. Reference: Page 9-150.

- **Tape cartridge catalog**

A list of all active programs and other files on a tape cartridge. It is recorded on all active HP-85 cartridges, and is displayed with the `CAT` command. Reference: Pages 9-150–9-152.

- `LET` **is not required for an assignment statement.** What is required to assign a number to a variable is:

[statement no.] [variable name] = [expression that always reduces to a single number]

Example:

```
250 N3=(Y+A2)/17
```

Reference: Page 9-154.

Review Test for Chapter 9

Answer all questions, then check your answers against mine on page 9-159.

1. A program is first renumbered by pressing R E N END LINE. This causes one line of the program to look like this:

   ```
   30 DISP "PROBLEM 1"
   ```

   Now the program is renumbered again by pressing R E N 5 0 , 2 5 END LINE. What line number will this same statement have after this second renumbering?

2. These commands are executed, one after the other·

   ```
   CRT IS 2
   PRINTER IS 1
   CRT IS 1
   ```

   Now a program is run containing this statement.

   ```
   55 DISP "WHAT'S HAPPENING?"
   ```

   Will this message be

   a. displayed,

   b. printed, or

   c. both displayed and printed?

3. These commands are executed, one after the other:

   ```
   CRT IS 2
   PRINTER IS 1
   PRINTALL
   ```

   and then RESET is pressed.

   Now a program containing these two lines is run:

   ```
   120 PRINT "ENOUGH"
   125 DISP "ALREADY"
   ```

   a. On what output device or devices will `ENOUGH` appear?

   b. On what output device or devices will `ALREADY` appear?

4. The HP-85 is operating in a ‘‘just-turned-on’’ condition. A program is run containing this statement:

   ```
   45 DISP "I HOPE THIS IS THE END
   OF THE   TEST."!"!!!"!!
   ```

   How many exclamation marks will be displayed when statement 45 is executed?

5. What kind of statement is this?

   ```
   67 X=1
   ```

Answers to Review Test Questions for Chapter 9

1. The statement number will be 100, as shown below:

Line numbers after the first renumbering	Line numbers after the second renumbering
10 — — — — — —	50
20 — — — — — —	75
→30 — — — — — —	100←
40 — — — — — —	125
50 — — — — — —	150
⋮	⋮

Reference: Page 9-140.

2. a. displayed

The final command, `CRT IS 1`, cancels the earlier command, `CRT IS 2`, and causes all normally displayed characters to be displayed. So all `DISP` statements will be displayed on the screen.

Reference: Pages 9-141–9-143.

3. a. printer

b. screen or CRT

Pressing RESET (SHIFT + RESET), puts the HP-85 into a "just-turned-on" condition. So `PRINT` statements are printed and `DISP` statements are displayed.

Reference: Pages 9-141–9-143.

4. No exclamation marks will be displayed. All characters following the first `!` are part of the remark. All characters in a remark are neither displayed, printed, nor calculated. Reference: Pages 9-144–9-145.

5. Assignment statement. Reference: Page 9-154.

Chapter 10

Tell Your Program Where It Can Go

Preview

In Chapter 10, you will:

- Learn how to tell your program to wait as long as you wish, then have it continue automatically.
- Learn how to tell the HP-85 to beep.
- Learn how to tell your program where it can go.
- Learn how to order the HP-85 to number your program statements for you.

You've survived a couple of long, tough chapters, so I'm giving you a short one as a reward.

WAIT: A BASIC Word

This is an easy one. For example:

```
20 WAIT 100
```

means wait 100 milliseconds, or 0.1 second. Here are some more examples:

```
125 WAIT 1000
```

means wait 1 second.

```
50 WAIT 10000
```

means wait 10 seconds.

```
290 WAIT T
```

means wait T milliseconds, where T's value is assigned to it somewhere else in the program.

You'll see `WAIT` in action soon.

BEEP: A BASIC Word

This BASIC word is a versatile helper. With an argument (numbers following `BEEP`) defining pitch and duration, `BEEP` can be used to compose a tune. For this course, `BEEP` will be used without an argument to keep things simple.

`BEEP` will join `WAIT` later in a program.

GOTO: A BASIC Word

Here is one of the big guns of BASIC. A line number always follows GOTO, like this:

```
25 GOTO 100
```

This means line 100 will be the next one executed in this program after line 25.

You may type GO TO or GOTO.

AUTO: A Command

I'd like you to use a new time-saving command, AUTO, to enter the 11 statements of "Beep," an example program using your three new BASIC words, WAIT, BEEP, and GOTO. After a little work with AUTO, you'll enter and run "Beep."

Clear your screen and press (A) (U) (T) (O) (END LINE). See the statement number 10 appear automatically at the left margin of the next line. Now press (END LINE) and see the line number 20 appear. Press (END LINE) again and see 30. Now, don't press any more keys until you enter "Beep" a few paragraphs ahead. Just read until I tell you to use your fingers.

Using AUTO with no argument; that is, with no numbers typed in before (END LINE) is pressed; automatically numbers your program lines starting at 10 and advancing 10 every time (END LINE) is pressed, giving the usual 10, 20, 30, 40, 50, ... series of line numbers.

Once you have your automatic line number generator going, how do you stop it? Simple. Press (SHIFT) + (BACK SPACE) to erase the last AUTO generated line number, then use the NORMAL command (type the word NORMAL, then press (END LINE)). You'll use this method when you enter the "Beep" program a little later.

Here are two other ways to cancel AUTO:

a. Say you wanted to display a listing of the program or program segment you have just entered using AUTO. Pressing (LIST) will list your program and also cancel AUTO.

b. Say you wanted to run the program you have just entered. Press (RUN). You program will run (hopefully), and AUTO will be cancelled.

See Figure 77 for the location of the (AUTO ↓) typing aid.

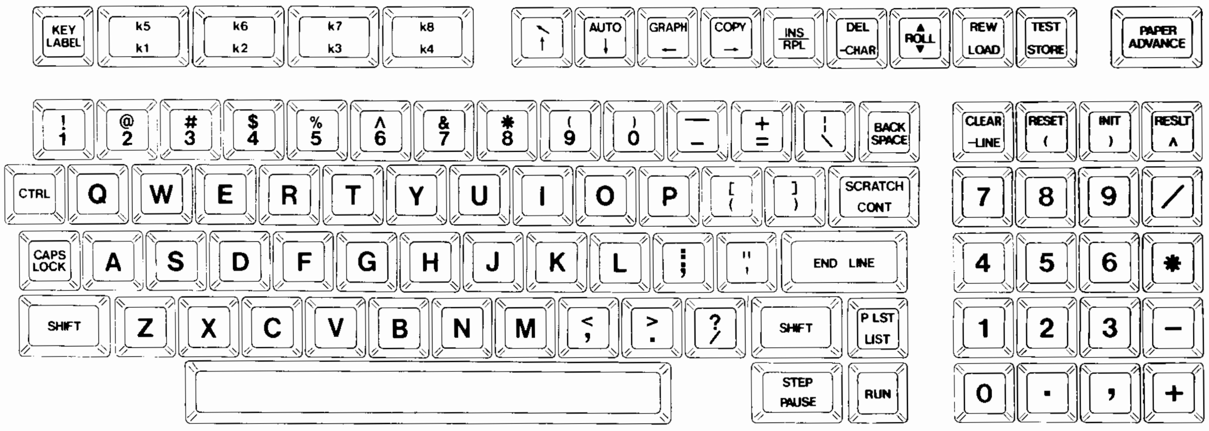


FIGURE 77

Location of (AUTO)

This typing aid allows the AUTO command to be executed by pressing

[SHIFT] + [AUTO ↓] [END LINE]

Say you wanted to start numbering at 100, and advance 5 numbers for each line. You'd press

[AUTO] [1] [0] [0] [,] [5] [END LINE]

Figure 78 shows several sets of line numbers generated by AUTO. After 1300 was displayed, [END LINE] was pressed, and the next line number, 1400, was displayed. To cancel the AUTO command (and to erase 1400), [SHIFT] + [BACK SPACE] was pressed, then the NORMAL command was executed.

```
AUTO
10
20
30
40
50
AUTO100,20
100
120
140
160
180
AUTO1000,100
1000
1100
1200
1300
NORMAL
```

FIGURE 78

PRINTALL of line number generation using AUTO

EXAMPLE: "Beep" Program

To fix AUTO in your mind and to enter "Beep" at the same time, follow these keystrokes and instructions.

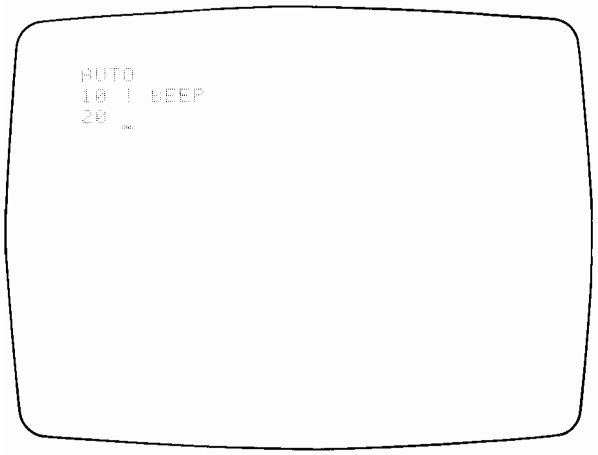
a. First, scratch memory.

b. Next, clear your screen.

c. Press [SHIFT] + [AUTO ↓] [END LINE] and see 10 displayed on the next line.

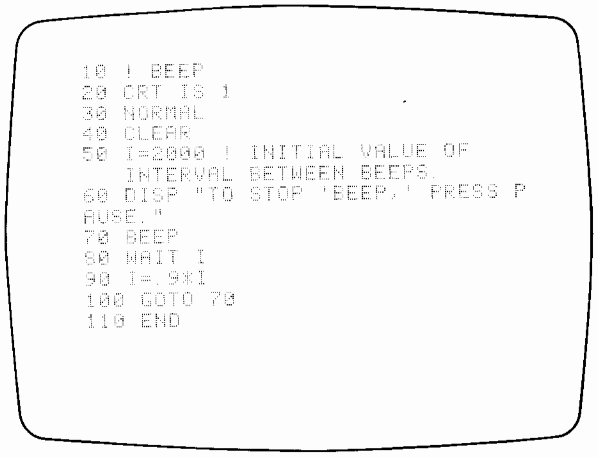
d. Press [ ] ▲ [B] [E] [E] [P] [END LINE] and see 20 displayed on the next line.

e. Now your screen should look like this:



```
AUTO
10 ! BEEP
20 _
```

f. The "Beep" program CRT listing is shown below:

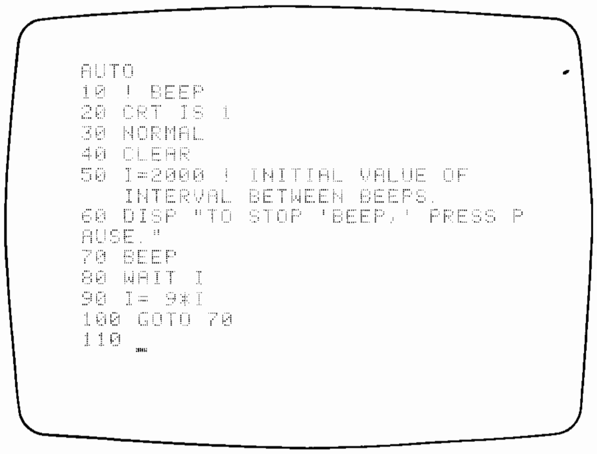


```
10 ! BEEP
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I=.9*I
100 GOTO 70
110 END
```

Listing of "Beep"

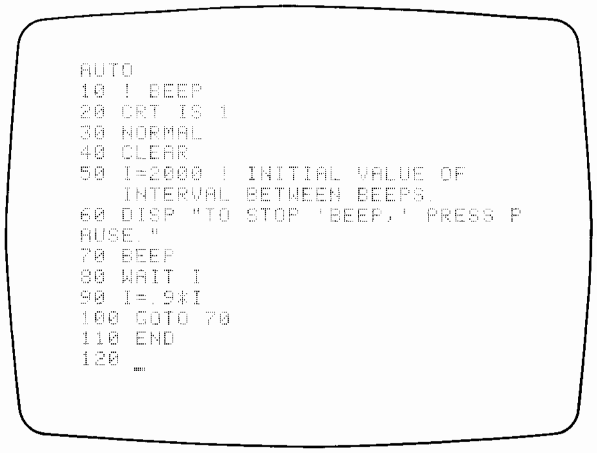
g. Now enter statements 20 through 100 inclusive, letting the HP-85 display your line numbers for you.

h. Now your screen should look like this:



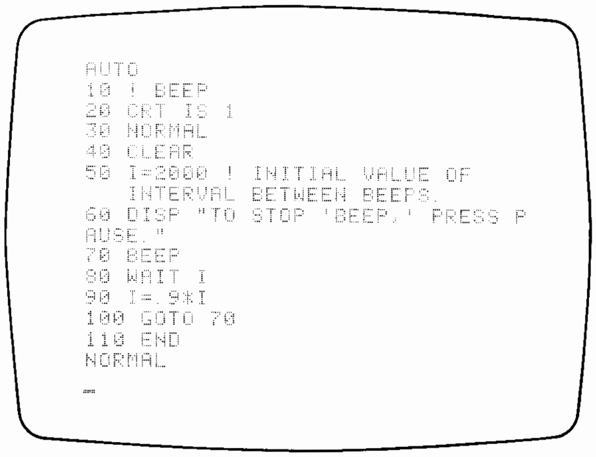
```
AUTO
10 ! BEEP
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I=.9*I
100 GOTO 70
110 _
```

i. Now press (E) (N) (D) (END LINE) and see:



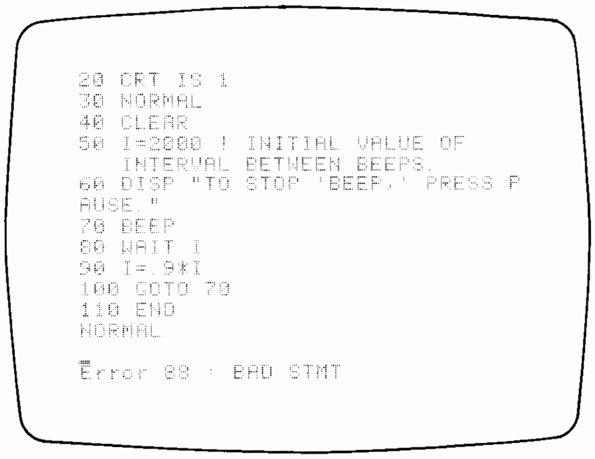
```
AUTO
10 ! BEEP
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I=.9*I
100 GOTO 70
110 END
120 _
```

j. To cancel the AUTO command, press SHIFT + BACK SPACE, use the NORMAL command, and see:



```
AUTO
10 ! BEEP
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I=.9*I
100 GOTO 70
110 END
NORMAL
```

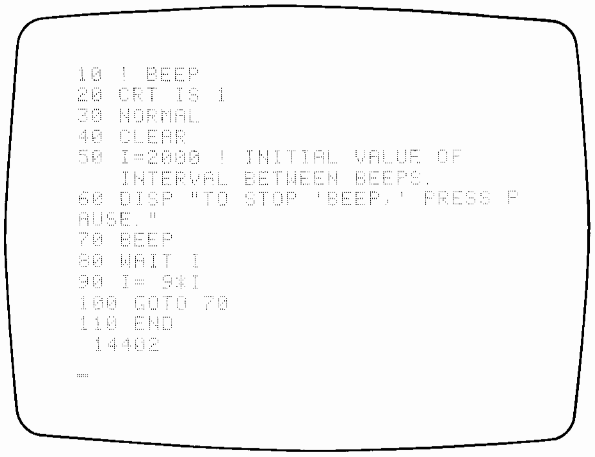
k. To confirm that AUTO has been cancelled, try to generate line number 120 by pressing END LINE. Hear the beep and see:



```
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I=.9*I
100 GOTO 70
110 END
NORMAL

Error 88 : BAD STMT
```

Don't panic. Remember that an error message returns control of the HP-85 back to you. You are still in calculator mode, and the keyboard is live.

1. To confirm that "Beep" has been successfully entered, clear your screen and press (LIST). You should see:



```
10 ! BEEP
20 CRT IS 1
30 NORMAL
40 CLEAR
50 I=2000 ! INITIAL VALUE OF
   INTERVAL BETWEEN BEEPS.
60 DISP "TO STOP 'BEEP,' PRESS P
AUSE."
70 BEEP
80 WAIT I
90 I= 9*I
100 GOTO 70
110 END
 14402
```

A flowchart of "Beep" is shown in Figure 79. Notice that the `GOTO` statement is replaced in the flowchart by a line which traces the flow of the program. This program has an `END` statement, as all BASIC programs should, but when the "Beep" program is executed, `END` is never reached. The program goes into an endless loop. Fortunately, pressing (PAUSE) stops the beeping.

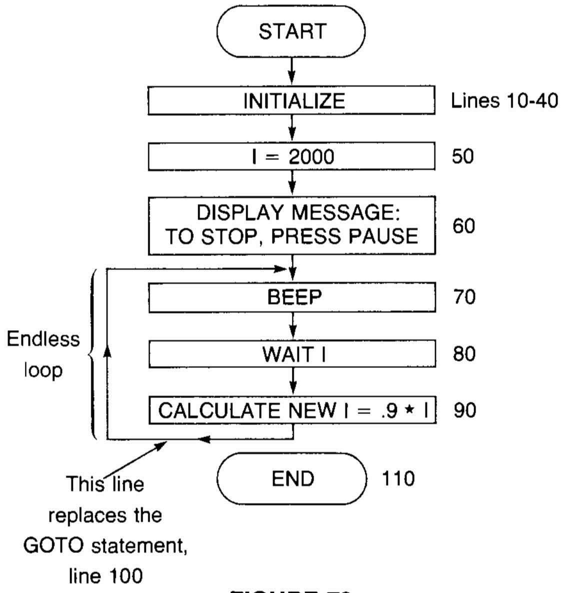


FIGURE 79

Flowchart for "Beep"

These four statements

```
 70 BEEP
 80 WAIT I
 90 I=.9*I
100 GOTO 70
```

are executed over and over. Each time they're executed, the time interval I gets smaller and smaller (10% smaller each time) until it becomes small compared to the execution time of these four statements. At this point, a further decrease in I is not audible. The small delay between beeps is due primarily to the execution time of these statements.

Run "Beep" if you wish.

PROBLEM: Work with the "SECRET" Program

Here's an exercise that will make `GOTO` a more familiar BASIC word. Load "SECRET" from your BASIC Training cartridge. When you run it, you'll see the program print a partial listing of itself. This is possible because `PLIST` (and `LIST`) are programmable. Now run "SECRET" and trace the flow of the program portion you see listed. Rewrite it omitting all `GOTO` statements. That is, end up with a new program consisting primarily of `PRINT` statements that print the secret message when the new program is run.

Hint Not all the `PRINT` statements in "SECRET" are part of the secret message.

Another challenge: Find an easy way to print the message without rewriting the program.

To see the answers, go to page H-25.

Summary of Chapter 10

- `WAIT`: **A BASIC word**

  General form:

  [line number] `WAIT` [no. of milliseconds]

  A `WAIT` statement instructs the program to halt execution for the specified number of milliseconds, then continue execution automatically with the next statement.

  Example:

  ```
  315 WAIT 1000
  ```

  When this statement is executed, the program halts for 1000 milliseconds (one second), then continues execution automatically.

  Reference: Page 10-160.

- `BEEP`: **A BASIC word**

  General form:

  [line number] `BEEP`

  A `BEEP` statement instructs the HP-85 to sound its beep.

  Example:

  ```
  330 BEEP
  ```

  Reference: Page 10-160.

■ GOTO: **A BASIC word**

General form:

[line number] GOTO [line number]

A GOTO statement instructs the program to execute next the statement whose line number follows GOTO.

Example:

```
128 GOTO 517
```

When statement 128 is executed, program execution moves to line 517. Line 517 is the next line executed after the execution of line 128.

Reference: Page 10-161.

■ **The AUTO key**

This typing aid displays the word AUTO when pressed.

Reference: Pages 10-161–10-162.

■ AUTO: **A command**

Automatically displays a new line number each time END LINE is pressed. Useful when entering program statements into memory.

To begin the line number series 10, 20, 30, 40, 50, ... press

SHIFT + AUTO END LINE

To begin another line number series, press SHIFT + AUTO [first line no.] , [interval between line numbers] END LINE

Example: To begin the line number series 100, 150, 200, 250, 300, ..., press SHIFT + AUTO 1 0 0 , 5 0 END LINE.

Reference: Pages 10-161–10-162.

■ **To cancel AUTO command,** press SHIFT + BACK SPACE to erase last line number, then execute NORMAL command (press N O R M A L END LINE). Also, executing the LIST or RUN command will cancel AUTO.
Reference: Page 10-161.

Review Test for Chapter 10

This test reviews material from earlier chapters as well as from Chapter 10. It is on your BASIC Training cartridge. Insert your cartridge, then execute LOAD "TEST10", and run it. Your instructions will appear on the screen. Program "TEST10" will also give you the answers. May you do well.

Chapter 11

Teach Your Program to Make Decisions

Preview

In Chapter 11, you will learn how to:

- Teach the HP-85 the difference between bigger, the same size, and smaller.
- Teach the HP-85 the difference between true and false.
- Tell the HP-85 to go this way if something is true, or that way if the same thing is false.
- Tell the HP-85 to stop.
- Tell the HP-85 to go this way if two things are true, or that way if either of the two things is false.
- Tell the HP-85 to go this way if two things are false, or that way if either of the two things is true.
- Try your luck at the Craps table.
- Write a program that sizes people up.
- Write an HP-85 Sweepstakes program giving all who enter an excellent chance to win big money.
- Write three other programs, including two quizzes.

Conditional Expressions

You're soon going to become immersed in IF ... THEN statements. These are concerned with the *truth* of such expressions as

A is bigger than B
X7 is smaller than 7
Y is equal to M + 3

As you'll learn later, the action of an IF ... THEN statement depends on the truth of such expressions; that is, the action is conditional on the truth of such expressions. That's where the term "conditional expressions" comes from.

BASIC borrows math symbols for these conditional expressions as follows, where "A" and "B" each stand for any number, variable name, or numeric expression, like 3*C/7+D.

(A > B) means A is greater than B.
(A = B) means A is equal to B.
(A < B) means A is less than B.

The parentheses are optional.

How about A is not equal to B?

(A <> B) or (A # B) means A is not equal to B

Here are the locations of the keys you press to get these symbols displayed.

> : shifted function of the (>.) key, two keys right of (M).
= : the (±=) key, two keys left of (BACK SPACE) (but you knew that already).
< : the shifted function of the (<;) key, just right of (M).
<> : first type <, then type >.
: the shifted function of the (#3) key, above (E).

For "not equal to," you may type either <> or #.

Here are some examples of conditional expressions that are true:

4 < 5	48 < 49
6 # 2	49 > 48
6 > 2	49 <> 48
3 = 3	48 # 49

TEST 11A: <, >, <> and #

A little drill won't hurt. Load "TEST11" and run it. The "TEST11" program is on your BASIC Training cartridge.

More Conditional Expressions

BASIC allows the truth of two more conditional expressions to be determined:

(A < = B) means A is less than or equal to B.
(A > = B) means A is greater than or equal to B.

Examples of true conditional expressions:

4 < = 5
6 > = 2
6 > = 6
4 < = 4
4 > = 4

TEST 11B: < = and > =

Run "TEST11" at line 3000 for another drill.

IF ... THEN: A BASIC Word

OK—Here's your chance to put your hard-learned conditionals to work. IF ... THEN is like a GOTO plus a question. IF this expression is true, THEN go to line so-and-so, otherwise go to the next line.

An example:

```
:
40 IF A=B THEN 70
50 C=0
60 GOTO 100
70 C=A*B
:
```

In this program segment, if A equals B; that is, if the conditional expression A = B is true, then program execution "branches" to line 70. C then becomes the product of A and B. However, if A is not equal to B, the conditional expression A = B is false. There is NO branching to line 70. Instead, the next line, 50, is executed, and C becomes zero.

Assignment Statement vs. the Conditional A = B

Take another look at lines 40 and 50 in the example above. The expression `C=0` in line 50 *is* an assignment statement because it *does* follow directly after the line number 50. However, the expression `A=B` in line 40 is *not* an assignment statement because it does *not* follow directly after the line number 40.

EXAMPLE: "Less Than Five" Program

```
10 ! LESS THAN FIVE
20 DISP "ENTER ANY NUMBER."
30 INPUT A
40 IF A>=5 THEN 70
50 DISP "YOUR NUMBER IS LESS TH
   AN FIVE."
60 GOTO 80
70 DISP "YOUR NUMBER IS 5 OR GR
   EATER."
80 END
```

FIGURE 80

Listing for "Less than Five"

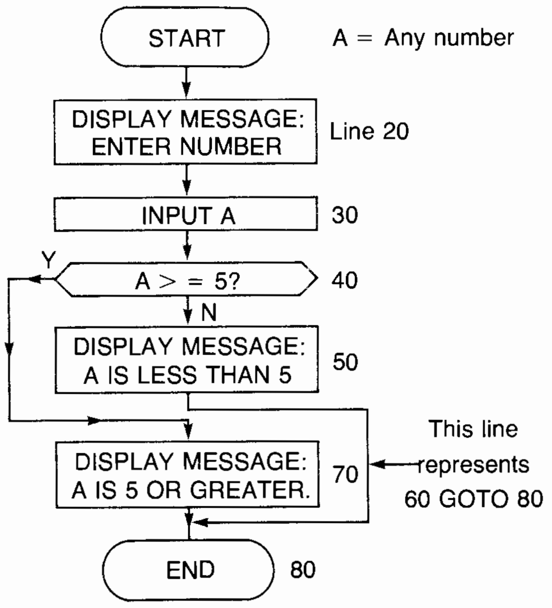

START
A = Any number
DISPLAY MESSAGE: ENTER NUMBER
Line 20
INPUT A
30
Y
A > = 5?
40
N
DISPLAY MESSAGE: A IS LESS THAN 5
50
This line represents 60 GOTO 80
DISPLAY MESSAGE: A IS 5 OR GREATER.
70
END
80

FIGURE 81

Flowchart for "Less than Five"

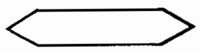
Note the new flowchart symbol: Y ? N

This is used whenever a yes-no decision is to be made, and results in a branching of the program flow.

Enter and run this program (Figure 80). Run it several times. Try to fool it. Perhaps, one time, the HP-85 will think 4 is greater than 5.

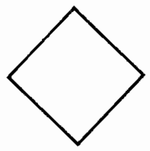
Figure 81 shows the flowchart for "Less Than Five."

Note the new conditional expression flowchart symbol:



This symbol is used in this course whenever a true-false or yes-no decision is to be made, and results in a branching of the program flow.

Here is another conditional expression symbol:

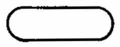


It is used in the Owner's Manual, and is often used by professional programmers.

When statement 40 in "Less Than Five" is executed, the value A is compared to 5. If A is greater or equal to 5, the expression A > = 5 is true (the answer to the question: Is A > = 5? is YES) and program execution branches (goes to) line 70. If A is not > = 5, the question: Is A > = 5? gets a NO answer (the expression A > = 5 is false), no branching occurs, and line 50 is the next statement executed.

STOP: A BASIC Word

One of the simplest BASIC words around. It operates just like END, except it may be used as any program statement *but* the last, whereas END should *always* be the last statement in a program. STOP also uses the same flowchart symbol as END:



The availability of STOP suggests a change in "Less Than Five."

PROBLEM: Modify "Less Than Five" Program

Change one line in your "Less Than Five" program to a STOP statement without affecting the visible operation of the program. Run your revised program to confirm your solution. Finally, draw a flowchart of your revised program.

To check your solution against mine, see page H-26. My flowchart shows line numbers, but they are optional. Most flowcharts do not show line numbers.

EXAMPLE: "Social Security, Anyone?" Program

```
 10 ! SOCIAL SECURITY, ANYONE?
 20 DISP "HOW OLD ARE YOU";
 30 INPUT A
 40 IF A<65 THEN 80
 50 DISP "YOU SHOULD BE RECEIVIN
    G SOCIAL"
 60 DISP "SECURITY PAYMENTS."
 70 STOP
 80 DISP "YOU'RE TOO YOUNG FOR S
    OCIAL"
 90 DISP "SECURITY PAYMENTS."
100 END
```

FIGURE 82

Listing for "Social Security, Anyone?"

Enter and run "Social Security, Anyone?," Figure 82.

PROBLEM: Flowchart for "Social Security, Anyone?" Program

Draw a flowchart for this program. To see an acceptable version, turn to page H-26. Again, line numbers on the flowchart are optional.

PROBLEM: Shorten "Social Security, Anyone?" Program

Now shorten this same program by one step without affecting its visible operation. Run your revised program to make sure it operates the same way. To see my listing and flowchart, turn to page H-27. Again, line numbers on the flowchart are optional.

PROBLEM: Write the "Size" Program

Program Description This program tells the user whether he or she is above average height, near average height, or below average height for his or her sex.

Important Data For this program a man is said to be near average height if he is 68, 69, 70 or 71 inches tall. A woman is said to be near average if she is 63, 64, 65 or 66 inches tall.

Hint In your answer to programming question 4, you might have the user of your program enter one number if male and another number if female. After your `INPUT` statement, you could then branch to a particular part of your program depending on the user's sex.

Your Turn Answer the four programming questions in a way that will be most useful to you. The four questions are repeated below.

Programming Questions (from page 134)

1. What answers do I want?
2. What things do I know?
3. What methods will I use to find answers using things I know? That is, how would I solve the problem using paper and pencil?
4. How can BASIC and the HP-85 help me find answers?

My answers to these questions are on page H-27. Compare your answers against mine before proceeding.

Next, draw a flowchart for ''Size.'' A flowchart that worked for me is on page H-28. Compare your flowchart against mine before continuing.

Finally, write your version of ''Size.'' To see one way to write this program, turn to page H-29.

If you had trouble writing a workable program, rest assured you have plenty of company. Study my flowchart and my listing. Modify your program based on the ideas they present. Of course, your messages will be different, and you may do things in a different order. One important thing is to write a program that solves the problem. Another important thing is to study and understand how my version works. Some of the techniques I use may be useful to you later on, and may suggest other techniques I haven't mentioned.

As always, the better you understand what you're doing now, the easier, more useful and more fun the rest of the course will be.

Writing the following two programs will give you more opportunities to use your new `IF` ... `THEN` power.

PROBLEM: Write the ''Sports Quiz'' Program

Program Description Test your friends' knowledge of the sports world with this sports quiz. The names of six athletes are printed, and six sporting events are displayed. The user is asked to enter the number of the athlete associated with each sporting event. If the user enters the wrong athlete number, he is given another chance. In fact, he is given as many chances as he needs.

Each athlete is associated with one and only one event.

Important Data Here are the athletes and the events:

The Athletes:

1. Mark Spitz
2. Don Larsen
3. Nadia Comaneci
4. Mickey Wright
5. Bob Beamon
6. Roger Bannister

The events:

1. First Olympian to win 7 gold medals in single Olympics.
2. First perfect game in world series history.
3. First perfect score in Olympic gymnastics.
4. Most golf victories in a single year.
5. World record shattered in long jump.
6. Four minute barrier is broken in the mile run.

Hint To help you write "Sports Quiz," you might wish to refer to my flowchart in Figure 83.

1 = Mark Spitz

2 = Don Larsen

3 = Nadia Comaneci

4 = Mickey Wright

5 = Bob Beamon

6 = Roger Bannister

A = Answer to 1st event

B = Answer to 2nd event

C = Answer to 3rd event

D = Answer to 4th event

E = Answer to 5th event

F = Answer to 6th event

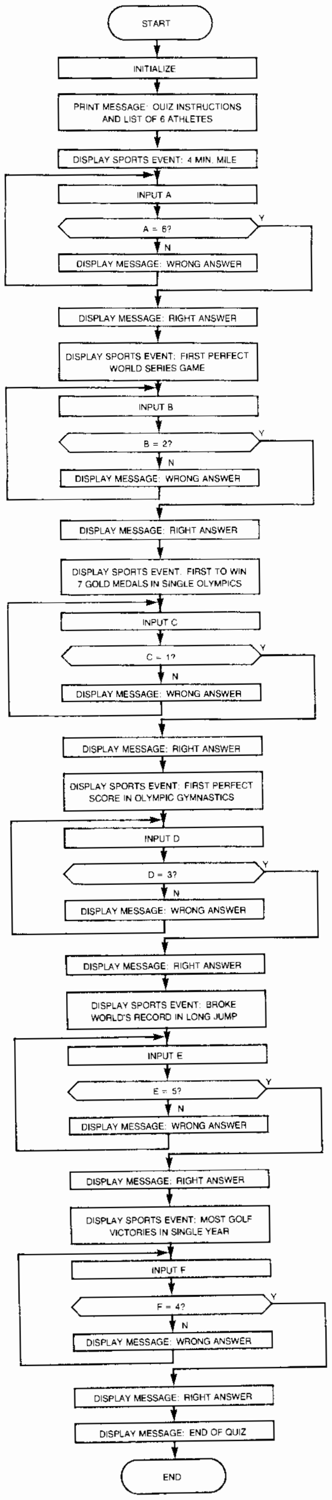

START
INITIALIZE
PRINT MESSAGE: QUIZ INSTRUCTIONS AND LIST OF 6 ATHLETES
DISPLAY SPORTS EVENT: 4 MIN. MILE
INPUT A
A = 6?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY SPORTS EVENT: FIRST PERFECT WORLD SERIES GAME
INPUT B
B = 2?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY SPORTS EVENT: FIRST TO WIN 7 GOLD MEDALS IN SINGLE OLYMPICS
INPUT C
C = 1?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY SPORTS EVENT: FIRST PERFECT SCORE IN OLYMPIC GYMNASTICS
INPUT D
D = 3?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY SPORTS EVENT: BROKE WORLD'S RECORD IN LONG JUMP
INPUT E
E = 5?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY SPORTS EVENT: MOST GOLF VICTORIES IN SINGLE YEAR
INPUT F
F = 4?
Y
N
DISPLAY MESSAGE: WRONG ANSWER
DISPLAY MESSAGE: RIGHT ANSWER
DISPLAY MESSAGE: END OF QUIZ
END

FIGURE 83

Flowchart for "Sports Quiz"

Your Turn Write your "Sports Quiz" program. Store it temporarily if you wish using the name "WORK." My listing is on page H-30.

PROBLEM: Write the "Science Quiz" program

This "Science Quiz" program is similar to "Sports Quiz," except the names of six scientists are printed and six discoveries or inventions are displayed. Again, one scientist is responsible for one and only one of the six scientific contributions.

Important Data

The scientists:	The contributions:
1. Ivan Pavlov	1. Conditioned reflex, 1910.
2. Sir Isaac Newton	2. Law of gravity in 1687.
3. Thomas A. Edison	3. Incandescent lamp, 1879.
4. Eli Whitney	4. Development of the cotton gin, 1793.
5. Sigmund Freud	5. Use of psychoanalysis in 1904.
6. Marie Curie	6. Discovery of radium, leading to radiotherapy.

Your Turn Write your "Science Quiz" program. If you wish, store it temporarily using the name "WORK." My flowchart is on page H-31 and my listing is on page H-32.

Abridged Dictionary of HP-85's BASIC Language

You may have already discovered this section at the back of the Supplement, starting on page AD-64.

This Dictionary collects in one place all the BASIC words, commands, functions, and mathematical operators covered in this course. If you forget how to use a particular BASIC word, for instance, this dictionary can help you. The Index at the back of this workbook can also be a help. As an example, if you wish to store a program and forget how to display the cartridge catalog, you can find help in the Dictionary, page AD-68, or in the Index under "catalog."

IF ... AND ... THEN: A BASIC Word

Let's say you wrote a program to tell a visitor from outer space whether or not today is Christmas. The visitor would enter today's date expressed as a month number and a day number. The program might include these statements:

```
   :
 50 INPUT M
   :
 80 INPUT D
   :
100 IF M=12 AND D=25 THEN 130
110 DISP "SORRY, IT'S NOT CHRIST
    MAS."
   :
130 DISP "IT'S CHRISTMAS!"
   :
```

IF the month is December (12) AND the day is 25, THEN our little green friend would see displayed IT'S CHRISTMAS! Otherwise, he (she? it?) would see SORRY, IT'S NOT CHRISTMAS.

Depending on the values of M and D, statement 100 presents 4 possibilities:

M = 12?	**D = 25?**	**Next line executed**
NO	NO	110
YES	NO	110
NO	YES	110
YES	YES	130

The general form of IF ... AND ... THEN is

IF *both* (A is true) AND (B is true) THEN go to [line number]

Otherwise, go to the next line.

EXAMPLE: "It's Christmas" Program

Figure 84 shows the flowchart for "It's Christmas" and Figure 85 shows its complete listing. Using the flowchart as a guide, study the listing until you feel comfortable with its operation. Then enter this program and run it. Try a number of inputs. Get ready for those visitors.

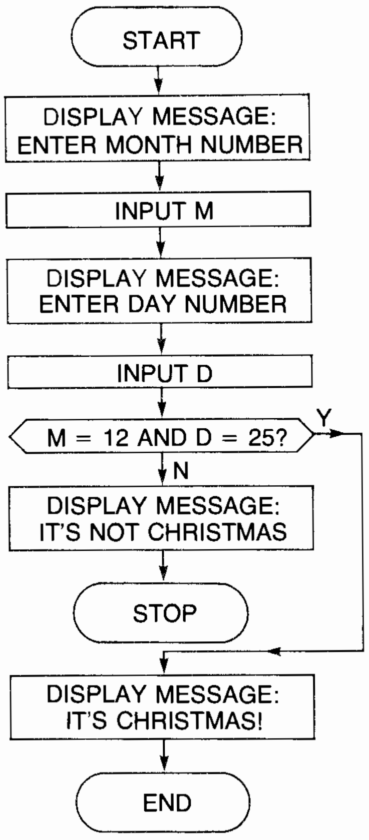

START
DISPLAY MESSAGE: ENTER MONTH NUMBER
INPUT M
DISPLAY MESSAGE: ENTER DAY NUMBER
INPUT D
M = 12 AND D = 25?
Y
N
DISPLAY MESSAGE: IT'S NOT CHRISTMAS
STOP
DISPLAY MESSAGE: IT'S CHRISTMAS!
END

FIGURE 84

Flowchart for "It's Christmas"

```
 10  ! IT'S CHRISTMAS
 20  DISP
 30  DISP
 40  DISP "ENTER MONTH NUMBER (1-
     12)."
 50  INPUT M
 60  DISP
 70  DISP "ENTER DAY NUMBER (1-31
     )."
 80  INPUT D
 90  DISP
100  IF M=12 AND D=25 THEN 130
110  DISP "SORRY, IT'S NOT CHRIST
     MAS."
120  STOP
130  DISP "IT'S CHRISTMAS!"
140  END
```

FIGURE 85

Listing for "It's Christmas"

M = Month number
(1-12)

D = Day number
(1-31)

PROBLEM: Write the "Temperature Conversion" Program

Program Description The program asks the user whether he wishes to convert from F° to C° or from C° to F°. Based on the reply, the program asks for a temperature to be entered and then displays the converted temperature. To convert another temperature, the user runs the program again.

If the user makes an input error when asked which conversion he wishes, he is asked again.

Important Formulas To convert from Fahrenheit to Celsius degrees:

$$F = 9/5 * C + 32$$

To convert from Celsius to Fahrenheit degrees:

$$C = 5/9 * (F - 32)$$

Hint Your program might ask the user to enter 1 if he wants to convert from F° to C°, and to enter 2 if he wants to convert from C° to F°. Then your program can test the user's input. If his input is not 1 and also not 2, he has made an input error. You can then have your program present the input message again.

Your Turn Draw a flowchart and write a program to help you enter the metric age. Or, if you're already there, your program will help you to understand those who aren't. If you wish to interrupt your programming, remember to store it using the name "WORK."

My flowchart and listing are on page H-33.

IF ... OR ... THEN: A BASIC Word

Consider the dice game Craps. On the first throw, the shooter wins if his dice total either 7 or 11. A BASIC program written to play a form of Craps might contain these statements:

```
:
360 IF T=7 OR T=11 THEN 430
370 ...
:
430 DISP "YOU WIN!"
:
```

Depending on the value of T, statement 360 presents three possibilities:

T = 7?	T = 11?	Next line executed
NO	NO	370
YES	NO	430
NO	YES	430

In general, there are four possibilities if two different variables are used in the `IF ... OR ... THEN` statement.

Consider this program segment:

```
:
250 IF A=2 OR B=5 THEN 300
260 ...
:
300 ...
:
```

Depending on the values of A and B, statement 250 presents four possibilities:

A = 2?	**B = 5?**	**Next line executed**
NO	NO	260
YES	NO	300
NO	YES	300
YES	YES	300

The general form of IF ... OR ... THEN is

IF *either* (A is true) OR (B is true) THEN go to [line number]

EXAMPLE: "CRAPS" Program

This program allows you to play Craps with the HP-85. You always throw the dice, and you always choose the size of the bet. The HP-85 always fades your entire bet; that is, the HP-85 always bets an equal amount. Let me review the rules. On your first throw, you win with a 7 or 11 and lose with 2, 3 or 12. Any other total (4, 5, 6, 8, 9, 10) becomes your point. On subsequent throws, if you throw your point, you win. If you throw a 7, you lose. If you throw any other total, you neither win nor lose, you simply throw again.

The "CRAPS" program is on your BASIC Training cartridge. Execute LOAD"CRAPS" and run it (press RUN). I hope you have a hot streak.

Now that you've won (lost?) all that money, look at the listing for "CRAPS," Figure 87. (You could easily generate the identical listing from the "CRAPS" program you just loaded and ran). Notice, in lines 50, 220 and 230, that I preview two BASIC words, RANDOMIZE and RND (for random). You'll study these in Chapter 18. These allow you to "throw" dice using the HP-85 with virtually the same element of chance as if you were tossing the cubes yourself.

Now take a look at the flowchart, Figure 86. First, satisfy yourself that the flowchart follows the flow of the game and the rules described above. Next, compare the flowchart with the listing, Figure 87. Relate each task on the flowchart with the corresponding lines on the listing. Before too long, I'll be asking you to write programs this complex.

More About Flowcharts

Lay your eyes once again on the "CRAPS" flowchart, Figure 86. It is written using English words rather than BASIC words.

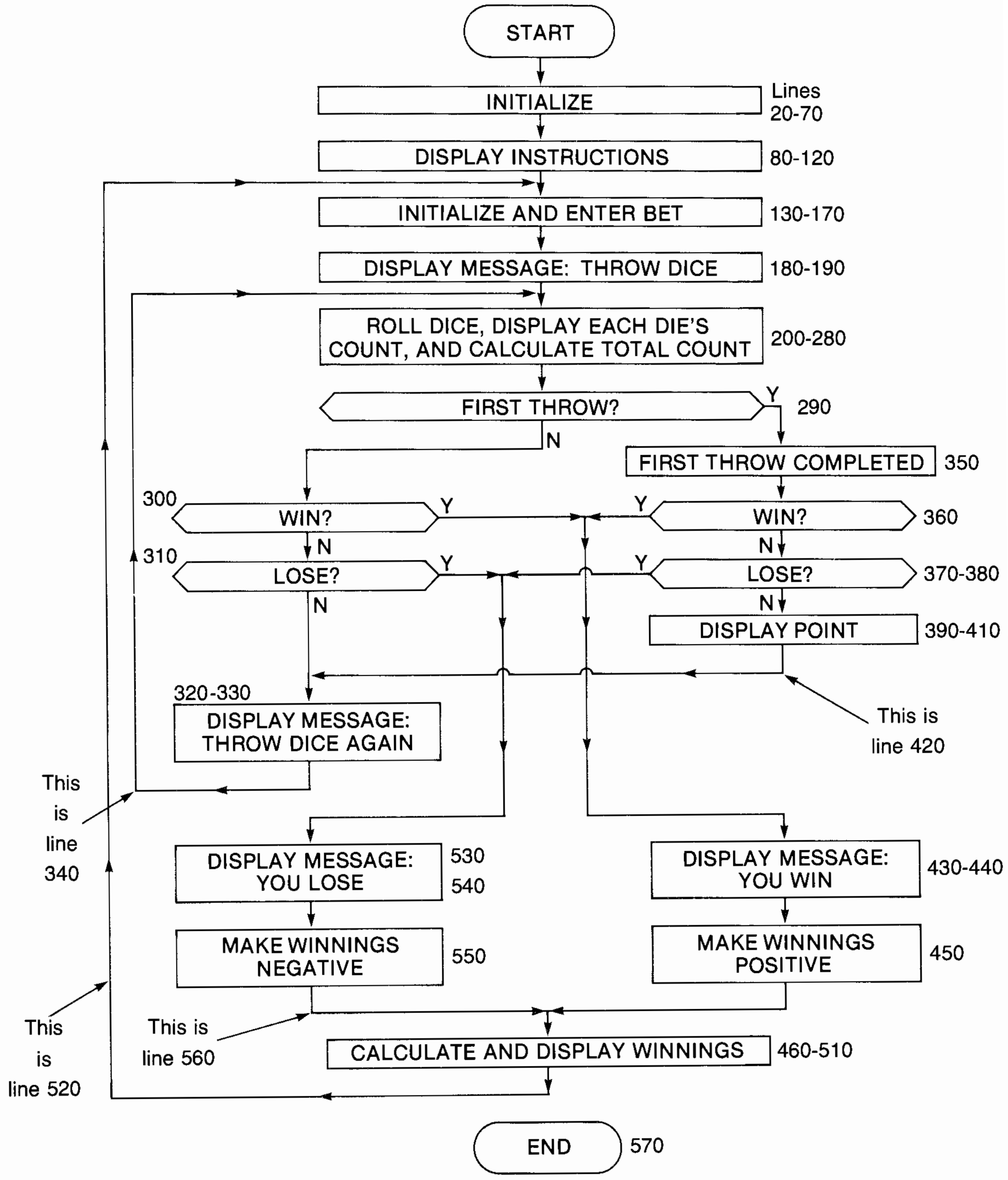


FIGURE 86

Flowchart for "CRAPS"

```
10 ! CRAPS
20 NORMAL
30 CLEAR
40 CRT IS 1
50 RANDOMIZE
60 W=0
70 W1=0
80 DISP
90 DISP
100 DISP "YOU AND THE HP-85 HAVE
     AGREED TOPLAY CRAPS. YOU WI
    LL ALWAYS"
110 DISP "THROW THE DICE, AND YO
    U WILL AL-WAYS CHOOSE THE SI
    ZE OF THE BET."
120 DISP "THE HP-85 WILL ALWAYS
    MATCH YOURBET."
130 P=0
140 R=0
150 DISP
160 DISP "HOW MUCH DO YOU BET (W
    HOLE DOL- LARS ONLY, PLEASE)
    ";
170 INPUT B
180 DISP
190 DISP "YOU THROW YOUR DICE AN
    D SEE:"
200 WAIT 1000
210 T=0
220 D1=INT(6*RND+1)
230 D2=INT(6*RND+1)
240 DISP
250 DISP "     ";D1;"     ";D2
260 DISP
270 WAIT 1000
280 T=D1+D2
290 IF R=0 THEN 350
300 IF T=P THEN 430
310 IF T=7 THEN 530
320 DISP
330 DISP "YOU THROW AGAIN AND SE
    E:"
340 GOTO 200
350 R=1
360 IF T=7 OR T=11 THEN 430
370 IF T=2 OR T=3 THEN 530
380 IF T=12 THEN 530
390 P=T
400 DISP P;"IS YOUR POINT."
410 WAIT 1000
420 GOTO 320
430 DISP "YOU WIN!"
440 DISP
450 W=B
460 WAIT 1000
470 W1=W1+W
480 DISP "YOUR TOTAL WINNINGS AR
    E"
490 DISP W1;"DOLLARS."
500 WAIT 1000
510 DISP
520 GOTO 130
530 DISP "SORRY, YOU LOSE."
540 DISP
550 W=-B
560 GOTO 460
570 END
```

FIGURE 87

Listing for "CRAPS"

Compare this flowchart for ''CRAPS'' with the flowchart for ''It's Christmas,'' Figure 84, which uses BASIC words and phrases like ''INPUT D'' and ''M=12 AND D=25.'' A BASIC language flowchart is generally more detailed. Each flowcharted task represents, in general, only a few statements. For example, no symbol in the ''It's Christmas'' flowchart, Figure 84,represents more than three statements. On the other hand, a flowcharted task in an English language flowchart may represent many statements. Notice how the task symbol in the ''CRAPS'' flow-chart containing ''Roll dice, display each die's count, and calculate total count'' represent 9 statements. When you write a program, it is natural to construct an English language flowchart, and use BASIC language only when you write your program statements.

Most of my flowcharts will use more BASIC than English language. When I give you both flowchart and listing, as I often do with example programs, a BASIC language flowchart allows easier comparison between flowchart and listing. For problem programs, where I ask you to construct a flowchart and write a program, you might get stuck and want to use my flowchart as a hint. In this case, a BASIC language flowchart might be a better hint than one using primarily English words.

Back to Conditional Statements

In a statement like

```
418 IF (A=17) OR (B=317) THEN 35
```

you might find it useful to use parentheses as shown around each conditional expression to help clarify the statement. However, they are not required. The HP-85 is just as happy with

```
418 IF A=17 OR B=317 THEN 35
```

and your listings will omit parentheses.

Both of the examples you just studied, "It's Christmas" and "CRAPS," use equalities in the two conditional statements. For instance, line 100 in "It's Christmas":

```
100 IF M=12 AND D=25 THEN 130
```

uses two equalities, `M=12` and `D=25`. Line 360 in "CRAPS" also uses two equalities:

```
360 IF T=7 OR T=11 THEN 430
```

Of course, the conditionals within the parentheses need not be equalities. Any conditional expression may be used, such as:

```
45 IF (25*F<=D/3) AND (G/7=W) TH
EN 305

37 IF (X6>=45^6) OR (A<6) THEN 2
50

15 IF (B=3) AND (50>=F*(G+4)) TH
EN 100
```

In each case, the *truth* of the first conditional expression is determined first, then the truth of the second expression is determined, and finally the HP-85 decides whether or not to branch to the line number following `THEN`.

PROBLEM: Write the "Sweepstakes" Program

Tired of entering those junk mail sweepstakes and never winning? Here is a sweepstakes designed for you! The HP-85 Sweepstakes program you will write will not only provide you with a tidy bundle, but it will also exercise your new BASIC tools, `IF` ... `AND` ... `THEN` and `IF` ... `OR` ... `THEN`.

Program Description "Sweepstakes" first displays a message stating that the certificate number to be requested later should be a whole number between 1 and 9999. The program then calculates a lucky number and asks for the certificate number. The certificate number that is entered is tested to see if it is a whole number between 1 and 9999. If it is not, an appropriate message is displayed, and another certificate number is requested.

When the number passes the test, the certificate and lucky numbers are compared to determine what prize, if any, is won. Most often, nothing is won, but prizes of $100, $10,000, $250,000 and $1,000,000 are possible. After this comparison is completed, an appropriate message of failure or success is displayed. The lucky number is then recalculated, and another certificate number is requested. The program operates as an endless loop—the END statement is never reached.

Important Expressions and Definitions "Sweepstakes" also previews RANDOMIZE and RND, the same new BASIC words used in "CRAPS." Use RANDOMIZE as an early statement, such as:

```
50 RANDOMIZE
```

and use RND as shown in expression 1 below:

1. L = INT(9999 ⋆ RND + 1)
   Where L = Lucky number
2. $1,000,000 winner: C within 50 of L.
   Where C = Certificate number
3. $250,000 winner: C within 400 of L.
4. $10,000 winner: C within 900 of L.
5. $100 winner: C within 2000 of L.

Hint To test the certificate number you'll enter when you run the program, use two statements. One can check to see if the number is smaller than 1 or larger than 9999. If it is, you entered an incorrect certificate number. The other can check to see if you entered a whole number; that is, an integer. If C is an integer, C would equal INT(C). If it doesn't, the number doesn't pass the test. Only if C passes both of these tests is it then compared against L to determine your winnings.

Your Turn This is your toughest challenge so far. If you complete a satisfactory program without getting too much help, you'll be in good shape to continue.

You'll find the answers to the first three programming questions (see below) to be no more difficult than the ones you've already completed. Answering the fourth question will require more thought, but you have all the tools you'll need. Remember, the more effort you put into solving this problem yourself before turning to the HELP Section, the more you'll learn.

Here are the questions:

1. What answers do I want?
2. What things do I know?
3. What methods will I use to find answers using things I know? That is, how would I solve the problem with paper and pencil?
4. How can BASIC and the HP-85 help me find answers?

The more completely you answer these questions—especially question 4—the easier time you'll have with the flowchart and program.

My answers to the questions are on pages H-34 and H-35. Check your answers against mine before continuing.

Now draw your flowchart. Compare yours against mine on page H-36.

Using your flowchart as a guide, write your program. Remember, if you want to store your unfinished program on your BASIC Training cartridge, execute `STORE"WORK"`. Get it back by executing `LOAD"WORK"`.

When you've finished, compare your ''Sweepstakes'' against mine on page H-37. Yours may be better.

Summary of Chapter 11

In this summary, ''A'' and ''B'' each represent any number, variable name, or numeric expression.

- **Conditional expressions**

  (A > B) means A is greater than B.

  (A = B) means A is equal to B.

  (A < B) means A is less than B.

  (A <> B) or (A # B) means A is not equal to B.

  (A <= B) means A is less than or equal to B.

  (A >= B) means A is greater than or equal to B.

  Reference: Pages 168, 11-169 and 11-182.

- **Difference between assignment statement `100 A=B` and conditional expression `A=B`.**

  In the assignment statement, `A=B` directly follows a line number. The conditional expression `A=B` never directly follows a line number.

  Reference: Page 11-170

- IF ... THEN: **A BASIC word**

General form:

[line number] IF [conditional expression] THEN [line number]

When this statement is executed, the truth of the conditional expression is determined. If the expression is true, then program execution branches to [line number]. If the expression is false, program execution continues with the next line.

Example:

```
:
35 A=5
40 B=10
45 IF A>B THEN 100
50
:
100
:
```

In statement 45, (A> B) is (5> 10) which is false. Since the conditional expression is false, program execution continues with the next line, 50. Branching to line 100 does *not* happen.

Reference: Pages 11-169 and 11-170.

- STOP: **A BASIC word**

General form:

[line number] STOP

When a STOP statement is executed, program execution halts. STOP and END each halt program execution. STOP may be at any line number *except* the highest numbered line in the program. END should appear *only* at the highest numbered line in the program.

Reference: Page 11-171.

- IF ... AND ... THEN: **A BASIC word**

General form:

[line number] IF [conditional expression] AND [conditional expression] THEN [line number]

If both conditional expressions are true, the program branches to [line number]. Otherwise, program execution continues with the next line.

Example:

```
10 A=1
20 B=2
30 C=3
40 D=4
50 IF (A<B) AND (C<D) THEN 100
60 A=0
:
100 D=0
:
```

In statement 50, (A<B) is (1<2) which is true, and (C<D) is (3<4) which is true. Since both conditional expressions are true, the program branches to line 100. Line 60 is not executed.

Reference: Pages 11-175–11-176.

- `IF ... OR ... THEN`: **A BASIC word**

General form:

[line number] `IF` [conditional expression] `OR` [conditional expression] `THEN` [line number]

If at least one conditional expression is true, the program branches to [line number]. If both expressions are false, program execution continues with the next line.

Example:

```
10 A=1
20 B=2
30 C=3
40 D=4
50 IF (A=B) OR (C<D) THEN 100
60 A=0
:
100 D=0
:
```

In statement 50, (A=B) is (1=2) which is false, and (C<D) is (3<4) which is true. Since at least one conditional expression is true, the program branches to line 100. Line 60 is not executed.

Reference: Pages 11-178–11-179.

- **Flowcharts: English vs. BASIC**

When you construct a flowchart to help organize your thoughts, then use the flowchart to help write your program, it's natural to use more English than BASIC words to describe your flowcharted tasks.

However, if you construct a flowchart to help explain in detail how an existing program works, you might choose to use more BASIC words to describe flowcharted tasks. Then a reader can more easily relate flowchart tasks to program statements.

The bottom line is still this: Use flowcharting in a way that helps you most.

Reference: Pages 11-179–11-182.

Review Test for Chapter 11

The answers to problems 1 through 5 start on page 11-188 immediately following this Review Test. Program "TEST11" will direct you to the answers to problems 6, 7 and 8.
For each problem, determine at which line program execution will halt, line 60 or line 70.

1.

```
10 A=3
20 B=5
30 C=7
40 D=9
50 IF A>B AND C<=D THEN 70
60 STOP
70 END
```

2.

```
10 A=3
20 B=5
30 C=7
40 D=9
50 IF A>B OR C<=D THEN 70
60 STOP
70 END
```

3.

```
10 A=3
20 B=5
30 C=7
40 D=9
50 IF A<B AND C<=D-2 THEN 70
60 STOP
70 END
```

4.

```
10 A=9
20 B=7
30 C=5
40 D=3
50 IF A-2=C OR B<D THEN 70
60 STOP
70 END
```

5.

```
10 A=1
20 B=2
30 C=3
40 D=4
50 IF A<B AND C<=D-2 THEN 70
60 STOP
70 END
```

Before proceeding, compare your answers to these first five problems with those on page 11-188.

Problems 6, 7 and 8 are on your BASIC Training cartridge. Insert your cartridge, load "TEST11" and run it at line 3500. The program will give you instructions and answers.

Answers to Review Test Questions for Chapter 11

Problems 1-5

Reference for problems 1-5: Pages 11-168–11-169; pages 11-175–11-176; pages 11-178–11-179.

1. Line 60

   (A>B) is (3>5) which is *false*
   (C<=D) is (7<=9) which is *true*

   Since *both* expressions are *not* true, branching does *not* occur. The next line, `60 STOP`, is executed, which halts the program.

2. Line 70

   (A>B) is (3>5) which is *false*
   (C<=D) is (7<=9) which is *true*

   Since *one* expression *is* true, branching *does* occur to line 70.

3. Line 70

   (A<B) is (3<5) which is *true*
   (C<=D−2) is (7<=7) which is *true*

   Since *both* expressions *are* true, branching *does* occur to line 70

4. Line 60

   (A−2=C) is (7=5) which is *false*
   (B<D) is (7<3) which is *false*

   Since *neither* expression is true, branching does *not* occur. The next line, `60 STOP`, is executed, which halts the program.

5. Line 60

   (A<B) is (1<2) which is *true*
   (C<=D−2) is (3<=2) which is *false*

   Since *both* expressions are *not* true, branching does *not* occur. The next line, `60 STOP`, is executed, which halts the program.

Chapter 12

Teach Your Program to Count

Preview

In Chapter 12, you will:

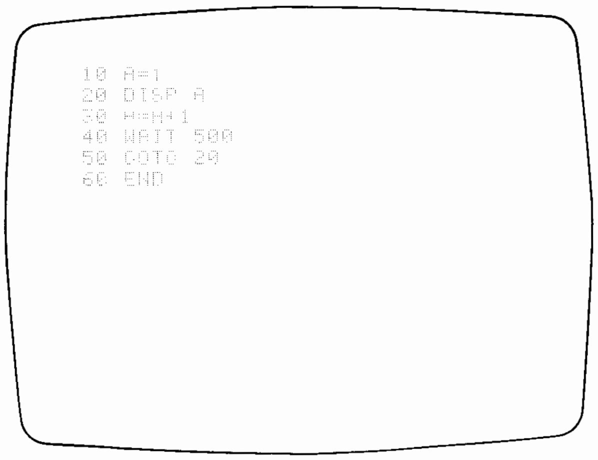
- Teach your program to count.
- Learn how a table can help you plan and check a program that loops.
- Modify two example programs to make them perform fancier tricks.
- Write three new programs.

Counter Based on A = A + 1

This is an important programming concept. Put it in a place of honor in your bag of tricks.

EXAMPLE: "Count" Program

Figure 88 shows the listing for "Count," and Figure 89 is the flowchart.



```
10 A=1
20 DISP A
30 A=A+1
40 WAIT 500
50 GOTO 20
60 END
```

FIGURE 88

Listing for "Count"

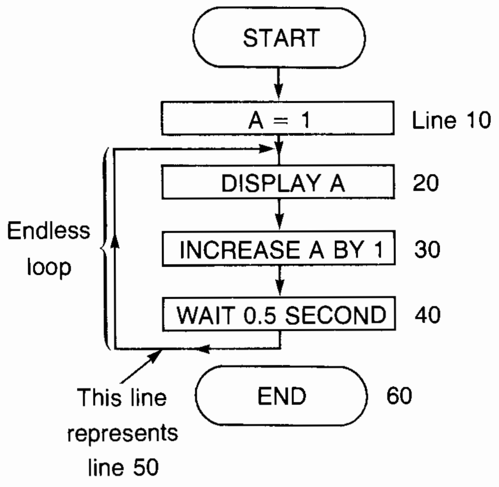


FIGURE 89

Flowchart for "Count"

Enter "Count" into your HP-85 and run it to test its performance.

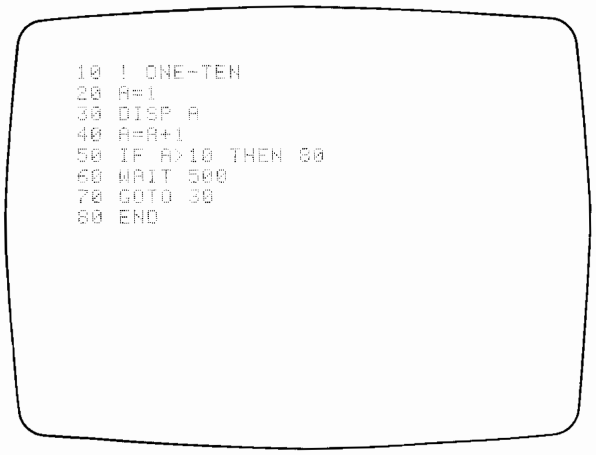
Each time through the loop, one is added to the old value of A, this new value is assigned to A, then A's value is displayed. To stop this endless loop, PAUSE must be pressed. Or, we can program our way around it as in the following example:

EXAMPLE: "One-Ten" Program

Please enter this new line into "Count":

```
35 IF A>10 THEN 60
```

Name the new program "One-Ten," put the title in a remark at the beginning, then execute REN and press LIST to get the listing shown in Figure 90.



```
10 ! ONE-TEN
20 A=1
30 DISP A
40 A=A+1
50 IF A>10 THEN 80
60 WAIT 500
70 GOTO 30
80 END
```

FIGURE 90
Listing for "One-Ten"

Notice another important power of REN. The line number at the end of the IF ... THEN statement was automatically changed from 60 to 80. Also, the line number in the GOTO statement was changed from 20 to 30. This power is especially welcome when renumbering long programs containing many IF ... THEN and GOTO statements.

Figure 91 shows the flowchart for "One-Ten."

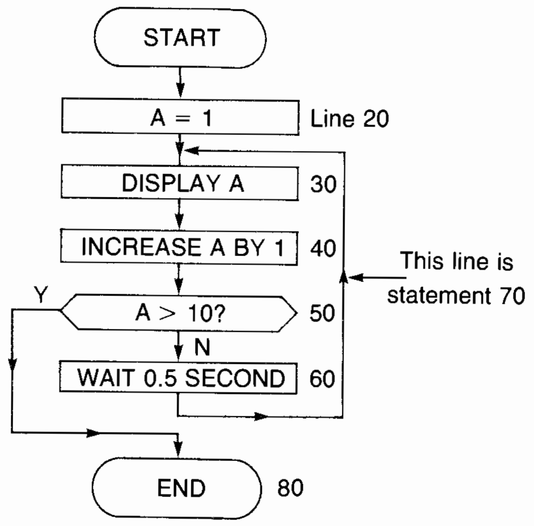


FIGURE 91

Flowchart for "One-Ten"

As you can see, each time through the loop, the variable A is tested in statement 50 to see if it's greater than 10. If the answer is no, the number is displayed, and incremented by one (one is added to the number). When the statement 50 question is answered yes, the program branches to the END statement.

If you haven't done so, run your "One-Ten" program.

PROBLEM: Write the "One-Hundred" Program

Modify "One-Ten" to create "One-Hundred," a program that counts from 1 to 100. You might wish to reduce the time interval in the WAIT statement to speed up the action a little. When you're finished, compare your version against mine on page H-38.

Next, draw a new flowchart. Mine is on page H-38.

These two programs illustrate the power of programming. The same size program gives an output ten times longer.

EXAMPLE: "Sum 1 Thru 25" Program

We want a program that will display the sum of the first 25 integers. That is, what is 1 + 2 + 3 + 4 + ... + 24 + 25? Let's look at one way to answer program planning questions 3 and 4.

3. What methods will I use to find answers using things I know?

   Well, I'll take 1 and add 2 to it and get a sum of 3. I'll then add the next integer 3 to the sum giving a new sum of 6. Next I'll add the next integer 4 to the latest sum and come up with a newer sum of 10, etc.

4. How can BASIC and the HP-85 help me find answers?

   I'll call each of the integers I'm adding together I. I'll form a loop including I = I + 1 (next integer = old integer + 1) to generate each integer in turn. I'll call the sum S. Within this loop, I'll use S = S + I (new sum = old sum + next integer). When I equals 25, I want to get out of the loop. I can use an `IF` ... `THEN` to do this. Knowing how the HP-85 shakes its finger at me if I don't initialize my variables, I had better use I = 0 and S = 0 before I get into the loop. When I've added 25 to the sum and left the loop, then I'll display the final sum S.

Why assign zero to I and S at the beginning? Why not one? A good question. To answer it, let's inspect the flowchart, Figure 92, and the listing, Figure 93. Each time through the loop, I is increased by 1 (statement 40) and I is added to S (statement 50). I'll show, step by step, how the program sums the first three integers when I and S are initialized to zero. Then I'll do the same when I and S are initialized to one.

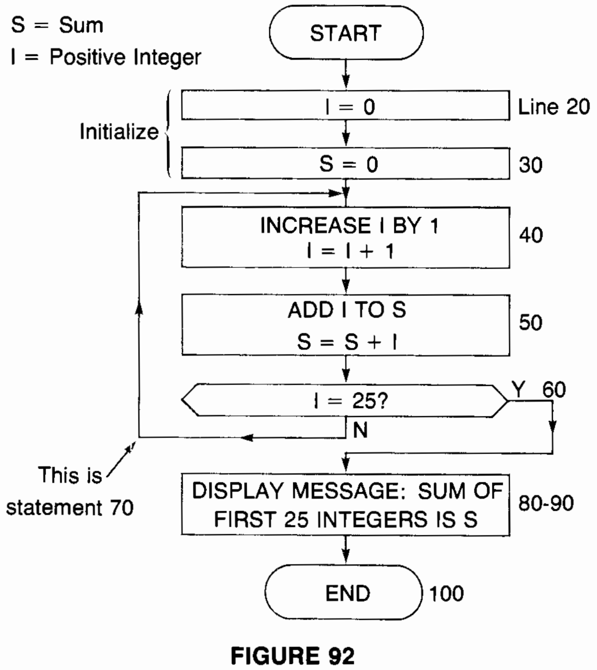
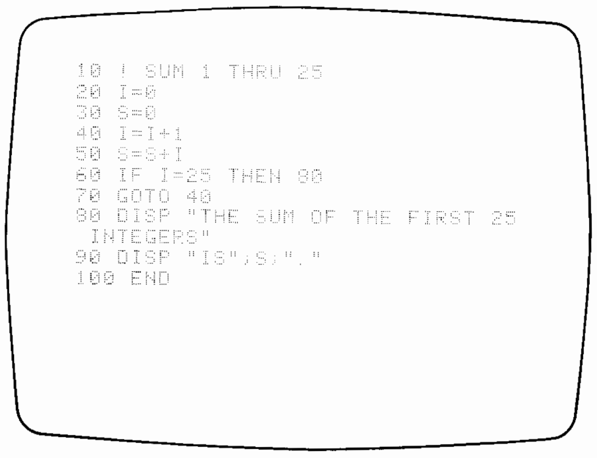


FIGURE 92
Flowchart for "Sum 1 Thru 25"



```
10 ! SUM 1 THRU 25
20 I=0
30 S=0
40 I=I+1
50 S=S+I
60 IF I=25 THEN 80
70 GOTO 40
80 DISP "THE SUM OF THE FIRST 25
 INTEGERS"
90 DISP "IS";S;"."
100 END
```

FIGURE 93
Listing for "Sum 1 Thru 25"

First, I and S are initialized to 0 in lines 20 and 30:

```
20 I=0
30 S=0
```

Then, during the first trip through the loop:

```
40 I=I+1
```

Since I=0 from line 20:

I=0+1

I=1

```
50 S=S+I
```

Since S=0 from line 30 and I=1 from line 40:

S=0+1

S=1

During the second trip through the loop:

```
40 I=I+1
```

Since I=1 from the last execution of line 40:

I=1+1

I=2

```
50 S=S+I
```

Since S=1 from the last execution of line 50 and I=2 from line 40:

S=1+2

S=3

In words, the sum of the first two integers (I=1, I=2) is 3 (S=3).

During the third trip through the loop:

```
40 I=I+1
```

Since I=2 from the last execution of line 40:

I=2+1

I=3

```
50 S=S+I
```

Since S=3 from the last execution of line 50 and I=3 from line 40:

S=3+3

S=6

In words, the sum of the first three integers (I=1, I=2, I=3) is 6 (S=6).

Now let's try initializing I and S to 1; that is, let's change lines 20 and 30 to:

```
20  I=1
30  S=1
```

For loop 1:

```
40  I=I+1
```

Since I=1 from line 20:

I=1+1

I=2

```
50  S=S+I
```

Since S=1 from line 30 and I=2 from line 40:

S=1+2

S=3

For loop 2:

```
40  I=I+1
```

Since I=2 from the last execution of line 40:

I=2+1

I=3

```
50  S=S+I
```

Since S=3 from the last execution of line 50 and I=3 from line 40:

S=3+3

S=6

In words, the sum of the first three integers (I=1, I=2, I=3) is 6 (S=6).

Our analysis has shown that, for this program, we could initialize I and S to one rather than zero. Initializing I and S to zero lets the program calculate the first integer and the first sum. When I and S are initialized to one, the programmer has already calculated the first integer and first sum. We have learned something about this particular program by trying a simple example. Which brings me to

ANOTHER IMPORTANT TRUTH!

This is really a corollary of that other great truth: **When in doubt—try it.**

This truth is:

Test your program with an example whose answer you know.

Let's ask the HP-85 to run the same simple example we just completed by hand. A way to do this with "Sum 1 Thru 25" is to change statement 60 to

```
60  IF  I=3  THEN  80
```

Don't worry about the text of statement 80.

We know that the sum of the first three integers is 1 + 2 + 3 = 6. Enter "Sum 1 Thru 25" with the revised statement 60, run it, and see line 90 display 6. Now change statements 20 and 30 to

```
20 I=1
30 S=1
```

and press RUN. Again, see 6. So far, so good. To be more certain, change 60 to

```
60 IF I=4 THEN 80
```

since your steel-trap brain can figure out that 1 + 2 + 3 + 4 = 10. Now press RUN. The program agrees with you that 1 + 2 + 3 + 4 does indeed equal 10. For a final check, change 20 and 30 back to

```
20 I=0
30 S=0
```

Press RUN again. Again OK. The final sum is still 10.

Now get the original program by changing 60 to

```
60 IF I=25 THEN 80
```

Press RUN, and discover that the sum of the first 25 integers is 325.

Table of Variable Values vs. Loop Numbers

This is the variable value vs. loop number table for the "Sum 1 Thru 25" program.

	Initial Values									General Expression
Loop No.		1	2	3	4	5	6	7	...	
Integer I	0	1	2	3	4	5	6	7	...	I=I+1
Sum S	0	1	3	6	10	15	21	28	...	S=S+I

To show how each succeeding sum can be calculated just by looking at the table, I'll redraw the table in this way:

	Initial Values										General Expression
Loop no.		1	2	3	4	5	6	7	8	...	
Integer I	0	1	2	3	4	5	6	7	8	...	I=I+1
Sum S	0^+	1^+	3^+	6^+	10^+	15^+	21^+	28^+	36	...	S=S+I

Start with the initial value for the sum, 0, and add the loop no. 1 value for the integer, which is 1. This gives 0 + 1 = 1, which is the loop no. 1 value for the sum. Now start with this loop no. 1 sum, 1, and add the loop no. 2 value for the integer, 2. Now we have 1 + 2 = 3, which is the loop no. 2 value for the sum. Similarly:

$3 + 3 = 6$

$6 + 4 = 10$

$10 + 5 = 15$

$15 + 6 = 21$

$21 + 7 = 28$

$28 + 8 = 36$

and so on.

Such a variable value vs. loop number table can help you in two ways to write a program.

1. It helps you to understand the problem.
2. It helps you to check the program.

EXAMPLE: How to Write the "Sum Odd 100" Program

This program should sum the first 100 odd integers. That is, it should display the sum of:

$$\begin{array}{ccccccccccccccccccc}
1 & + & 3 & + & 5 & + & 7 & + & 9 & + & \ldots & + & 197 & + & 199 \\
(1) & & (2) & & (3) & & (4) & & (5) & & & & (99) & & (100) \\
\uparrow & & & & & & \uparrow & & & & & & \uparrow & & \\
\text{1st} & & & & & & \text{4th} & & & & & & \text{99th} & & \\
\text{odd} & & & & & & \text{odd} & & & & & & \text{odd} & & \\
\text{integer} & & & & & & \text{integer} & & & & & & \text{integer} & &
\end{array}$$

Let's use a variable value vs. loop number table to help us plan this program.

	Initial Values					
Loop No.		1	2	3	4	...
Number Counter	0	1	2	3	4	...
Odd Integer	0	1	3	5	7	...
Sum	0	1	4	9	16	...

Now let's try to answer program planning question 4: How can BASIC and the HP-85 help me find answers?

First off, the number counter is familiar: Initialize C = 0, then within the loop: C = C + 1. But how about the odd integer value? If we initialize I = 0, we can get the first odd integer using I = I + 1. But this would give us 2 for the second value (I = I + 1 = 1 + 1 = 2). The second value should be 3.

Let's try initializing I = 1. Now I = I + 2 works, giving:

Loop 1: I = I + 2 = 1 + 2 = 3

Loop 2: I = I + 2 = 3 + 2 = 5

Loop 3: I = I + 2 = 5 + 2 = 7

and so on.

Now we'll revise the table to show initial values of one instead of zero:

	Initial Values				
Loop No.		1	2	3	...
Number Counter: C	1	2	3	4	...
Odd Integer: I	1	3	5	7	...
Sum: S	1	4	9	16	...

Now we can study this table and write these expressions:

	Initial Value	**Expression Used in Loop to Get Additional Values**
Number Counter	C=1	C=C+1
Odd Integer	I=1	I=I+2
Sum	S=1	S=S+I

Test these expressions by using them to generate the values in the table:

	Initial Values					**General Expression**
Loop No.		1	2	3	...	
Number Counter C	1	C=C+1 =1+1 =2	C=C+1 =2+1 =3	C=C+1 =3+1 =4	...	C=C+1
Odd Integer I	1	I=I+2 =1+2 =3	I=I+2 =3+2 =5	I=I+2 =5+2 =7	...	I=I+2
Sum S	1	S=S+I =1+3 =4	S=S+I =4+5 =9	S=S+I =9+7 =16	...	S=S+I

Looks like we're on the right track. Now we're ready for the flowchart. Figure 94.

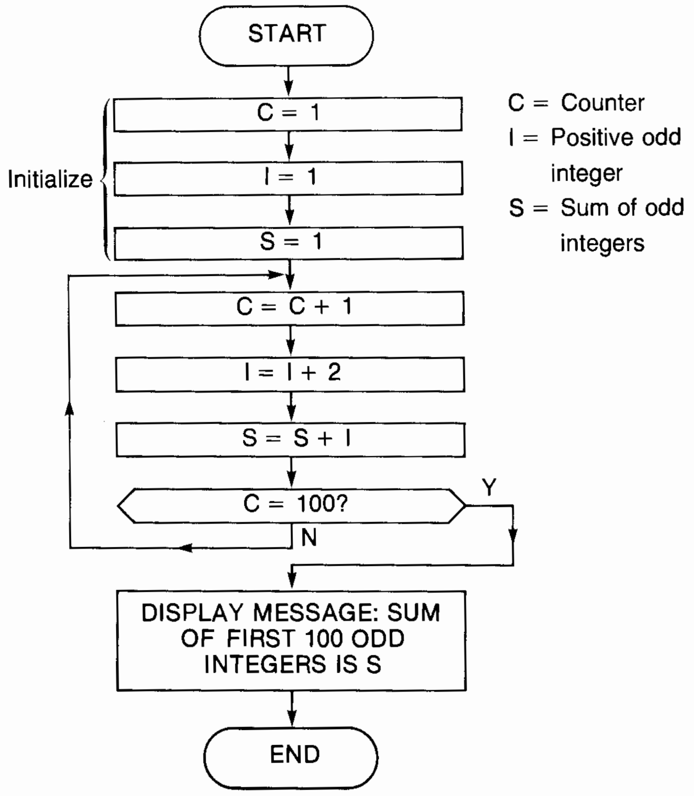
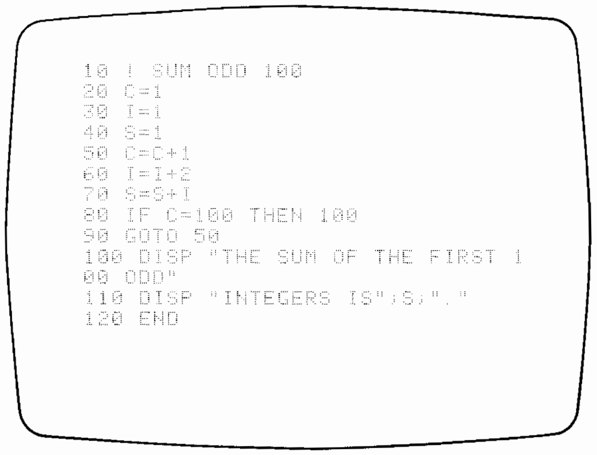


FIGURE 94

Flowchart for "Sum Odd 100"

Now the listing, Figure 95, gives us no big problem. Type it in and run it. You should get 10000.



```
10 ! SUM ODD 100
20 C=1
30 I=1
40 S=1
50 C=C+1
60 I=I+2
70 S=S+I
80 IF C=100 THEN 100
90 GOTO 50
100 DISP "THE SUM OF THE FIRST 1
00 ODD"
110 DISP "INTEGERS IS";S;"."
120 END
```

FIGURE 95

Listing for "Sum Odd 100"

PROBLEM: Write the "100 Odd Sums" Program

Modify "Sum Odd 100" to display each of the intermediate 99 sums as well as the final sum. Since this modified program uses the same variable value vs. loop number table, you've got a good head start.

Draw a flowchart. I show one way to draw it on page H-38.

Now write and run your program. See page H-38 for a listing based on my flowchart. Page H-39 shows the output.

PROBLEM: Write the "Every Ten Times" Program

Program Description Display the sum of this series every 10 terms:

$$\frac{2}{1*3} + \frac{2}{3*5} + \frac{2}{5*7} + \frac{2}{7*9} + \ldots + \frac{2}{A*(A+2)} + \ldots$$

That is, after your program forms and evaluates each fraction in this series, it should calculate the sum. The only sums that should be displayed are the 10th, 20th, 30th and so on.

The program should continue until the PAUSE key is pressed.

Initialize your program with some friendly statements so the HP-85 will think clearly and normally. Of course, you'll need some additional initializing statements to give your variables initial values.

Hints

1. Use the general term $\frac{2}{A*(A+2)}$ to help you construct your variable value vs. loop number table. Notice that only one variable is used. If you could figure out how the value of A changes from one term to the next, your table would be well under way. Look at the values of A in the first four terms: 1, 3, 5, 7. How is each successive value derived from the last?

2. Here's the beginning of my variable value vs. loop number table:

	Initial Values		
Loop No.		1	...
Term counter C	1	2	...
Denominator variable A	1	3	...
Sum S	$\frac{2}{1*3}$	$\frac{2}{1*3} + \frac{2}{3*5}$	...

3. After every 10th term, your program should display the current sum of all terms. Say the term counter is C. If C/10 = INT(C/10), it would mean the term number was exactly divisible by 10, and your program should display the current sum of all terms. This trick is similar to the one you used in "Sweepstakes" to test if the certificate number was a whole number.

4. The sum of the first two terms is .800. To test your completed program, change it for a moment to display the sum of every second term and see if your program gives .800 for the first sum.

Your Turn Now draw your flowchart and compare it with mine on page H-39.

Finally, write your program. You may stop your programming and continue hours or days later. Simply store your unfinished program temporarily on your BASIC Training cartridge using the name "WORK."

My listing and output for "Every Ten Times" is on page H-39.

Summary of Chapter 12

- **A fundamental counting routine of BASIC** using a loop is:

```
10 A=1
20 DISP A
30 A=A+1
40 GOTO 20
50 END
```

  Reference: Page 190.

- **Test your program with an example whose answer you know.** Reference: Pages 12-193–12-195.

- **Table of variable values vs. loop numbers:** It helps you to understand and to check a program that uses a loop. Reference: Pages 12-196–12-197.

Review Test for Chapter 12

The answers are on page 12-202, immediately following this review test.

1. What, if anything, is wrong with each of these programs that would cause it to generate an error or warning or otherwise cause it not to count 1, 2, 3, 4, 5, ... ? Resist the temptation to enter these into the HP-85 and let it do your work for you. However, once you have your answer, why not let the HP-85 check it for you?

a.

```
10 A=A+1
20 DISP A
30 WAIT 500
40 GOTO 10
50 END
```

b.

```
10 GOTO 30
20 DISP A
25 GOTO 40
30 A=0
40 A=A+1
45 WAIT 500
50 GOTO 20
60 END
```

2. a. Write a general expression for the sum S for the following series. N is the term number.

$$\text{Series: } 1 + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \frac{9}{5} + \frac{11}{6} + \ldots + \frac{2\,N - 1}{N} + \ldots$$

b. Next, write a program based on this series that displays the sum of any chosen number of terms M. Have your program ask the user to enter M.

This variable value vs. loop number table may be of help:

	Initial Value						General Expressions
Loop No.		1	2	3	4	...	
Term No. N	1	2	3	4	5	...	N
Term T	1	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{7}{4}$	$\frac{9}{5}$	...	(2*N−1)/N
Sum S	1	$\frac{5}{2}$	$\frac{25}{6}$	$\frac{71}{12}$	$\frac{463}{60}$	...	

Answers to Review Test Questions for Chapter 12

1. a. The variable A is not initialized. The program would count if run, but a warning message,

```
Warning 7 on line 10 : NULL DATA
```

would be displayed before counting began. One cure would be to add this statement:

```
5 A=0
```

b. This is a rather fat program, but it counts correctly without any warning message.

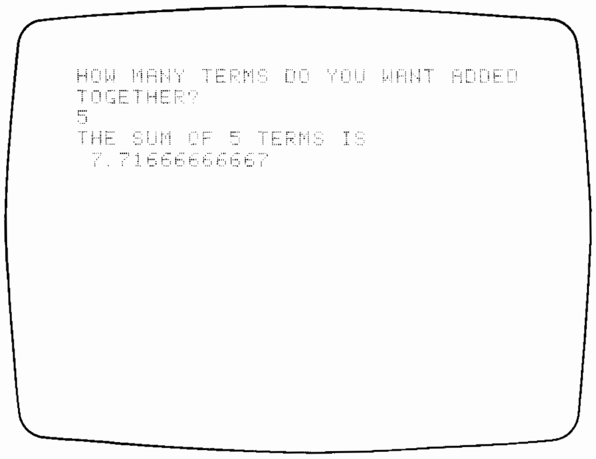
Reference: Page 190.

2. a. `S=S+(2*N-1)/N`

b. Here is one way to write the program:

```
10 ! REVIEW TEST FOR CHAPTER 12
    PROBLEM 2
20 CRT IS 1
30 CLEAR
40 NORMAL
50 DISP "HOW MANY TERMS DO YOU
   WANT ADDEDTOGETHER";
60 INPUT M
70 N=1
 80 S=1
 90 IF M=N THEN 130
100 N=N+1
110 S=S+(2*N-1)/N
120 GOTO 90
130 DISP "THE SUM OF";M;"TERMS I
    S";S;" "
140 END
```

Here is a typical output for this program:



```
HOW MANY TERMS DO YOU WANT ADDED
TOGETHER?
5
THE SUM OF 5 TERMS IS
 7.71666666667
```

Reference: Pages 12-196–12-199.

Chapter 13

Teach Your Program to Count Without Using its Fingers

Preview

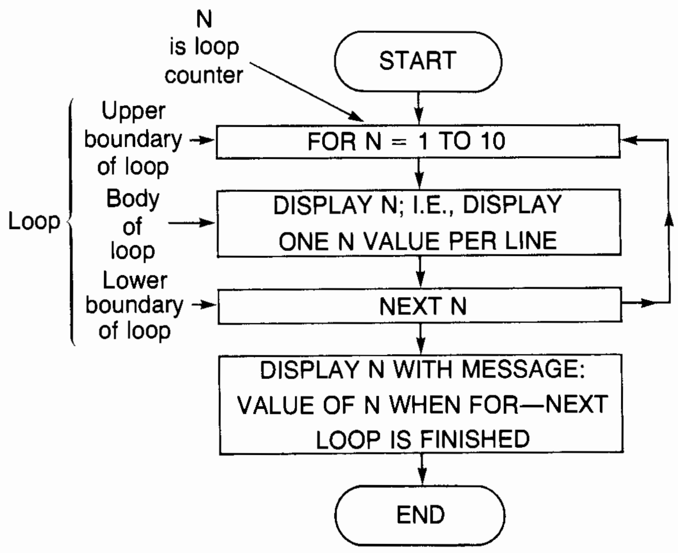
In Chapter 13, you will learn:

- How real programmers tell their programs to count by using a FOR—NEXT loop.
- About another trick the little semicolon can perform.
- How to make sure a number is optimistic, with a positive outlook.
- How to find program bugs.

FOR—NEXT, STEP: A BASIC Word

Using FOR—NEXT, STEP is the way to create and control a loon with more elegance and power than using IF ... THEN and GOTO.

EXAMPLE: "Count to Ten A" Program



N is loop counter
START
Upper boundary of loop
FOR N = 1 TO 10
Loop
Body of loop
DISPLAY N; I.E., DISPLAY ONE N VALUE PER LINE
Lower boundary of loop
NEXT N
DISPLAY N WITH MESSAGE: VALUE OF N WHEN FOR—NEXT LOOP IS FINISHED
END

FIGURE 96

Flowchart for "Count to Ten A"

On the flowchart for this example, Figure 96, the FOR—NEXT loop, loop body, and the upper and lower loop boundaries are indicated. They are also shown in Figure 97, which gives the listing. Note that N is the loop counter.

```
                 10 ! COUNT TO TEN A
               { 20 FOR N=1 TO 10◄──────────── Upper loop boundary
FOR—NEXT loop  { 30 DISP N◄─────────────────── Loop body
               { 40 NEXT N◄─────────────────── Lower loop boundary
                 50 DISP "WHEN THE FOR - - NEXT
                    LOOP IS"
                 60 DISP "FINISHED, N =";N;"."
                 70 END
```

FIGURE 97

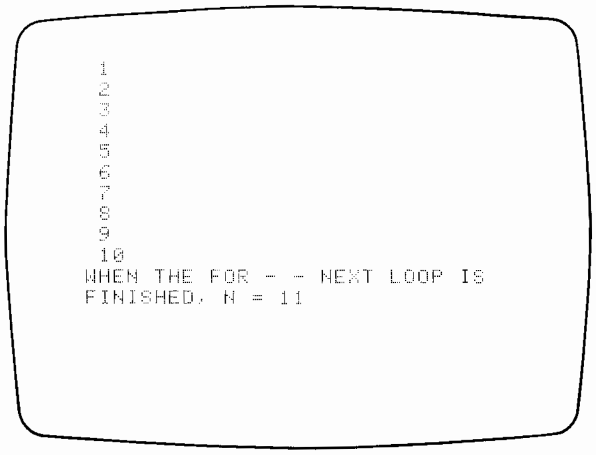
Listing for "Count to Ten A"

This loop body is as small as it gets, only one line, but it exhibits a feature shared with many loops. The loop counter, N, is used within this loop. Many other FOR—NEXT loops do not use the loop counter within the loop.

The boundries of a loop are always the FOR ... TO and the NEXT statements.

Enter this "Count to Ten A" program and run it. Notice a very important point. *The value of the loop counter when the loop is finished is 11, not 10.*

Your "Count to Ten A" program should produce the output shown in Figure 98.



```
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
WHEN THE FOR - - NEXT LOOP IS
FINISHED, N = 11
```

FIGURE 98

Output of "Count to Ten A"

Step-By-Step Analysis of "Count to Ten A"

Because FOR—NEXT is such an important and widely used tool, it will pay to understand its operation thoroughly before continuing. Let's start at the beginning, statement 10, which is simply a remark.

```
10 ! COUNT TO TEN A

20 FOR N=1 TO 10
```

Statement 20 defines the loop counter variable name. Also defined are the values this counter will take during loop execution. If no STEP appears in the FOR ... TO statement, a step of +1 is understood. This means the value of N is advanced by one each time NEXT is executed. At this point, N is assigned the first value, 1.

```
30 DISP N
```

The current value of N is displayed, which is 1.

```
40 NEXT N
```

Now the value of N is changed to the next value determined by statement 20. In this case, the next value is 1 higher than its present value. So now, N = 1 + 1 = 2. At this point, the question is automatically asked, "Is N greater than 10?" Since the answer is NO, the body of the loop is executed again.

```
30 DISP N
```

Now 2 is displayed.

```
40 NEXT N
```

Again, N is increased by 1, and again, N's new value, 3, is not greater than 10.

```
30 DISP N
```

This process continues. Let's pick it up later, after N has reached the value 9.

```
30 DISP N
```

The screen shows 9, the current value of N.

```
40 NEXT N
```

Now N is incremented by 1 to reach 10. Since 10 is not *greater* than 10, the body of the loop is again executed.

```
30 DISP N
```

This value of N, 10, is displayed.

```
40 NEXT N
```

N is increased again by 1, making N = 11. Now the same question is asked: "Is N greater than 10?" Since 11 *is* greater than 10, the answer is YES, and program execution moves on to the next two statements.

```
50 DISP "WHEN THE FOR - - NEXT
   LOOP IS"
60 DISP "FINISHED, N =";N;"."
```

The message contained in these statements is displayed, telling us that N currently does have the value 11.

Now there is only one more statement to be executed:

```
70 END
```

EXAMPLE: "Count to Ten B" Program

Let's modify "Count to Ten A" by adding a DISP statement to the body of the loop. Figure 99 shows the new flowchart, and Figure 100 gives the new listing. Run your "Count to Ten B" program. You should get the output shown in Figure 101. Each time through the loop, the DISP statement causes a blank line to appear on the screen. As a result, each of the displayed numbers is separated from its neighbor by a blank line.

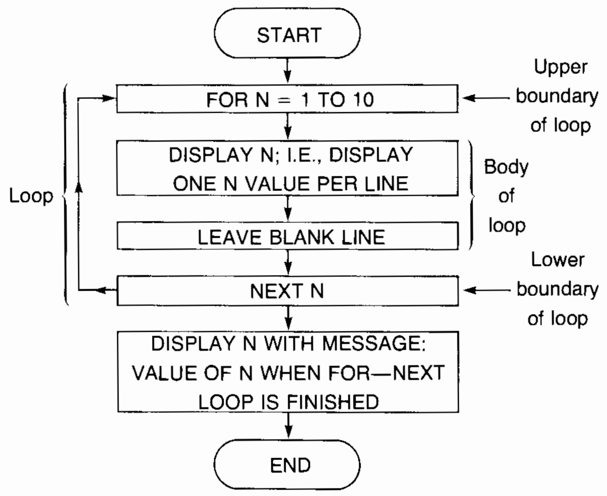

START
FOR N = 1 TO 10
Upper boundary of loop
DISPLAY N; I.E., DISPLAY ONE N VALUE PER LINE
Body of loop
Loop
LEAVE BLANK LINE
NEXT N
Lower boundary of loop
DISPLAY N WITH MESSAGE: VALUE OF N WHEN FOR—NEXT LOOP IS FINISHED
END

FIGURE 99
Flowchart for "Count to Ten B"

```
10 ! COUNT TO TEN B
20 FOR N=1 TO 10
30 DISP N
35 DISP
40 NEXT N
50 DISP "WHEN THE FOR - - NEXT
   LOOP IS"
60 DISP "FINISHED, N =";N;"."
70 END
```

FIGURE 100
Listing for "Count to Ten B"

```
  1

  2

  3

  4

  5

  6

  7

  8

  9

  10

WHEN THE FOR - - NEXT LOOP IS
FINISHED, N = 11
```

FIGURE 101

PRINTALL output for "Count to Ten B"

EXAMPLE: "Count to Ten C" Program

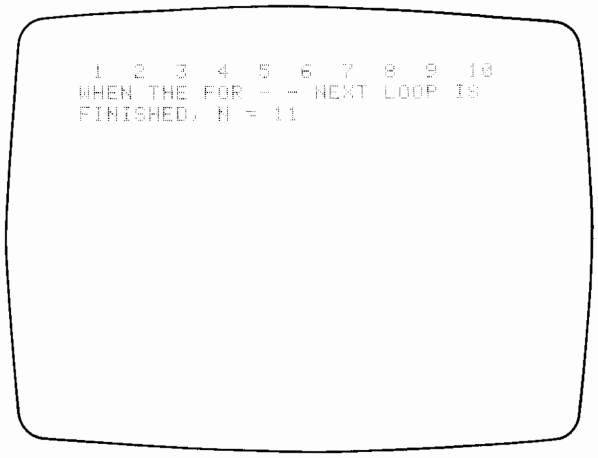
Now remove line 35 DISP from your ''Count to Ten B'' program, and run an experiment. Add a semicolon after the N of statement 30. This will give the listing in Figure 102 (except for line 10).

Run your revised program. You should get the output shown in Figure 103.

```
10 ! COUNT TO TEN C
20 FOR N=1 TO 10
30 DISP N;
40 NEXT N
50 DISP "WHEN THE FOR - - NEXT
   LOOP IS"
60 DISP "FINISHED, N =";N;"."
70 END
```

FIGURE 102

Listing for "Count to Ten C"



```
 1  2  3  4  5  6  7  8  9  10
WHEN THE FOR - - NEXT LOOP IS
FINISHED, N = 11
```

FIGURE 103

Output for "Count to Ten C"

More Semicolon Power

How did the "Count to Ten C" output happen? The lowly semicolon has another important power in BASIC besides causing close spacing between items in a `DISP` or `PRINT` statement, and moving the question mark of an `INPUT` statement.

When a semicolon is added at the end of statement 30, the value of `N` is *not* displayed each time the loop is executed. The semicolon says to the `DISP N` instruction: "Don't display that value! Put it in storage until either

1. Your storeroom has 32 characters in it, or
2. Another `DISP` statement is executed that has no final semicolon.

"If either of these happens, then display what you have stored. If reason 2 causes you to empty your storeroom, be sure to display your stored characters before the next `DISP` statement displays its message." The `DISP N` instruction has learned to obey the small but mighty semicolon, so it waits until line 50 is executed before displaying its stored values.

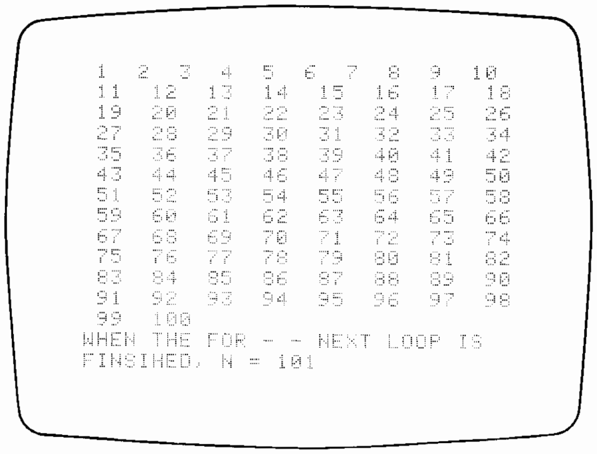
EXAMPLE: "Count to 100" Program

Run another experiment. Start by making another small change in your program. Add one zero to statement 20 of your "Count to Ten C" program. Instead of `FOR N=1 TO 10`, change it to read `FOR N=1 TO 100`. Make sure statement 30 hangs on to its semicolon. Now your listing looks like Figure 104.(Except maybe you didn't change statement 10. If you didn't you're excused).

Run this latest modification and see on the screen the output shown in Figure 105.

```
10 ! COUNT TO 100
20 FOR N=1 TO 100
30 DISP N;
40 NEXT N
50 DISP "WHEN THE FOR - - NEXT
   LOOP IS"
60 DISP "FINISHED, N =";N;"."
70 END
```

FIGURE 104
Listing for "Count to 100"



```
 1  2  3  4  5  6  7  8  9  10
 11  12  13  14  15  16  17  18
 19  20  21  22  23  24  25  26
 27  28  29  30  31  32  33  34
 35  36  37  38  39  40  41  42
 43  44  45  46  47  48  49  50
 51  52  53  54  55  56  57  58
 59  60  61  62  63  64  65  66
 67  68  69  70  71  72  73  74
 75  76  77  78  79  80  81  82
 83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98
 99  100
WHEN THE FOR - - NEXT LOOP IS
FINSIHED, N = 101
```

FIGURE 105
Output for "Count to 100"

Each full line of displayed numbers shown in Figure 105 was displayed because the semicolon storehouse (called a "buffer" in the computer world) became filled repeatedly with 32 characters. The last two displayed numbers, 99 100, were displayed because another DISP statement, line 50, was executed that has no final semicolon. Try deleting the two final DISP statements, lines 50 and 60, and then run your "Count to 100" program again. You'll see all numbers from 1 through 98 displayed. When the program ends, the semicolon storeroom will still hold the characters: 99 100. There they will die of neglect, since no DISP statement came along to rescue them.

PROBLEM: Work with your "MONEY" Program

Remember that "MONEY" program you stored on your BASIC Training cartridge way back in Chapter 5? Now is the time to execute LOAD"MONEY" and to display a listing.

I'd like you to make a table of loop counter values vs. loop variable values for "MONEY." Continue the table through S = 10, and write the general expressions. Here is how the table looks for the first three values of S:

					General Expressions
Loop counter S	1	2	3	...	
Square number S	1	2	3	...	
Number of dollars D	1	2	4	...	

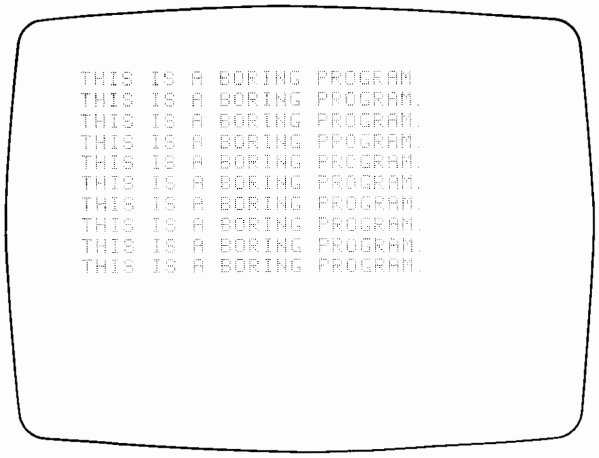
The table continued through S = 10, plus the general expressions, are on page H-40.

"MONEY" is a simple program, and this is a simple table. However, the construction of such tables can help a programmer write a long and complex FOR—NEXT loop.

EXAMPLE: "Boredom" Program

The listing and output are shown in Figure 106. This shows a FOR—NEXT loop where the loop counter N is *not* used in the loop body. Enter and run this one if you wish.

```
10 ! BOREDOM
20 FOR N=1 TO 10
30 DISP "THIS IS A BORING PROGR
   AM."
40 NEXT N
50 END
```



```
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
THIS IS A BORING PROGRAM.
```

FIGURE 106
Listing and output for "Boredom"

Flexibility of FOR—NEXT Loops

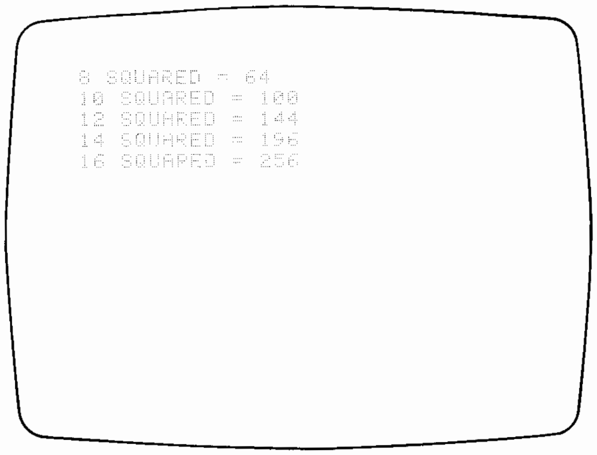
1. They do not have to start counting at 1.
2. They do not have to count one at a time.
3. They can count either forwards or backwards.
4. Variables can be used to define loops.

The following programs will illustrate these strengths of FOR—NEXT loops.

EXAMPLE: "Squares" Program

Figure 107 gives both listing and output, and previews the STEP part of FOR—NEXT, STEP.

```
10 ! SQUARES
20 FOR K=8 TO 16 STEP 2
30 DISP K;"SQUARED =";K^2
40 NEXT K
50 END
```



```
8 SQUARED = 64
10 SQUARED = 100
12 SQUARED = 144
14 SQUARED = 196
16 SQUARED = 256
```

FIGURE 107
Listing and output for "Squares"

The loop counter K in "Squares" starts at 8, and the loop stops when K exceeds 16.

STEP: Part of FOR—NEXT, STEP

If the loop counter is to increase by one each time NEXT is executed, STEP is not required in the FOR ... TO statement. For any loop counter change other than plus one, STEP must be used and defined. In "Squares," STEP 2 appears in the FOR ... TO statement, line 20, causing K to increase by 2 each time NEXT is executed.

ABS(X): A Function

You'll use ABS(X) very soon in a program. Here are some examples showing how ABS(X) works:

```
ABS(3) = 3              ABS(317) = 317
ABS(-5) = 5             ABS(-2) = 2
ABS(+17) = 17           ABS(1) = 1
```

When the argument X is positive, ABS(X), which stands for "absolute value of X," returns a value of positive X. When the argument X is negative, ABS(X) returns a value of positive X. Hard to find a simpler function.

EXAMPLE: "Big Number" Program

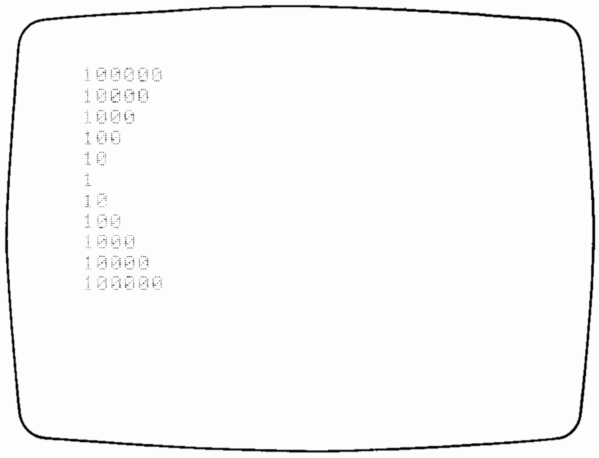
As the "Big Number" program's listing and output, Figure 108, shows, the loop counter N starts at +5, steps down one number each time NEXT is executed, and the program exits the loop when N gets *smaller* than -5. The value of ABS(N) in line 30 is always positive, even when N becomes negative. The effect is to cause ABS(N) to take these values as N goes from +5 to -5:

N	5	4	3	2	1	0	-1	-2	-3	-4	-5
ABS(N)	5	4	3	2	1	0	1	2	3	4	5

Take a good look at the STEP -1 part of line 20. The STEP value should always change the value of the loop counter from its initial value toward its final limit. Sometimes, as here, the STEP value should be negative.

Note: In "Big Number," since the STEP value is negative, NEXT N asks "Is N *less* than -5?"

```
10 ! BIG NUMBER
20 FOR N=5 TO -5 STEP -1
30 DISP 10^ABS(N)
40 NEXT N
50 END
```



```
100000
10000
1000
100
10
1
10
100
1000
10000
100000
```

FIGURE 108

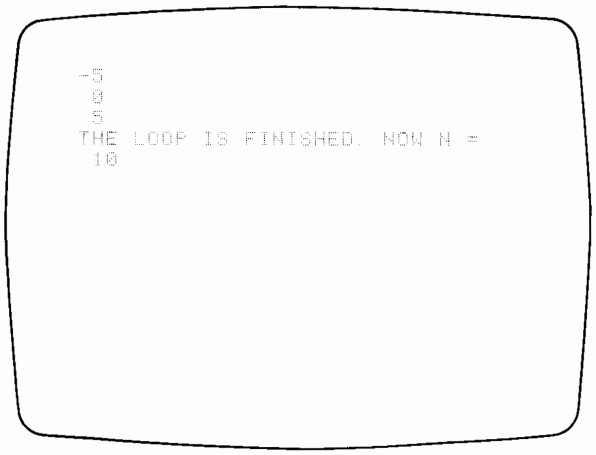
Listing and output for "Big Number"

Here's a modified table of loop counter values vs. loop variable values for "Big Number":

Loop Counter N	10^ABS(N)	**Display**
5	10 ∧ ABS(5) = 10 ∧ 5	100000
4	10 ∧ ABS(4) = 10 ∧ 4	10000
3	10 ∧ ABS(3) = 10 ∧ 3	1000
2	10 ∧ ABS(2) = 10 ∧ 2	100
1	10 ∧ ABS(1) = 10 ∧ 1	10
0	10 ∧ ABS(0) = 10 ∧ 0	1
-1	10 ∧ ABS(-1) = 10 ∧ 1	10
-2	10 ∧ ABS(-2) = 10 ∧ 2	100
-3	10 ∧ ABS(-3) = 10 ∧ 3	1000
-4	10 ∧ ABS(-4) = 10 ∧ 4	10000
-5	10 ∧ ABS(-5) = 10 ∧ 5	100000

EXAMPLE: "Big Step" Program

```
100 ! BIG STEP
150 FOR N=-5 TO 8 STEP 5
200 DISP N
250 NEXT N
300 DISP "THE LOOP IS FINISHED.
    NOW N =";N;" "
350 END
```



```
-5
 0
 5
THE LOOP IS FINISHED. NOW N =
 10
```

FIGURE 109
Listing and output for "Big Step"

Figure 109 shows that it is not necessary for the loop counter to assume the limit value. Here, N never assumes the value 8, but the program still runs smoothly. Remember, the question asked by NEXT N, line 250, is this: "Is N *greater* than 8?" The NEXT N statement has no way of knowing if N equals 8 or not.

EXAMPLE: "Rich" Program

This program, Figures 110 and 111, uses a variable to define the final loop counter limit value.

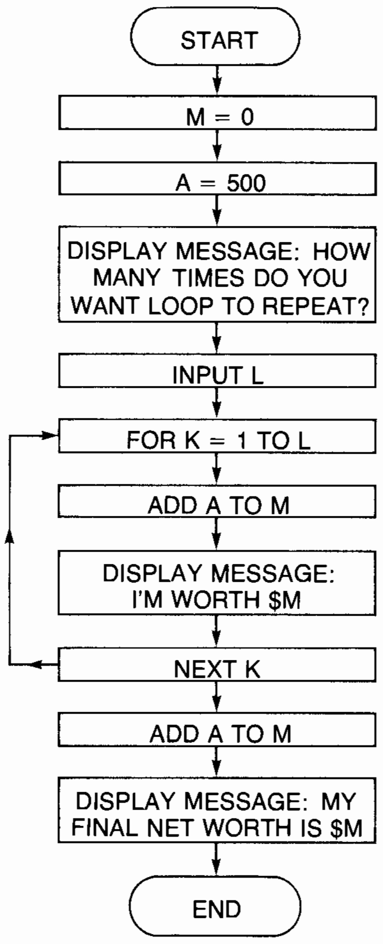

START
M = 0
A = 500
DISPLAY MESSAGE: HOW MANY TIMES DO YOU WANT LOOP TO REPEAT?
INPUT L
FOR K = 1 TO L
ADD A TO M
DISPLAY MESSAGE: I'M WORTH $M
NEXT K
ADD A TO M
DISPLAY MESSAGE: MY FINAL NET WORTH IS $M
END

FIGURE 110

Flowchart for "Rich"

```
 10 ! RICH
 20 M=0
 30 A=500
 40 DISP "HOW MANY TIMES DO YOU
    WANT LOOP"
 50 DISP "TO REPEAT";
 60 INPUT L
 70 FOR K=1 TO L
 80 M=M+A
 90 DISP "I'M WORTH $";M;"."
100 NEXT K
110 M=M+A
120 DISP "MY FINAL NET WORTH IS"
130 DISP "$";M;" "
140 END
```

FIGURE 111

Listing for "Rich"

Enter "Rich" and run it. Become as wealthy as you choose. Keep "Rich" in memory, since you're going to use it soon.

Bugs

You've heard about program bugs. They're the pesky little devils that cause a perfect program to bomb. In spite of massive efforts to wipe these vermin from the face of the earth, they remain epidemic.

We do have effective weapons to use against them, however. I'll introduce you to three of these weapons, a command, a BASIC word and a key, and then I'll show you how to use them on your "Rich" program.

INIT: A Command

The INIT (initialize) command reserves memory for the values of a program's variables. When you execute RUN, initialization is done automatically. INIT is usually executed by pressing SHIFT + INIT), above 9 on the number pad. INIT is an immediate execute key.

TRACE ALL: A BASIC Word

TRACE ALL allows the order of execution of a program's statements to be traced. TRACE ALL also allows the values of all variables in a program to be traced. TRACE ALL is executed as a *command* by typing TRACE ALL, then pressing (END LINE).

The STEP Key

Each time (SHIFT) + (STEP PAUSE) is pressed, the next statement of the program in memory is executed. So (STEP) allows your program to be executed one statement at a time. The main power of (STEP) is realized when it's used with the TRACE ALL command, as described below. The (STEP) key can be used by itself, but there is generally no advantage in doing so.

Bug Chasing with TRACE ALL

If you wanted to expose a bug or two, an effective way would be to execute CRT IS 2, INIT, TRACE ALL, and then press (STEP) repeatedly.

If your program included a NORMAL statement and/or a CRT IS 2 statement, you would convert them temporarily into remarks by inserting a ! just after their line numbers.

Using TRACE ALL in "Rich"

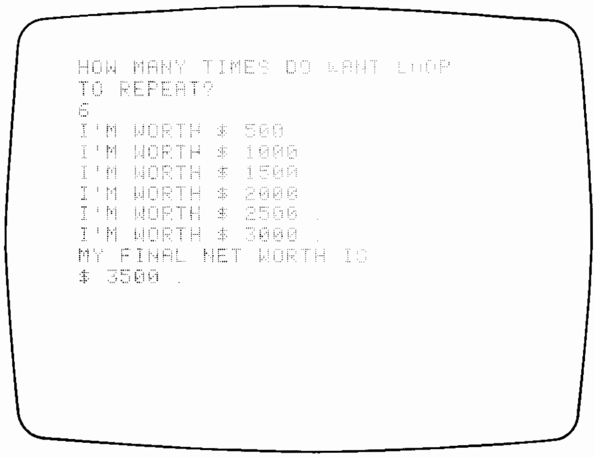
Practice using this bug spray on your "Rich" program.

1. Execute the CRT IS 2 command. This is desirable, since all TRACE ALL outputs are printed, not displayed. CRT IS 2 will put your displayed messages on the printer, and will result in a more understandable TRACE ALL record.
2. Execute the INIT command by pressing (SHIFT) + (INIT]).
3. Execute TRACE ALL by typing TRACE ALL and pressing (END LINE).
4. Step through "Rich," one line at a time, by holding down (SHIFT) and pressing (STEP PAUSE) repeatedly. After pressing (STEP), wait until the execution of the statement and its associated TRACE ALL output is complete before pressing (STEP) again. Watch your printout. When TO REPEAT? is printed, press (6)(END LINE) to respond to the input request. The 6 will be displayed, not printed. Continue pressing (STEP). When you press (STEP) and nothing is printed you've finished the program.

Figure 113 shows the printout you should get. The listing and the normal program output are shown in Figure 112 for reference.

If "Rich" had a bug, you could find its hiding place by comparing each statement's TRACE ALL record as it appeared with your knowledge of what the program should do at that statement.

```
 10 ! RICH
 20 M=0
 30 A=500
 40 DISP "HOW MANY TIMES DO YOU
    WANT LOOP"
 50 DISP "TO REPEAT";
 60 INPUT L
 70 FOR K=1 TO L
 80 M=M+A
 90 DISP "I'M WORTH $";M;"."
100 NEXT K
110 M=M+A
120 DISP "MY FINAL NET WORTH IS"
130 DISP "$";M;"."
140 END
```



```
HOW MANY TIMES DO WANT LOOP
TO REPEAT?
6
I'M WORTH $ 500
I'M WORTH $ 1000
I'M WORTH $ 1500
I'M WORTH $ 2000
I'M WORTH $ 2500 .
I'M WORTH $ 3000 .
MY FINAL NET WORTH IS
$ 3500 .
```

FIGURE 112

Listing and output of "Rich" shown to clarify `TRACE ALL` output (shown below)

```
CRT IS 2
TRACE ALL
```

This line displayed only.

INIT pressed here.

This line displayed and printed.

STEP pressed here and pressed again after every line is printed, except after `TO REPEAT?`

```
 Trace line 10 to 20
 Trace line 20 M=0
 Trace line 20 to 30
 Trace line 30 A=500
 Trace line 30 to 40
HOW MANY TIMES DO YOU WANT LOOP
 Trace line 40 to 50
 Trace line 50 to 60
TO REPEAT?
 Trace line 60 L=6
 Trace line 60 to 70
 Trace line 70 K=1
 Trace line 70 to 80
 Trace line 80 M=500
 Trace line 80 to 90
I'M WORTH $ 500
 Trace line 90 to 100
 Trace line 100 K=2
 Trace line 100 to 80
 Trace line 80 M=1000
 Trace line 80 to 90
```

6 END LINE pressed here. Then STEP pressed.

```
I'M WORTH $ 1000
 Trace line 90 to 100
 Trace line 100 K=3
 Trace line 100 to 80
 Trace line 80 M=1500
 Trace line 80 to 90
I'M WORTH $ 1500
 Trace line 90 to 100
 Trace line 100 K=4
 Trace line 100 to 80
 Trace line 80 M=2000
 Trace line 80 to 90
I'M WORTH $ 2000
 Trace line 90 to 100
 Trace line 100 K=5
 Trace line 100 to 80
 Trace line 80 M=2500
 Trace line 80 to 90
I'M WORTH $ 2500
 Trace line 90 to 100
 Trace line 100 K=6
 Trace line 100 to 80
 Trace line 80 M=3000
 Trace line 80 to 90
I'M WORTH $ 3000
 Trace line 90 to 100
 Trace line 100 K=7
 Trace line 100 to 110
 Trace line 110 M=3500
 Trace line 110 to 120
MY FINAL NET WORTH IS
 Trace line 120 to 130
$ 3500
 Trace line 130 to 140
```

When STEP pressed here, no printout occurred, showing that program was finished.

FIGURE 113

Action of TRACE ALL and STEP on "Rich"

Another effective way to use TRACE ALL is to have the TRACE ALL printout proceed continuously until END is reached or until you press PAUSE. You use TRACE ALL this way by executing these commands in this order: CRT IS 2, TRACE ALL, and then pressing RUN.

When TRACE ALL is executed by itself in this way, the output is the same as when TRACE ALL and STEP are used together, except the printout is continuous. Using STEP gives you more control, although the output is slower.

TRACE: A BASIC Word

TRACE VAR: A BASIC Word

TRACE ALL includes the actions of TRACE and TRACE VAR. TRACE prints whenever statements are *not* executed sequentially, as with GOTO, FOR—NEXT and IF ... THEN. The two numbers are printed whose lines are executed out of sequence. TRACE VAR is entered with a trace variable list, like TRACE VAR A,C7,M2. Every time a variable in this list changes value, the line number, variable name and new value are printed.

To cancel the action of TRACE, TRACE VAR and TRACE ALL, execute the NORMAL or SCRATCH commands, press (RESET), or use NORMAL as a statement in your program.

If you wish to trace a certain segment of your program, insert temporarily a TRACE, TRACE VAR or TRACE ALL statement at the beginning of the segment, and temporarily insert a NORMAL statement at the end of the segment. Then run your program in the usual way. Only the program segment of interest will be traced. Don't forget to delete the TRACE, TRACE VAR or TRACE ALL and the NORMAL statements when you're through with them.

Summary of Chapter 13

- FOR—NEXT, STEP: **A BASIC word**

  General form:

  [line number] FOR [loop counter variable] = [initial value of loop counter variable] TO [final limit of loop counter variable] STEP [amount loop counter variable changes each time loop executes]

  If the STEP value is +1, STEP [ ] may be omitted.

  Two or more lines below FOR[ ] =[ ] TO[ ] STEP[ ], the last part of FOR—NEXT, STEP appears:

  [line number] NEXT [loop counter variable]

  Example:

```
:
150 FOR C=3 TO 10 STEP 2
160 ! THIS IS THE
170 ! BODY OF THIS
180 ! FOR - - NEXT
190 ! LOOP
200 NEXT C
210 ...
:
```

Loop counter variable: C

Initial value of loop counter variable: 3

Final limit of loop counter variable: 10

Amount loop counter variable changes each time loop executes: 2

Values loop counter variable assumes as `FOR`—`NEXT` loop executes: 3, 5, 7, 9. Notice that the loop counter variable need not assume the value of the upper limit.

When program execution reaches `NEXT C`, two things happen:

1. First, C is changed by +2. (If `STEP` were absent, C would increase by 1).
2. Then the program asks itself this question: "Is C greater than 10?"

   If the answer is *no,* program execution continues with line 160.

   If the answer is *yes,* program execution continues with line 210, and the value of the loop counter variable C is now 11.

Note: For a negative step value, say `FOR C=10 TO 3 STEP -2`, the question asked by `NEXT C` is: "Is C *less* than 3?"

Reference: Pages 204-13-214.

- **Semicolon suppresses action of `DISP` or `PRINT` statement.**

When a semicolon is outside the final quotation mark of a `DISP` or `PRINT` statement, the message is not displayed or printed. Instead, the characters of the message are stored. These stored characters are displayed or printed if either:

1. The number of characters stored reaches 32, or
2. Another `DISP` or `PRINT` statement is executed that has no final semicolon.

If reason 2 empties the storeroom (empties the buffer), the stored characters are displayed or printed before the other `DISP` or `PRINT` statement displays or prints its message.

Reference: Pages 13-209–13-210.

- `ABS (X)`**: A function**

Gives the absolute value of X.

Reference: Pages 13-211–13-212.

- `INIT`: **A command**

`INIT` reserves memory for the values of a program's variables. It is immediately executed by pressing SHIFT + INIT), above the 9 key in the number pad. `INIT` is often used with `TRACE`, `TRACE VAR` and `TRACE ALL`.

Reference: Page 13-214.

- **The STEP key**

Pressing SHIFT + STEP/PAUSE executes the first statement of a program. Pressing STEP again executes the next statement, and so on. For STEP to function, the program must first be initialized. This can be done with INIT.

Reference: Page 13-215.

- TRACE**: A BASIC word**

General form:

[line number] TRACE

TRACE is often used as a command. After TRACE is executed, it remains in force until it is cancelled with NORMAL, RESET or SCRATCH. When TRACE is in force, and a program is executed with RUN or STEP, a printout occurs whenever a program statement is *not* executed sequentially. When a program branching does occur, two line numbers are printed. These numbers belong to the lines executed before and after the branching.

Reference: Page 13-218.

- TRACE VAR**: A BASIC word**

General form:

[line number] TRACE VAR [variable list]

Example:

```
255 TRACE VAR B,W3,K,L
```

TRACE VAR is often used as a command. After TRACE VAR is executed, it remains in force until it is cancelled with NORMAL, RESET or SCRATCH. When TRACE VAR is in force, and a program is executed with RUN or STEP, a printout occurs whenever a variable in the variable list changes value. When the value changes, the line number, variable name and new value are printed.

Reference: Page 13-218.

- TRACE ALL**: A BASIC word**

General form:

[line number] TRACE ALL

`TRACE ALL` is often used as a command. After `TRACE ALL` is executed, it remains in force until it is cancelled with `NORMAL`, RESET or `SCRATCH`. Executing `TRACE ALL` is equivalent to executing `TRACE` and `TRACE VAR`, where the `TRACE VAR` variable list includes all the program's variables.

Reference: Page 13-215.

- **Powerful bug finder**

With program in memory, execute or press:

```
CRT IS 2
INIT
TRACE ALL
```

STEP (repeatedly) to execute each statement, one at a time.

Or, to give continuous printout, execute:

```
CRT IS 2
TRACE ALL
RUN
```

Press PAUSE to stop printout.

Reference: Pages 13-215–13-217.

Review Test for Chapter 13

The answers are on page 13-223 immediately following this review test.

1. What value will A have when each `FOR—NEXT` loop is finished?

a.
```
:
50 FOR A=3 TO 18 STEP 5
:
80 NEXT A
:
```

b.
```
:
50 FOR A=4 TO 12
:
80 NEXT A
:
```

c.
```
:
50 FOR A=-4 TO 9 STEP 3
:
80 NEXT A
:
```

d.
```
:
50 FOR A=4 TO -4 STEP -1
:
80 NEXT A
:
```

2. What, if anything, is wrong with this FOR—NEXT example?

```
:
50 FOR A=-4 TO -10
:
80 NEXT A
:
```

3. How many times will each FOR—NEXT loop be executed when its program is run?

a.

```
:
50 FOR A=3 TO 13 STEP 3
:
80 NEXT A
:
```

b.

```
:
50 FOR A=-3 TO -13 STEP -3
:
80 NEXT A
:
```

c.

```
:
50 FOR A=10 TO 14 STEP 6
:
80 NEXT A
:
```

d.

```
:
50 FOR A=-11 TO 10 STEP 7
:
80 NEXT A
:
```

4. Without entering these programs into the HP-85 (except to check your answers), what will each program display? On how many lines?

a.

```
10 FOR A=8 TO -3 STEP -3
20 DISP A;
30 NEXT A
40 END
```

b.

```
10 FOR A=8 TO -3 STEP -3
20 DISP A
30 NEXT A
40 END
```

Answers to Review Test Questions for Chapter 13

1. a. 23 Reference: Page 13-211.

   b. 13 Reference: Pages 13-204–13-207.

   c. 11 Reference: Page 13-213.

   d. -5 Reference: Page 13-212.

2. Line 50 should show a negative `STEP`. Since no `STEP` appears in line 50, the loop counter variable A will try to change +1 each time through the loop; that is, from −4 to (−4 + 1) = −3, −2, −1, etc. If the HP-85 allowed this to happen, the loop variable would never reach -10. Fortunately, the HP-85 is smart enought to avoid this trap. If a program like this were run, the first execution of line 50 would cause a branching to the first line following line 80.

   Reference: Page 13-212.

3. a. 4

   b. 4

   c. 1

   d. 4

   Reference: Pages 13-211–13-214.

4. a. Nothing is displayed. Since the semicolon storeroom (buffer) was not filled with 32 characters when the program ended, and since no later `DISP` statement emptied the storeroom, the 8, 5, 2 and −1 characters died.

   Reference: Pages 13-209–13-210.

   b.

```
 8
 5
 2
-1
```

   The program displays these numbers on 4 lines.

   Reference: Pages 13-204–13-207, 13-213.

Chapter 14

More Fingerless Counting With a Touch of Math

Preview

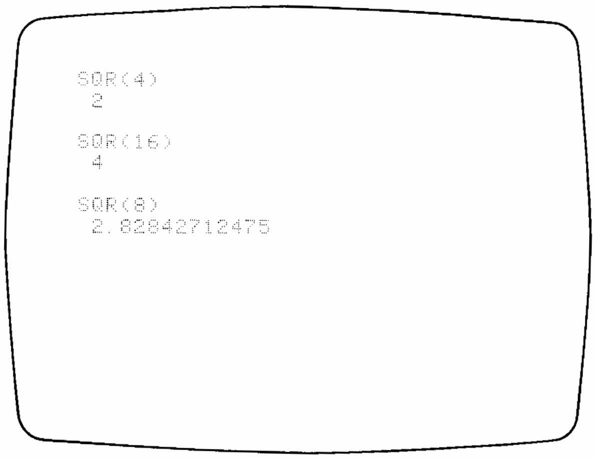
In Chapter 14, you will:

- Become casually acquainted with three more math functions and three more BASIC words about math.
- Write three FOR — NEXT math programs you can write without knowing hardly any math.
- See a forbidden FOR—NEXT performance.
- Learn about a FOR — NEXT with branches.

SQR(X): A Function

This function returns the square root of the argument X.

Examples:



```
SQR(4)
 2

SQR(16)
 4

SQR(8)
 2.82842712475
```

LGT(X): A Function

This returns the common logarithm of X, which is the logarithm to the base 10 of X.

If you don't know what a logarithm is, don't worry. You don't have to know what it is to use it with ease in this course. All you have to know is:

LGT(X) is a function that gives the common logarithm of X.

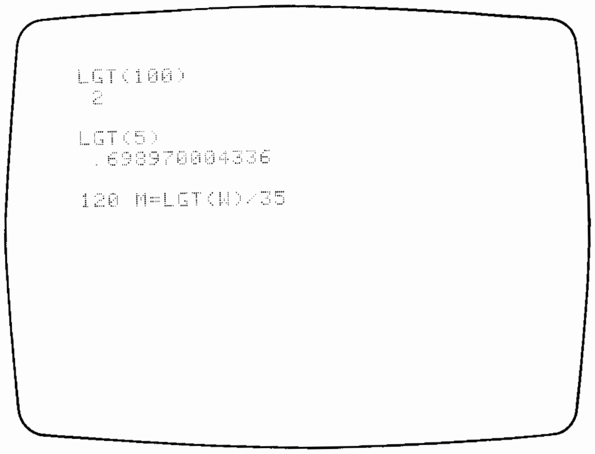
The rest of this brief discussion on logarithms is optional, except you should look at the examples.

Truths About Logarithms and Exponents

This Expression	Is Equivalent to This Expression
If 10 ∧ N equals X then LGT(X) equals N	If $10^N = X$ then $\mathrm{LOG}_{10}X = N$

These truths can be reduced to one sentence: A logarithm is an exponent to which the base must be raised to equal the argument.

Examples:



```
LGT(100)
 2

LGT(5)
 .698970004336

120 M=LGT(W)/35
```

SIN(X): A Function

This is the trigonometric function sine of X.

The HP-85 Wakes Up in Radians

To comply with industry standards (American National Standards Institute or ANSI), the normal unit used by the HP-85 to measure angles is the radian.

There are 2π radians in a circle, or in BASIC language, there are `2*PI` radians in a circle.

`PI` is the ratio of the circumference of a circle to its diameter, and is approximately equal to 3.14159265359... .

You won't use `PI` in this course, but since I'm on the subject, to type the above approximation of `PI` into the HP-85, simply press (P) (I). When you press (END LINE) at the end of your expression or statement, you've entered all twelve digits and the decimal point (as well as the rest of the expression or statement). To see the value of `PI` displayed immediately, press (P) (I) (END LINE).

OK, back to angles. To instruct the HP-85 to think in degrees or grads, use the following BASIC words in statements, or use them as commands.

DEG: A BASIC Word

DEG selects degrees mode. There are 360 degrees in a circle.

GRAD: A BASIC Word

GRAD selects grads mode. There are 400 grads in a circle.

If you plan to measure angles in degrees in your program, be sure to use DEG as one of your initializing statements. Otherwise, your results will be rather strange. Say you forgot to use a DEG statement in your program, and you also did not execute the DEG command. Responding to an INPUT statement in your program, you enter 90, thinking you are entering a right angle. Not so. What you did enter was 90 radians, or about 5156.62 degrees.

Your error would be even greater if you measure angles in grads, and forget to use a GRAD statement or command. Ninety radians is about 5729.58 grads.

Forgetting that the HP-85 wakes up in radians is a familiar error. Watch out for it.

If you're in degrees or grads mode, and want radians mode, use:

RAD: A BASIC Word

RAD puts the HP-85 into radians mode.

DEG, GRAD and RAD are often used as commands.

Here are some examples of SIN(X), GRAD, DEG and RAD in action:

Program	**Display When Run**

`10 GRAD`
`20 DISP SIN(90)`
`30 END`
	`.987688340595`

`10 DEG`
`20 DISP SIN(90)`
`30 END`
	`1`

`10 RAD`
`20 DISP SIN(90)`
`30 END`
	`.893996663601`

SIN(X), like all functions and simple variables, represents a single number. So it may be used anywhere a simple variable may be used, as in this example:

```
70 FOR N=1 TO SIN(A) STEP .1
```

PROBLEM: Write the "Roots" Program

Program Description "Roots" displays titles for two columns, then displays the numbers 1 through 9 in the first column and the nine square roots of those numbers in the second column.

Hints

1. Display your titles before you begin the FOR — NEXT loop that displays the number and square root. See my flowchart on page H-40 if you'd like a little extra help.

2. A way to separate words or numbers on a line, as you will wish to do in "Roots" to create two columns, is to "display" spaces. Consider this program:

Program	**Display When Run**

`120 DISP "A▲▲▲▲B"`
`130 END`
	`A▲▲▲▲B`

When it's run, four spaces are "displayed" between the A and the B.

Let me preview something I'll throw at you formally in Chapter 16. Consider another example where numeric variables are used instead of quoted characters:

Program	**Display When Run**

`100 A=4`
`110 B=4^2`
`120 DISP A;"▲▲▲▲";B`
`130 END`
	`4▲▲▲▲▲▲16`

Running this example gives *six* spaces between the two numbers, even though only four spaces are enclosed in quotes in line 120. When a numeric variable is displayed or printed, a leading space is reserved for a possible minus sign, and a following space is displayed or printed to comply with an ANSI (American National Standards Institute) standard.

A handy space scale can be made by displaying a line of apostrophies and copying it on the printer. Simply hold the [,] key down to get at least one full line of apostrophies, then copy your screen.

3. Answer the program planning questions (page 134) and draw your own flowchart if you feel these would help you.

Your Turn Write and run your program. To see my flowchart and listing, see page H-40. Remember, to be successful, your version need not be identical to mine.

PROBLEM: Write the "Sine" Program

Write a program that prints the angle and its sine for every fifth degree from 0 to 90 degrees.

My flowchart: Page H-41.
My listing and output: Page H-41.

PROBLEM: Write the "Common Log" Program

Write a program that prints all even numbers from 2 to 50 inclusive and the logarithm to the base 10 for each.

My flowchart and listing: Page H-41.
My output: Page H-42.

Do NOT Branch Into a FOR—NEXT Loop

Always begin execution of a FOR—NEXT loop at the beginning, at the FOR statement. Branching into the middle of a FOR—NEXT loop will cause an error. To help remember this rule, enter and run the "Forbidden" program. Figure 114 shows its flowchart, and Figure 115 shows its listing and output.

THIS PROGRAM IS FORBIDDEN

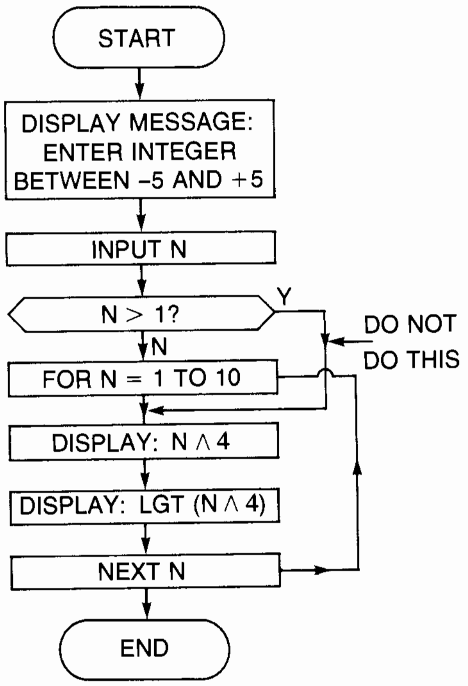
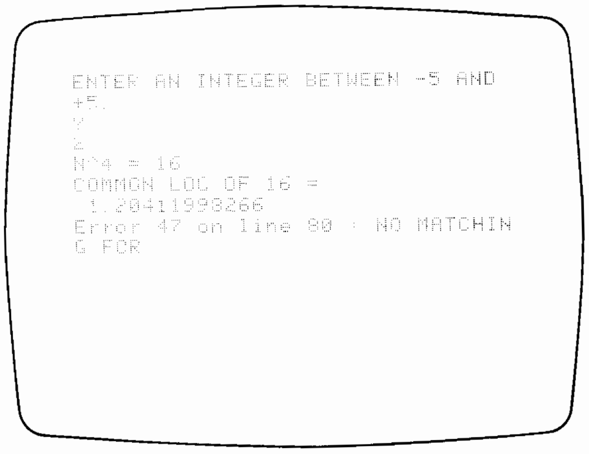


FIGURE 114
Flowchart for "Forbidden"

```
10 DISP "ENTER AN INTEGER BETWE
   EN -5 AND"
20 DISP "+5."
30 INPUT N
40 IF N>1 THEN 60
50 FOR N=1 TO 10
60 DISP "N^4 =";N^4
70 DISP "COMMON LOG OF";N^4;"="
   ;LGT(N^4)
80 NEXT N
90 END
```



```
ENTER AN INTEGER BETWEEN -5 AND
+5.
?
2
N^4 = 16
COMMON LOG OF 16 =
 1.20411998266
Error 47 on line 80 : NO MATCHIN
G FOR
```

FIGURE 115

Listing and output for "Forbidden"

Branching Out of a FOR—NEXT Loop

If a FOR ... TO statement is executed first, a branch out of a FOR — NEXT loop is allowed. The loop counter retains its value and may be used in the rest of the program like any other variable. In this situation, it is also permissible to branch back into the loop. This kind of branch into a FOR — NEXT loop is OK, since the FOR statement has already been executed.

To avoid complications when writing your programs, it's generally good practice to have your program complete a FOR — NEXT loop before it continues. Remember that a FOR — NEXT loop may include literally hundreds of statements, so your program may begin and complete many IF ... THEN and GOTO branches without ever leaving the FOR — NEXT loop.

Summary of Chapter 14

- **SQR(X): A function**

  Gives the square root of X.

  Reference: Page 14-224.

- **LGT(X): A function**

  Gives the common logarithm of X (the logarithm to the base 10).

  Reference: Pages 14-224–14-225.

- SIN(X): **A function**
  Gives the sine of X.

  Reference: Pages 14-225–14-227.

- DEG: **A BASIC word**
  General form:

  [line number] DEG

  Puts the HP-85 into degrees mode.

  Reference: Page 14-226.

- GRAD: **A BASIC word**
  General form:

  [line number] GRAD

  Puts the HP-85 into grads mode.
  Reference: Page 14-226.

- RAD: **A BASIC word**
  General form:

  [line number] RAD

  Puts the HP-85 into radians mode. **The HP-85 wakes up in radians mode.**

  Reference: Pages 14-225–14-226.

- **Do not branch into the middle of a FOR—NEXT loop.** If you do, the HP-85 will get confused and give you an error message. Reference: Pages 14-228–14-229.

- **You may branch out of a FOR — NEXT loop,** and then stay out or branch back in if you wish. However, it's generally safer programming (less chance for errors) if you avoid such branching. Reference: Page 14-229.

Review Test for Chapter 14

The answers are on page 14-231.

1. When the HP-85 wakes up, in what units does it measure angles?

2. Do the program segments below violate any BASIC programming rules? If so, which one(s)? Hints: SIN(), SQR() and LGT() are all used correctly. Also, each individual statement is acceptable taken by itself.

a.

```
 :
140 A=SIN(D)
150 IF A<.23 THEN 180
160 ! THE ALPHA FUNCTION FOLLOWS
170 FOR A=9 TO 1 STEP -1
180 W=W+SQR(A)
190 NEXT A
200 IF W>.5 THEN 30
 :
```

b.

```
 :
120 J=SIN(11)
130 A=17
140 IF G=500 THEN 200
150 ! BYPASS FRAGINATOR
160 FOR A=10 TO 5 STEP -1
170 J=J+SIN(A)
180 IF SIN(A)+G>55 THEN 30
190 NEXT A
200 X=A*LGT(J+10)
 :
```

Answers to Review Test Questions for Chapter 14

1. In radians

   Reference: Pages 14-225–14-226.

2. a. In statement 150, when A is less than .23, a branch into the middle of the `FOR` — `NEXT` loop is attempted. This is against the rules.

   Reference: Pages 14-228–14-229.

   b. This program segment is OK. Statement 180 does branch out of the `FOR` — `NEXT` loop when `SIN(A)+G` is greater than 55, but this is acceptable.

   Reference: Page 14-229.

Chapter 15

Tell Your Program to Nest Without Laying Eggs

Preview

In Chapter 15, you will:

- Learn how a FOR—NEXT loop can contain
  a FOR—NEXT loop which can contain
  a FOR—NEXT loop which can contain ...
- Learn that FOR—NEXT loops are not mixers
- Learn how one statement can do a lot of assigning
- Get some idea of how powerful you can make a program using the BASIC smarts you already have.

Nested FOR—NEXT Loops

When one FOR—NEXT loop is entirely contained within another, the loops are said to be nested.

EXAMPLE: "Yawn" Program

Enter and run this nested loop program whose listing is shown in Figure 116. Figure 117 gives the flowchart.

```
 10 ! YAWN
 20 FOR O=1 TO 3
 30 PRINT "O =";O
 40 PRINT
 50 FOR I=1 TO 5
 60 PRINT "I =";I
 70 PRINT "YAWN"
 80 NEXT I
 90 PRINT
100 NEXT O
110 END
```

FIGURE 116
Listing for "Yawn"

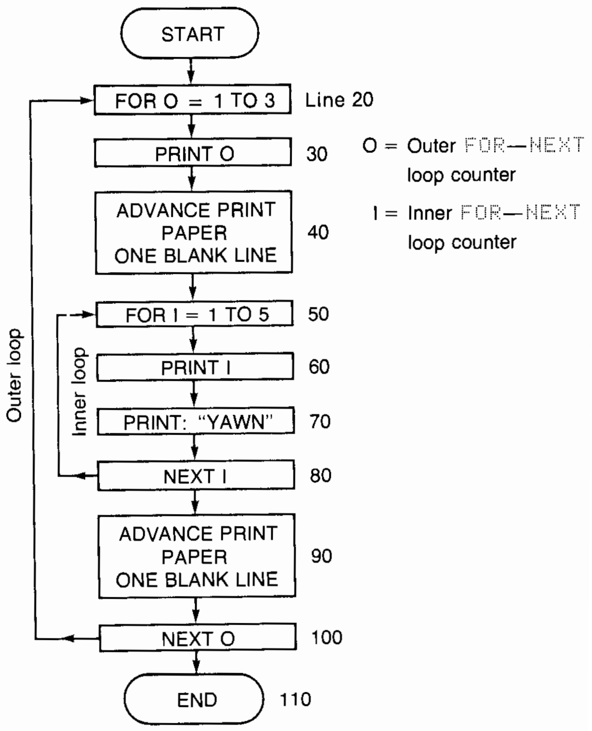

START
FOR O = 1 TO 3
Line 20
PRINT O
30
O = Outer FOR—NEXT loop counter
ADVANCE PRINT PAPER ONE BLANK LINE
40
I = Inner FOR—NEXT loop counter
FOR I = 1 TO 5
50
PRINT I
60
PRINT: "YAWN"
70
NEXT I
80
ADVANCE PRINT PAPER ONE BLANK LINE
90
NEXT O
100
END
110
Outer loop
Inner loop

FIGURE 117
Flowchart for "Yawn"

Check your output with mine in Figure 118 to make sure the HP-85 didn't make a typing error when you entered "Yawn."

Outer Loop Counter O	Inner Loop Counter I	Printed Output
1	—	O = 1
1	1	I = 1
1	1	YAWN
1	2	I = 2
1	2	YAWN
1	3	I = 3
1	3	YAWN
1	4	I = 4
1	4	YAWN
1	5	I = 5
1	5	YAWN
2	6	O = 2
2	1	I = 1
2	1	YAWN
2	2	I = 2
2	2	YAWN
2	3	I = 3
2	3	YAWN
2	4	I = 4
2	4	YAWN
2	5	I = 5
2	5	YAWN
3	6	O = 3
3	1	I = 1
3	1	YAWN
3	2	I = 2
3	2	YAWN
3	3	I = 3
3	3	YAWN
3	4	I = 4
3	4	YAWN
3	5	I = 5
3	5	YAWN
4	6	—
End	**of**	**program**

FIGURE 118
Output for "Yawn" and table of loop counters vs. output for "Yawn"

A study of your output for "Yawn" together with Figure 118 will help you understand how two nested FOR—NEXT loops work together.

The key to the operation of nested FOR—NEXT loops is in the location of their NEXT statements. As you've learned, when a NEXT statement gives the next STEP value to the loop counter. The HP-85 asks and answers this question: "Do I proceed to the next statement, or do I go back to the statement immediately following the related FOR?" When you ran "Yawn," the first NEXT that was executed was 80 NEXT I. The HP-85 answered its own question by going back to statement 60, the statement immediately following the related FOR. This process continued until statement 80 caused I to increase to 6. This time the HP-85's question was answered: "Proceed to the next statement." So a blank line was created on the printer, and 100 NEXT O was executed. Again, the HP-85 went to the statement immediately following the related FOR, but this time the related FOR statement was 20 FOR O=1 TO 3. Therefore, the HP-85 went to statement 30, and printed O = 2. Next came a blank line, then I was given the value 1 once again, and the cycle repeated.

How Many Loops Can Be Nested?

A large number—in fact, up to 255 nested loops may exist in a program, although available HP-85 memory would no doubt be used up long before the 255th innermost loop was created.

In practical terms, you may build as many nests as you wish.

Don't Mix Up the Nests

Figure 120 is a flowchart of another programming operation that is **not allowed.** Enter and run this program listed in Figure 119. You should get the output shown in Figure 121.

NOT ALLOWED! {40–60}

```
 10 ! MIX-UP
 20 FOR M=1 TO 4
 30 DISP "M =";M
 40 FOR X=6 TO 7
 50 DISP "X =";X
 60 NEXT M
 70 DISP "M LOOP IS FINISHED. M
    =";M
 80 NEXT X
 90 DISP "X LOOP IS FINISHED. X
    =";X
100 END
```

FIGURE 119

Listing for "Mix-up"

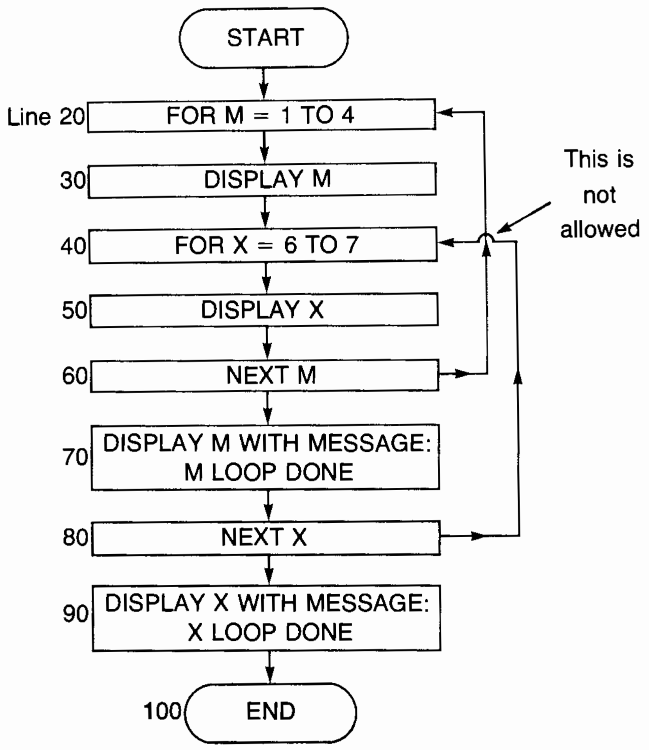
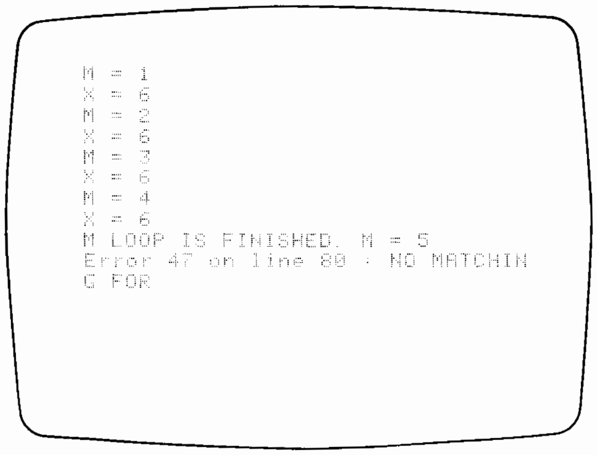


FIGURE 120

Flowchart for "Mix-up"

During the execution of the `M` loop, the `X` loop counter never changes its initial value of 6 since `NEXT X` is never reached. Also, and most important, each time `NEXT M` is executed, the `X` loop is cancelled to comply with the standard rules of BASIC. Consider the situation when the `M` loop is completed. The HP-85 tries to execute `NEXT X`, but discovers there is no `FOR X` statement alive and working in its memory. So it expresses its confusion with an error statement, as Figure 121 shows.



```
M = 1
X = 6
M = 2
X = 6
M = 3
X = 6
M = 4
X = 6
M LOOP IS FINISHED. M = 5
Error 47 on line 80 : NO MATCHIN
G FOR
```

FIGURE 121

Output for "Mix-up"

PROBLEM: Write the "Mix-up Unmixed" Program

The "Mix-up" program, Figure 119, needs attention. Fix "Mix-up" so it runs without error. Each value of M and X should be displayed. The message `X LOOP IS FINISHED.` ... should be displayed for each value of M. Finally, the message `M LOOP IS FINISHED.` ... should be displayed once at the end of the program.

My cure (flowchart, listing and output) is shown on pages H-42 and H-43.

EXAMPLE: "Son of Roots" Program

Enter and run this nested loop program, Figure 122. Check your output on page H-43.

```
10 ! SON OF ROOTS
20 FOR D=2 TO 4
30 PRINT "NUMBER    ROOT";D
40 FOR N=1 TO 9
50 PRINT N;"        ";N^(1/D)
60 NEXT N
70 PRINT
80 NEXT D
90 END
```

FIGURE 122

Listing for "Son of Roots"

PROBLEM: Draw a Flowchart

Draw a flowchart for "Son of Roots". Mine is on page H-43.

PROBLEM: Write the "Multiplication Test" Program

The computer society is charged by some with creating a generation where brains can calculate only as far as fingers and toes let them. Here's a program to stir those unexercised brain cells, and to teach them the multiplication table.

Problem Description "Multiplication Test" displays each of 81 multiplication problems in turn, from 1 × 1, 1 × 2, 1 × 3, ... up to 9 × 7, 9 × 8 and 9 × 9. It checks the user's answers, and tells him if he's right or wrong. If he's wrong, he's shown the right answer. In either case, the next problem is displayed. The number of right and wrong answers is remembered and displayed at the end of the test. If the user gets all problems right, a congratulatory message is also displayed.

The program starts by instructing the user how to take the test on the HP-85.

Hints

1. You'll use two variables to keep track of the number of right and wrong answers. Say you choose C and W. You'll want to initialize them to zero before you start generating your problems. Here's a handy way to assign the same value to a number of variables using a **multiple assignment statement:**

```
140 C,W=0
```

The only limits on the number of variables that can be assigned one value in one statement are the availability of variable names and the 95 character maximum length of one HP-85 line. For instance, this is a valid BASIC statement:

```
30 A,B2,X7,M,C,I,N3,O=1
```

2. A FOR—NEXT loop may contain many statements. (Technically as many as 9998, with the loop body using 9996 statements). My inner loop (loop body plus the FOR and NEXT statements) uses 16 statements. It includes an IF ... THEN statement which branches to a line within the loop. Remember my advice on page 14-229 to keep any branches which start within a FOR—NEXT loop confined to that loop. BASIC allows you to branch out of a loop, but experience has taught programmers to avoid branching out of a loop except when necessary.

Your Turn If you have trouble with this problem or with any of the others, get some ideas from my flowchart and then try to write your program before taking a look at my listing. Your most effective learning will occur when you're busy creating your own programs.

Here's where you can find my efforts:

Flowchart: Page H-44.

Listing: Page H-44.

Complicated FOR—NEXT Loop Arrangements

Many nested and unnested FOR—NEXT loops can exist in a single program in a variety of ways.Figure 123 shows one of these possible arrangements. As long as loops do not mix, the arrangements may be as complex as HP-85 memory and your ingenuity permit.

The blank task boxes might each represent tens or hundreds of statements of all kinds, including many conditional branches.

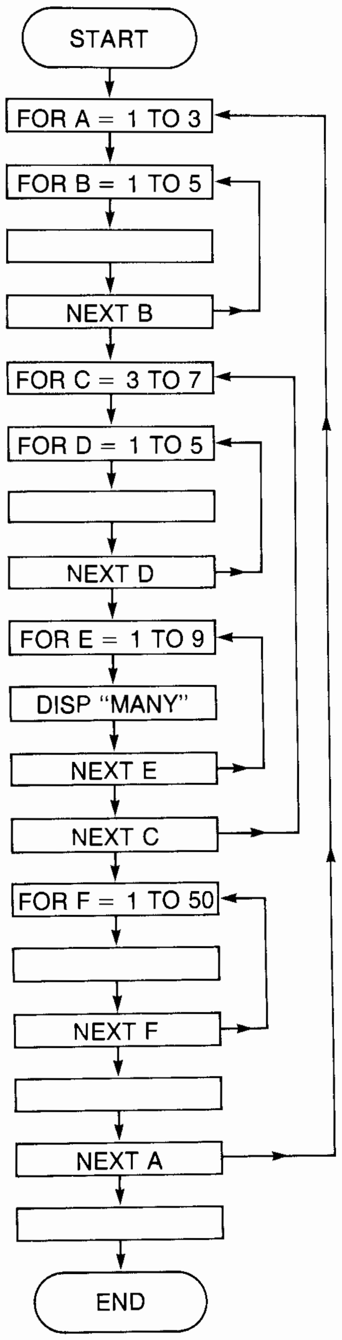


FIGURE 123

Complicated FOR—NEXT loop arrangement

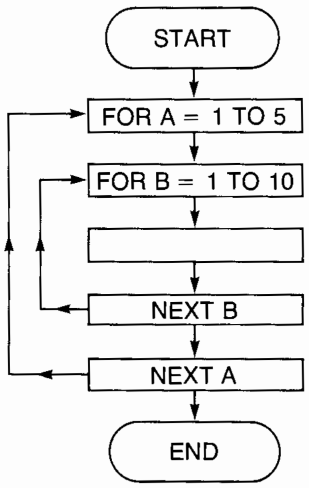
PROBLEM: Flowchart Analysis

If the program flowcharted in Figure 123 were run, how many times would the word ''many'' be displayed? For the answer, see page 15-240.

Summary of Chapter 15

- **Two FOR—NEXT loops are nested when one loop is entirely contained within another.**

  Example: Loop B is nested within loop A

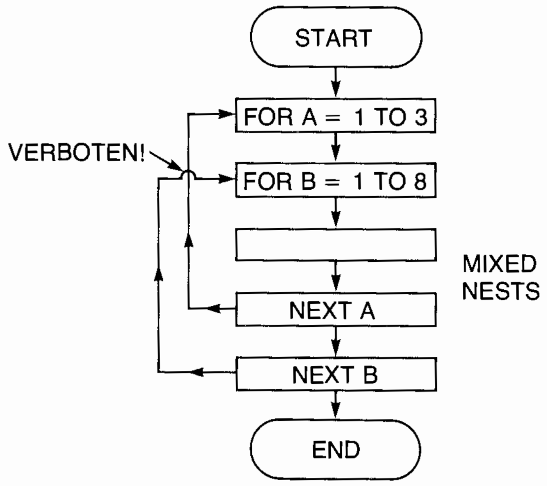


  Reference: Pages 15-232–15-234.

- **Up to 255 loops may be nested, one within the other.** Reference: Page 15-234.

- **FOR—NEXT loops may not be mixed.**

  Example:



Reference: Pages 15-234–15-235.

- **Multiple assignment statements**

  One assignment statement can assign one value to a number of variables. The variables are separated by commas, and they all appear to the left of the = symbol.

  Example:

```
    58 A,C,X7,P3,M=0
```

  Reference: Page 15-237.

Review Test for Chapter 15

This review test is on your BASIC Training cartridge. Execute `LOAD"TEST15"` and run it. The program will give you instructions and answers.

Answer to Flowchart Question

This question is asked on page 15-239.

The word ''many'' would be displayed 135 times.

The A loop would execute 3 times. For each A loop, the C loop would run 5 times (3, 4, 5, 6, 7). For each C loop, the E loop would run 9 times. $9 \times 5 \times 3 = 135$.

Notes

Chapter 16

Teach Your Program to Read

Preview

In Chapter 16, you will:

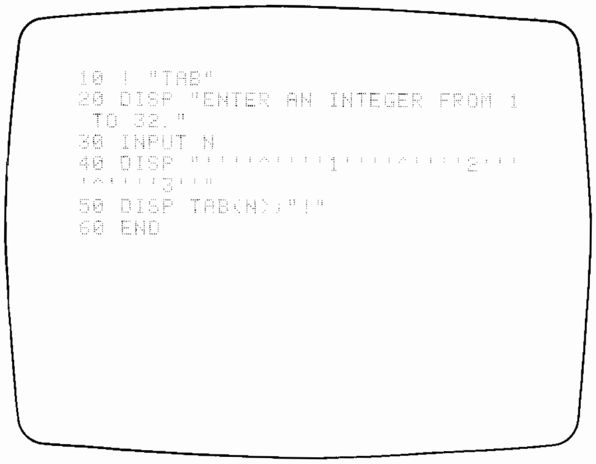
- Learn how to increase your control over where things are printed and displayed.
- Learn a third way to put numbers into your program.
- Write 3 programs and modify another.

TAB(X): A Function

TAB(X) is used only in PRINT and DISP statements. It acts like a tab function on a typewriter. To see it in action, run the following example.

EXAMPLE: "Tab" Program

Enter "Tab" whose listing is shown in Figure 124. Notice that statement 40 displays a scale showing each position of Capricorn's 32 position line. Also notice the semicolon in statement 50.



```
10 ! "TAB"
20 DISP "ENTER AN INTEGER FROM 1
 TO 32."
30 INPUT N
40 DISP "''''^''''1''''^''''2'''
'^''''3''"
50 DISP TAB(N);"!"
60 END
```

FIGURE 124
Listing for "Tab" program

Run "Tab" several times, entering a number from 1 to 32 each time.

As you see, the ! is displayed at the line position specified by TAB(N) in statement 50.

Now work through the following problem to see an important advantage of TAB(X).

PROBLEM: Write the "Grandson of Roots" Program

Rewrite my "Son of Roots" program (see Figure 122, page 15-236). Do *not* use `TAB(X)`, at least not yet. Have your "Grandson of Roots" program print the second, third and fourth roots of the integers from one to ten, rather than from one to nine. The result will show a cosmetic flaw that you'll soon fix with `TAB(X)`.

Check your listing and output against mine in Figure 125.

Notice in Figure 125 how all the roots line up nicely for all numbers from 1 through 9, but the three roots of 10 are moved right one place. This results from statement 50:

```
50 PRINT N;"    ";N^(1/D)
```

This statement "prints" four spaces between the number and its root.

The second digit of 10 in the number column moves the four spaces and the root one additional position to the right.

Now to the rescue comes `TAB(X)`. Change statement 50 to:

```
50 PRINT N;TAB(8);N^(1/D)
```

```
10 ! GRANDSON OF ROOTS
20 FOR D=2 TO 4
30 PRINT "NUMBER    ROOT";D
40 FOR N=1 TO 10
50 PRINT N;"    ";N^(1/D)
60 NEXT N
70 PRINT
80 NEXT D
90 END
```

```
NUMBER     ROOT 2
 1         1
 2         1.41421356237
 3         1.73205080757
 4         2
 5         2.2360679775
 6         2.44948974278
 7         2.64575131106
 8         2.82842712475
 9         3
 10         3.16227766017
```

```
NUMBER     ROOT 3
 1         1
 2         1.25992104989
 3         1.44224957031
 4         1.58740105197
 5         1.70997594668
 6         1.81712059283
 7         1.91293118277
 8         2
 9         2.08008382305
 10         2.15443469003
```

```
NUMBER     ROOT 4
 1         1
 2         1.189207115
 3         1.31607401295
 4         1.41421356237
 5         1.49534878122
 6         1.56508458007
 7         1.6265765617
 8         1.68179283051
 9         1.73205080757
 10         1.77827941004
```

FIGURE 125

Listing and output for "Grandson of Roots"

Now run your program again. Your listing and output should look like Figure 126 (except for the remark).

```
10 ! TAB, GRANDNEPHEW OF ROOTS
20 FOR D=2 TO 4
30 PRINT "NUMBER   ROOT";D
40 FOR N=1 TO 10
50 PRINT N;TAB(8);N^(1/D)
60 NEXT N
70 PRINT
80 NEXT D
90 END
```

```
NUMBER   ROOT 2
 1       1
 2       1.41421356237
 3       1.73205080757
 4       2
 5       2.2360679775
 6       2.44948974278
 7       2.64575131106
 8       2.82842712475
 9       3
 10      3.16227766017
```

```
NUMBER   ROOT 3
 1       1
 2       1.25992104989
 3       1.44224957031
 4       1.58740105197
 5       1.70997594668
 6       1.81712059283
 7       1.91293118277
 8       2
 9       2.08008382305
 10      2.15443469003
```

```
NUMBER   ROOT 4
 1       1
 2       1.189207115
 3       1.31607401295
 4       1.41421356237
 5       1.49534878122
 6       1.56508458007
 7       1.6265765617
 8       1.68179283051
 9       1.73205080757
 10      1.77827941004
```

FIGURE 126

Listing and output for "Tab, Grandnephew of Roots"

There are 32 possible tab positions on the HP-85's display and printer, starting with `TAB(1)` and ending with `TAB(32)`. In "Tab, Grandnephew of Roots," `TAB(8)` instructs each root to begin in the eighth position from the left, including the roots for the number ten. Then why do the numerals of each root actually begin in the ninth position? Read on.

Spaces Displayed and Printed with Numbers

Not only do the numerals printed in the root column start in the ninth, not the eighth position, but the numerals in the number column start in the second position, even though statement 50 instructs N to be printed in position one. These numbers in fact do start at positions eight and one. The space that every positive number or zero begins with, whether displayed or printed by the HP-85, is reserved for the minus sign used for negative numbers. So when you use `TAB(X)` to position numbers, remember that a positive number will start with a space at position X, and a negative number will start with a minus sign at position X.

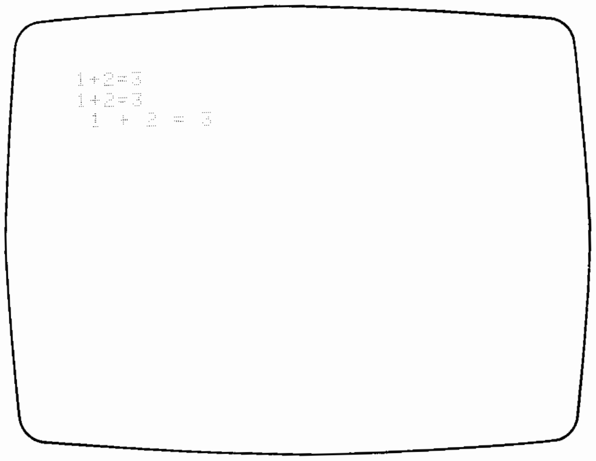
Also, when a number is displayed or printed, a space always follows the number.

These leading and following spaces are displayed and printed with any number, whether represented by a constant, a variable, or a mathematical expression, like 3*H/V-D.

EXAMPLE: "Spaced Out Numbers" Program

Enter and run this program, Figure 127 See the extra spaces that are displayed only with numbers. These spaces are not displayed when the numerals are used as quoted characters.

```
10 DISP "1+2=3"
20 DISP "1";"+";"2";"=";"3"
30 DISP 1;"+";2;"=";3
40 END
```



```
1+2=3
1+2=3
 1  +  2  =  3
```

FIGURE 127

Listing and output for "Spaced Out Numbers"

Some TAB(X) Truths

1. When you use TAB(X), you must instruct the HP-85's
   a. What to print or display, and
   b. Where to print or display it.

2. The argument in TAB(X) may be a constant, a variable or an expression.

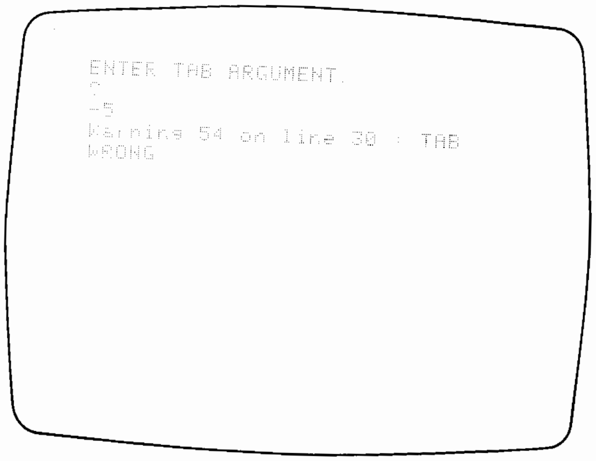
   Examples:

```
TAB(5)
TAB(G7)
TAB(ABS(4*H2-3^C)+1)
```

3. The argument X in TAB(X) must be one or greater (X > = 1). If X is less than one (X < 1), a warning message is given, and TAB(X) is changed to TAB(1).

   Example program (listing and output) showing a TAB(X) argument of –5:

```
10 DISP "ENTER TAB ARGUMENT."
20 INPUT N
30 DISP TAB(N);"WRONG"
40 END
```



```
ENTER TAB ARGUMENT.
?
-5
Warning 54 on line 30 : TAB
WRONG
```

4. For an argument X between 1 and 32.5 ($1 < X < 32.5$), X is rounded to the nearest integer before TAB(X) is executed.

   Examples:

   ```
   TAB(5.4) = TAB(5)
   TAB(5.5) = TAB(6)
   ```

5. For X greater than or equal to 32.5 ($X >= 32.5$), a whole multiple of 32 is subtracted from X to leave X positive but less than 32.5 ($.5 <= X < 32.5$). Then this smaller X is rounded to the nearest integer before TAB (X) is executed.

   Examples:

   ```
       TAB(50) = TAB(50-1*32) = TAB(50-32) = TAB(18)

   TAB(271.7) = TAB(271.7-8*32) = TAB(271.7-256) = TAB(15.7) =
   TAB(16)
   ```

6. Always use semicolons with TAB(X). If you use a comma instead, you'll get a wide space rather than the close spacing you probably want.

EXAMPLE: "Hard Z" Program

Study the program shown in Figures 128 and 129. Enter and run it if you wish. Here's a good opportunity to use a position scale. Remember one way to make one? Display a complete line of apostrophies and copy it on the printer. Fold the paper to put the apostrophies at the edge.

Note the FOR—NEXT loop, lines 210-230, that advances the printing beyond the tear-off bar.

PROBLEM: Write the "Easy Z" Program

Rewrite "Hard Z" using TAB(X). Have your program give an output identical to that given by "Hard Z," Figure 129. My flowchart and listing are on page H-45.

```
10 ! HARD Z
20 PRINTER IS 2
30 PRINT "          THIS IS A Z"

40 PRINT
50 PRINT "        **************
**"
60 PRINT "
*"
70 PRINT "                     *
"
80 PRINT "                    *"

90 PRINT "                   *"
100 PRINT "                  *"
110 PRINT "                 *"
120 PRINT "                *"
130 PRINT "               *"
140 PRINT "              *"
150 PRINT "             *"
160 PRINT "            *"
170 PRINT "           *"
180 PRINT "          *"
190 PRINT "         *"
200 PRINT "        *************
***"
210 FOR L=1 TO 7
220 PRINT
230 NEXT L
240 END
```

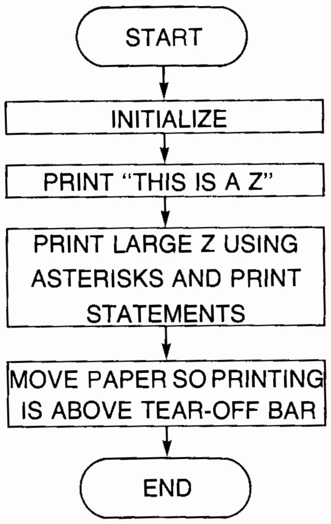

START
INITIALIZE
PRINT "THIS IS A Z"
PRINT LARGE Z USING ASTERISKS AND PRINT STATEMENTS
MOVE PAPER SO PRINTING IS ABOVE TEAR-OFF BAR
END

FIGURE 128

Flowchart and PRINTALL CRT listing for "Hard Z"

PROBLEM: Write the "Center" Program

Write a program using TAB(X) that prints this familiar heading and moves it comfortably above the tear off bar.

```
     HEWLETT-PACKARD
  GETTING DOWN TO BASIC
CONTINUATION OF CHAPTER 3
```

Page H-45 shows my listing.

```
  THIS IS A Z

****************
              *
             *
            *
           *
          *
         *
        *
       *
      *
     *
    *
   *
  *
 *
****************
```

FIGURE 129

Output for "Hard Z" and "Easy Z"

READ: A BASIC Word

DATA: A BASIC Word

Get an introduction to these new words through an example.

EXAMPLE: "Average" Program

Enter and run "Average." The flowchart is shown in Figure 130, while Figure 131 gives the listing and output. Notice how the DATA statement does not appear in the flowchart. The presence of "READ" in the flowchart means that at least one DATA statement must be in the program.

When statement 50 is executed the first time, the first number in the DATA statement, 5, is read into the program. The number 5 is then added to the sum S, which was initialized to zero in statement 30. The second time through the loop, the next number in the DATA statement, 14, is read, then added to 5 to make a new sum S of 19.

This sequence continues until, during the last execution through the loop, the last number, –471, is read. Statement 70 then sets N to 11, and since 11 is greater than 10, the program moves to statement 80. The average is computed and printed. The HP-85 then skips the DATA statement and the program ends.

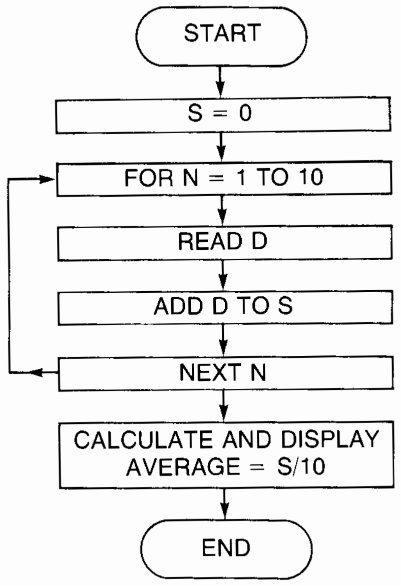


FIGURE 130

Flowchart for "Average"

```
 10 ! AVERAGE
 20 CRT IS 1
 30 S=0
 40 FOR N=1 TO 10
 50 READ D
 60 S=S+D
 70 NEXT N
 80 DISP "AVERAGE:";S/10
 90 DATA 5,14,362,43,201,-61,77,
    -843,2,-471
100 END
```

```
AVERAGE:-67.1
```

FIGURE 131

Listing and PRINTALL output for "Average"

Facts about READ and DATA

1. READ is pronounced "reed."

2. The number of DATA statements in a program is limited only by the 9999 limit on program statements (or the memory capacity of your HP-85).

3. DATA statements may be placed anywhere in a program.

4. There are no special restrictions on the number of READ statements in a program.

5. The numbers of READ and DATA statements need not be equal.

6. Data items in DATA statements are identified by a DATA statement "pointer" the HP-85 has inside its body. When a DATA statement is executed, it reads the data item identified by this pointer.

7. As READ statements read data items, one after the other, the DATA statement pointer moves from the first item in the first DATA statement to the last item in that statement, then to the first item in the second DATA statement, and so on. The pointer continues to move, item by item, until the last READ statement is executed, or until the last data item is read.

8. The DATA statement pointer moves to the next data item (if one exists) immediately after a DATA statement reads a data item, and before the next program statement is executed.
9. A program may have more data items than are used by READ statements.
10. But, a program may not have READ statements ask for more items than exist in DATA statements. A way around this is RESTORE. More on this later.

You now have three ways to assign a value to a variable. Here's how READ and DATA compare with assignment and INPUT statements:

1. READ and DATA statements are more useful than assignment statements for large amounts of data.
2. READ and DATA statements are more useful than INPUT statements if the program is to be run several times using the same data.

PROBLEM: Write the "Biggest and Smallest" Program

Program Description This programs prints and identifies the largest and smallest numbers in a group of numbers. The group of numbers is that in statement 90 of the "Average" program, Figure 131.

Hint The first time through the loop, set L (for largest) and S (for smallest) equal to D (the data item). For subsequent trips through the loop, avoid this statement.

Your Turn This is a small but tricky program. Good luck!

My program:	Flowchart: Page H-46.
	Listing and output: Page H-46.

RESTORE or RESTORE [line number]: A BASIC Word

This BASIC word allows a DATA statement list to be used again.

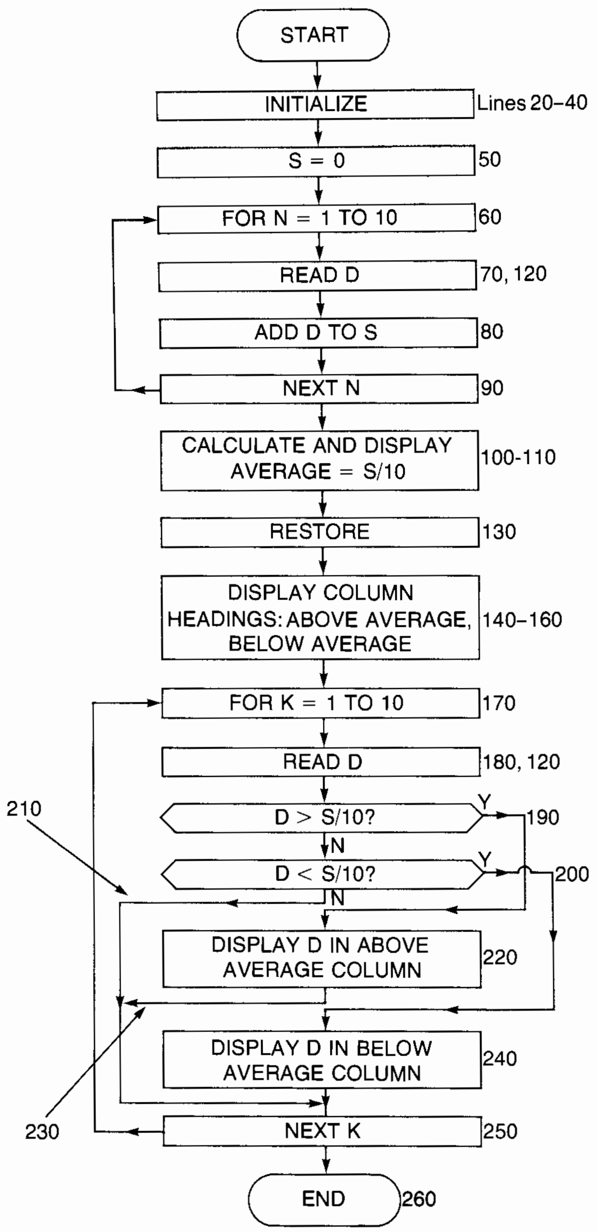
If *no* line number follows RESTORE, the execution of RESTORE positions the DATA statement pointer at the first data item in the *first* DATA statement in the program.

If a line number *does* follow RESTORE (like RESTORE 140), the execution of RESTORE [line number] positions the DATA statement pointer at the first item in the DATA statement at the referenced line number.

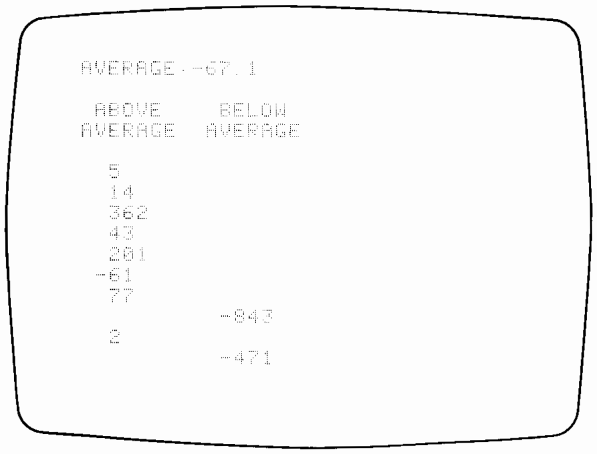
In either case, the DATA statement item located by the pointer will be read when the next READ statement is executed.

EXAMPLE: "Big?" Program

This is an enhancement of the "Average" program, Figure 131. After the average is obtained, the program prints those numbers larger than average in one column and those smaller than average in another. The flowchart appears in Figure 132, while the listing and output are in Figure 133.



```
10 ! BIG?
20 CRT IS 1
30 NORMAL
40 CLEAR
50 S=0
60 FOR N=1 TO 10
70 READ D
80 S=S+D
90 NEXT N
100 DISP "AVERAGE:";S/10
110 DISP
120 DATA 5,14,362,43,201,-61,77,
    -843,2,-471
130 RESTORE
140 DISP " ABOVE      BELOW"
150 DISP "AVERAGE   AVERAGE"
160 DISP
170 FOR K=1 TO 10
180 READ D
190 IF D>S/10 THEN 220
200 IF D<S/10 THEN 240
210 GOTO 250
220 DISP TAB(2);D
230 GOTO 250
240 DISP TAB(11);D
250 NEXT K
260 END
```



```
AVERAGE:-67.1

 ABOVE     BELOW
AVERAGE   AVERAGE

  5
  14
  362
  43
  201
 -61
  77
           -843
  2
           -471
```

FIGURE 133

Listing and output for "Big?"

S = Sum of numbers in DATA statement

N = First loop counter

D = Number in DATA statement

K = Second loop counter

FIGURE 132

Flowchart for "Big?"

Notice the RESTORE in statement 130. The execution of 130 RESTORE allows all the data items to be read a second time by the second FOR—NEXT loop, statements 170-250. Statements 190 and 200 sort the DATA statement items by size. The average value is never assigned to a variable. Instead, it is represented by S/10, thereby saving one step.

Take time to study the "Big?" program. A good understanding of how this program works will help you master the sorting routine you'll study in the next chapter. If it would help you feel comfortable with its operation, enter and run "Big?." Check to see that your output agrees with mine in Figure 133.

Summary of Chapter 16

- **Spaces displayed and printed with numbers**

  When any value (constant, variable or evaluated expression) is displayed or printed, a leading space appears with a non-negative number, and a minus sign appears with a negative number. Also, a trailing space always appears when any value is displayed or printed.

  Reference: Pages 16-244–16-245.

- TAB(X): **A function**

  - TAB(X) works like the tab control on a typewriter. It is used only with DISP or PRINT statements. To display the word "BASIC" starting on the 14th position of the HP-85's 32 character line, use this statement:

    [line number] DISP TAB(14);"BASIC"

  - The argument in TAB(X) may be a constant, a variable or an expression.
  - The argument X in TAB(X) must be one or greater (X > = 1). If X is less than one (X < 1), a warning message is given, and TAB(X) is changed to TAB(1).
  - For an argument X between 1 and 32.5 (1 < X < 32.5), X is rounded to the nearest integer before TAB(X) is executed.
  - For X greater than or equal to 32.5 (X > = 32.5), a whole multiple of 32 is subtracted from X to leave X positive but less than 32.5 (.5 < = X < 32.5). Then this smaller X is rounded to the nearest integer before TAB(X) is executed.

  Reference: Pages 242-16-248.

- READ: **A BASIC word**

  General form:

  [line number] READ [variable name]

- DATA: **A BASIC word**

General form:

[line number] DATA [data item] , [data item] , [data item] ...

Example:

```
:
130 READ M6
:
200 DATA 14,312,2197,3,0,14
:
```

- ◆ There are no special restrictions on the number of READ or DATA statements in a program.
- ◆ The numbers of READ and DATA statements in a program need not be equal.
- ◆ DATA statements may be placed anywhere in a program.
- ◆ Each DATA statement item is used in order each time any READ statement is executed. Data items are read in order of DATA statement number and in order of appearance within a DATA statement.
- ◆ READ statements may not ask for more items than exist in DATA statements.

Reference: Pages 16-248–16-250.

- RESTORE: **A BASIC word**

There are two general forms:

[line number] RESTORE (with no line number)

When executed, the DATA statement pointer is repositioned to the first data item in the first DATA statement.

[line number] RESTORE [line number]

When executed, the DATA statement pointer is repositioned to the first data item in the DATA statement referenced by the RESTORE statement.

Reference: Page 16-250.

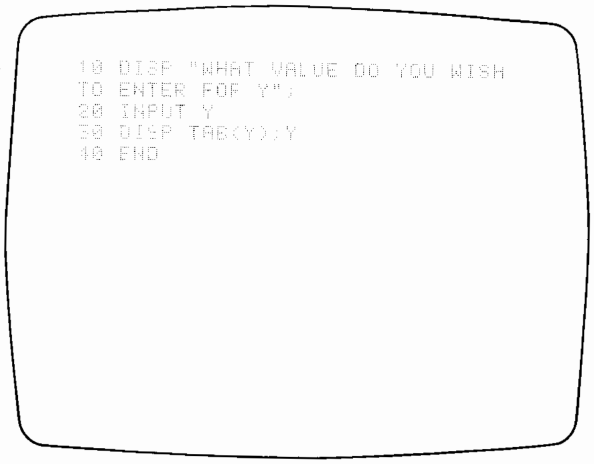
- READ **and** DATA **vs. assignment statements:** READ and DATA are more useful for large amounts of data. Reference: Page 16-250.

- READ **and** DATA **vs.** INPUT: READ and DATA are more useful if the program is to be run several times using the same data. Reference: Page 16-250.

Review Test for Chapter 16

For answers, see page 16-255.

1. Enter and run the program immediately following this paragraph. When you're asked to enter a value for Y, enter a number between 100 and 120 that will leave 9 blank spaces to the right of the last displayed character. To get this one right, you should enter the correct number the first try. Here's the program:



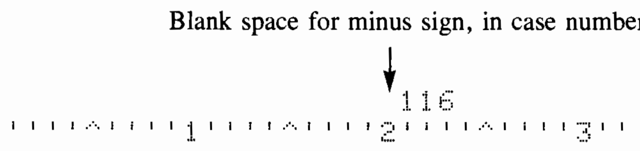
```
10 DISP "WHAT VALUE DO YOU WISH
TO ENTER FOF Y";
20 INPUT Y
30 DISP TAB(Y);Y
40 END
```

2. When statement 80 in the following program is executed, to which data item in what DATA statement will the internal DATA statement pointer point? For instance, if the pointer pointed to the first 7 in the first DATA statement, the answer would be item 3 in statement 150.

```
 10 T=0
 20 FOR A=1 TO 4
 30 IF T>=50 THEN 50
 40 GOTO 60
 50 RESTORE 160
 60 IF T>70 THEN 80
 70 GOTO 100
 80 DISP "T=";T
 90 STOP
100 FOR B=1 TO 5
110 READ X
120 T=T+X
130 NEXT B
140 NEXT A
150 DATA 4,6,7,4
160 DATA 6,4,3,7
170 DATA 3,6,3,4
180 DATA 3,7,7,4
190 END
```

Answers to Review Test Questions for Chapter 16

1. The number you should enter is 116.



To leave 9 blank spaces, the number display should start at character position 20. This means the argument of the `TAB()` function should be either 20 or a whole multiple of 32 plus 20, like

$32 \times 1 + 20 = 32 + 20 = 52$
$32 \times 2 + 20 = 64 + 20 = 84$
$32 \times 3 + 20 = 96 + 20 = 116$
$32 \times 4 + 20 = 128 + 20 = 148$
etc.

Of these possible arguments, only 116 is between 100 and 120.

Reference: Pages 16-244–16-246.

2. Item 1 in statement 160. If you were very clever, you noticed that the last 4 statements executed when this program is run must be:

```
50 RESTORE 160
60 IF T>70 THEN 80
80 DISP "T=";T
90 STOP
```

Regardless of what data item causes T to exceed 70, statement 50 will move the `DATA` statement pointer to the first data item in statement 160. (For the record, just before statement 50 was executed for the last time, the pointer pointed to item 2 in statement 170.)

Reference: Pages 16-249–16-250.

Chapter 17

Sort Those Numbers!

Preview

In Chapter 17, you will:

- Learn how your program can handle long lists of numbers with ease.
- Learn a slick way to sort a group of numbers by size.
- Acquire more bug killing power.
- Write two programs to analyze the rainfall in Drench, Oregon.

Arrays

You now have enough BASIC tools available to take advantage of one of the biggest weapons of all: arrays. What are arrays? Arrays are organized collections of numbers, like lists and tables. Why have arrays? Arrays allow programs to handle easily large collections of numbers.

The HP-85 offers you one dimensional arrays (lists) and two dimensional arrays (tables, with rows and columns). This course will cover one dimensional arrays. After you finish this course, you should find your Owner's Manual discussion of two dimensional arrays to be straightforward.

One Dimensional Arrays

There are three important characteristics of one dimensional arrays:

1. They are lists of numbers or values.
2. Each number is represented by a single valued **variable,** like B(4).
3. Each single valued variable has a single **subscript.** For instance, the variable B(4) has the subscript 4.

Here are a couple of examples of lists that can be used for one dimensional arrays. See Figures 134 and 135.

RACE

Runner's Finishing Position	1	2	3	4	5	6	7	8	9	10
Runner's Identification Number	4	7	6	1	8	3	5	10	9	2

FIGURE 134

List of race results

DAILY RAINFALL IN DRENCH, OREGON FOR JANUARY 1978

Date	Rainfall in Inches
1	3
2	7
3	1
4	19
5	37
6	6
7	1
8	5
9	2
10	4
11	7
12	9
13	8
14	3
15	0
16	2
17	6
18	4
19	3
20	6
21	10
22	12
23	13
24	11
25	14
26	16
27	14
28	13
29	8
30	7
31	4

FIGURE 135
List of January rainfall

Array Mysteries Revealed

Here's another look at the Race table:

Runner's Finishing Position	Runner's Identification Number	Subscripted Variable	Value
1	4	R(1)	4
2	7	R(2)	7
3	6	R(3)	6
4	1	R(4)	1
5	8	R(5)	8
6	3	R(6)	3
7	5	R(7)	5
8	10	R(8)	10
9	9	R(9)	9
10	2	R(10)	2

FIGURE 136
List of race results showing subscripted variables

Using Figure 136, I'll explain some things about one dimensional arrays.

1. The **array variable name** in Figure 136 is R. Just like simple, one valued variable names, there are 286 array variable names available to you for each program, such as C7, H3, M, and I2. The same name may be used for an array as is used for a simple variable in the same program, but you'll learn to tell them apart easily. Remember, an array variable represents a *collection of single valued variables,* or numbers, *not* one number.

2. R(3) is a subscripted variable. In an array, a subscripted variable *does* represent one number just like a simple variable. When you use and see a subscripted variable, remember that it only represents a single number, in spite of its imposing appearance. For instance:

   W(17) = 2
   Z(162) = 1
   B(7) = 362.13

3. In a subscripted variable, R(3) for instance, 3 is the subscript. It tells programmer, user and the HP-85 that the number represented by R(3) is located in the third position or element of array R.

4. R(3) is pronounced "R sub-3."

5. There is a difference between R3 and R(3). It's something like the difference between "Richard Third" and "Richard the Third." "Richard Third" is simply the name of a person, but "Richard the Third" is the special name of the third Richard in a line of Richards, starting with Richard the First.

6. However, there is an important similarity between R3 and R(3). Each represents a single number.

7. A subscript may be a simple variable, a subscripted variable, or even an expression, which is a combination of constants and variables joined by arithmetic operators, like +, −, ⋆, /, Λ. Here are two subscripted variables using other variables as the subscripts.

   H(V)
   W(H(V))

   Perhaps a further look at that last one would be worthwhile. Look at it this way: Say we have two arrays, H and W, shown in Figure 137.

ARRAY H

Subscripted Variable	Value
H(1)	2
H(2)	3
H(3)	4

ARRAY W

Subscripted Variable	Value
W(1)	21
W(2)	22
W(3)	23
W(4)	24
W(5)	25

FIGURE 137
Arrays H and W

Figure 138 shows these arrays used in a mini-program. The program initializes only variables H(3) and W(4):

```
10 H(3)=4                                   A = 4
20 W(4)=24                                  B = 24
30 V=3
40 A=H(V)
50 PRINT "A =";A
60 B=W(H(V))
70 PRINT "B =";B
80 END
```

FIGURE 138

Listing and output for program "W(H(V))"

Why don't you enter this mini-program and confirm that I've shown you the correct output?

Here's how this small but complex program works. Since V=3 from statement 30, H(V) = H(3). H(3) represents the number in the third position or element of array H.
From statement 10, H(3) = 4. So:

$$H(V) = H(3) = 4$$

Using these relationships in statement 60:

$$W(H(V)) = W(H(3)) = W(4).$$

The variable W(4) represents the fourth value of array W. In statement 20, this variable was initialized to 24. This gives:

$$W(H(V)) = W(H(3)) = W(4) = 24$$

Make an effort to understand what just happened. Arrays may seem a little cloudy at first, but as you work with them, they get easier.

Here's another example of an array variable:

$$A(4 + 3)$$

Now do you see why BASIC does not allow the algebraic way of expressing multiplication? In algebra, A(4 + 3) means "the variable A times the sum of 4 + 3" or "A times 7." In BASIC, A(4 + 3) is the seventh value of array A.

Another example:

$$P(A + B + 2 \star C)$$

You'll seldom see or use array variables that look this complicated, but such a variable is possible, and such a variable represents nothing more than a single number, a value in array P.

OPTION BASE 0: A BASIC Word

OPTION BASE 1: A BASIC Word

The standard way to count positions or elements in an array is to start with zero, not one. This is another ANSI standard with which Hewlett-Packard complies. For instance, an array B with five positions would be arranged like this:

ARRAY B

Position	Subscript	Variable
1	0	B(0)
2	1	B(1)
3	2	B(2)
4	3	B(3)
5	4	B(4)

However, the HP-85's BASIC allows you to have arrays start with subscript one, not zero. When you put an `OPTION BASE 1` statement *before any statement referencing an array variable,* the first position in that array will be associated with a subscript of 1, not 0. For instance, say your program using array B had, as its first statement:

```
10 OPTION BASE 1
```

Your array B would look like this:

ARRAY B

Position	Subscript	Variable
1	1	B(1)
2	2	B(2)
3	3	B(3)
4	4	B(4)
5	5	B(5)

If `OPTION BASE 1` is not executed in a program, `OPTION BASE 0` is automatically in force. The HP-85 wakes up in `OPTION BASE 0`, which means the first subscript of an array is numbered zero. The optional `OPTION BASE 0` statement may be used in a program to remind users that the first element or position of all arrays in the program is numbered zero.

Perhaps you feel as I do, that arrays are easier to work with when the third position, for instance, is numbered three, not two. All my array programs in this course use, as a low numbered statement:

```
OPTION BASE 1
```

Note that `OPTION BASE` may appear only once in a program. Also note that `OPTION BASE` cannot be executed from the keyboard.

DIM: A BASIC Word

If a one dimensional array has more than 10 (`OPTION BASE 1`) or 11 (`OPTION BASE 0`) positions or elements, it must be **dimensioned** in the program. That is, a statement in the program must define how many elements the array has. One `DIM` statement may dimension one array or more than one array. Also, one program may have more than one `DIM` statement. Examples:

```
43 DIM K(15),L(35),M(200)
25 DIM B(30)
15 DIM C2(5)
```

These statements mean that one dimensional array K has 15 elements, L has 35, M has 200, B has 30, and array C2 has 5 elements. Notice that the word `DIM` appears only once in the dimension statement.

Also notice that array C2 was dimensioned to have five elements. Such a dimension statement is not required, since in the absence of a dimensioning statement, the HP-85 would automatically dimension this array to have 10 or 11 elements.

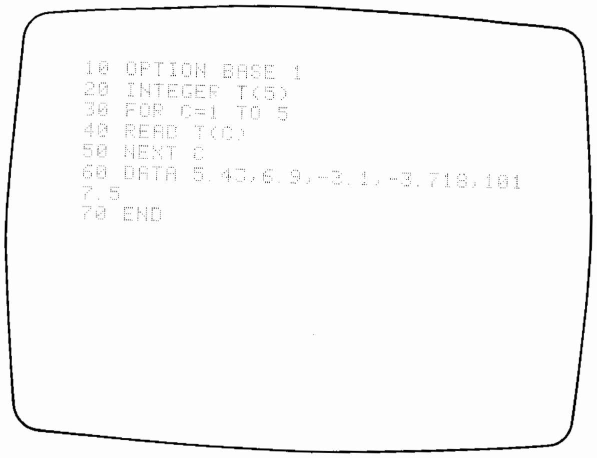
However, using a `DIM` statement for your under 10 or 11 element arrays gives two benefits:

1. It saves HP-85 memory for other purposes (more data, more statements).

2. It makes it easy for one using your listing to learn the name and size of each array in your program. This is especially true if you put all your dimensioning statements near the beginning of your listing.

INTEGER: A BASIC Word

Another way to dimension an array is to use `INTEGER`, which not only dimensions like `DIM` but rounds each value in the array to the nearest integer. This is called integer precision. For instance, take a look at this program:



```
10 OPTION BASE 1
20 INTEGER T(5)
30 FOR C=1 TO 5
40 READ T(C)
50 NEXT C
60 DATA 5.43,6.9,-3.1,-3.718,101
7.5
70 END
```

Note that the last data item is 1017.5, not 7.5, since no comma follows 101.

The first time through the FOR—NEXT loop, statement 40 says READ T(1), which assigns the first DATA statement item to variable T(1). Since statement 10 dimensions array T to be INTEGER T(5), the first data item, 5.43, is rounded to the nearest integer, 5, before it is assigned to T(1).

In a similar way, the remaining DATA statement items are rounded to the nearest integer before being assigned to the corresponding subscripted variable, as follows:

Number in DATA statement	**Number in array T**
5.43	5
6.9	7
-3.1	-3
-3.718	-4
1017.5	1018

A program may have more than one DIM statement and more than one INTEGER statement. Also, a program may have both DIM and INTEGER statements.

Some more examples of INTEGER dimension statements are shown below:

```
15 INTEGER B(45)
5 INTEGER K4(17),K5(21),K6(32)
22 INTEGER M(312)
```

All five arrays are dimensioned to be INTEGER arrays, including arrays K5 and K6. Notice, in statement 5, that the word INTEGER is not repeated for arrays K5 and K6. They are automatically dimensioned to be integer arrays. By the way, INTEGER works just fine for simple variables. Also, a single INTEGER statement can dimension both simple and array variables. The following statements declare the variables A, B, C, D and E plus the array L to have integer precision.

```
25 INTEGER A
17 INTEGER B,C,D
21 INTEGER E,L(20)
```

You might wish to enter and run the following six statement program to get a feel for integer precision. Enter decimal numbers, like 25.731 and see the integer results.

```
 5 INTEGER N
10 DISP "ENTER NUMBER."
20 INPUT N
30 DISP N
40 GOTO 10
50 END
```

EXAMPLE: "Race" Program

I'll describe this program and then ask you to become familiar with its flowchart and listing before you move on.

Program Description This program assists the timekeeper at the finish line of a ten man marathon. As each runner pants across the line, the timekeeper enters the number strapped to the runner's back. After the last man staggers by, and his number is entered, the program prints in one column his finishing position and in the second column his identifying number.

Answer to Program Planning Question 4 How can BASIC and the HP-85 help me find answers?

I'll use a one dimension, ten element array named R to store the identification numbers. The INPUT statement with its message will be within a FOR—NEXT loop. I'll call the finishing position of each runner P and each runner's identification number N. The key FOR—NEXT statements will look like this:

```
80 FOR P=1 TO 10
130 INPUT N
140 R(P)=N
150 NEXT P
```

Using the race results shown in Figure 134, page 256, array R will look like this when the FOR—NEXT loop is finished:

ARRAY R

Subscript	Variable	Value
1	R(1)	4
2	R(2)	7
3	R(3)	6
4	R(4)	1
5	R(5)	8
6	R(6)	3
7	R(7)	5
8	R(8)	10
9	R(9)	9
10	R(10)	2

Before the FOR—NEXT loop starts, the ten subscripted variables will not be initialized. The FOR—NEXT loop with its INPUT statement will initialize the ten variables of array R.

Figure 139 shows the listing. The flowchart and output may be found in Figures 140 and 141. Become familiar with the flowchart and listing to understand how the program works.

Arrays and FOR—NEXT loops

In "Race," you see one of the big advantages of arrays. Values can be put into arrays and displayed or printed from arrays using FOR—NEXT loops. Consider writing this "Race" program using ten simple variables instead of an array. You would need ten different INPUT statements and ten different PRINT statements. By replacing the eight key FOR—NEXT statements with the 20 INPUT and PRINT statements, you'd get twelve more statements. And this is only a ten element array.

```
 10 ! "RACE"
 20 OPTION BASE 1
 30 CRT IS 1
 40 PRINTER IS 2
 50 NORMAL
 60 CLEAR
 70 DIM R(10)
 80 FOR P=1 TO 10
 90 DISP
100 DISP "WHAT IS IDENTIFICATION
      NUMBER OF"
110 DISP "RUNNER FINISHING IN PO
    SITION"
120 DISP P;
130 INPUT N
140 R(P)=N
150 NEXT P
160 PRINT "FINISHING IDENTIFYING
    "
170 PRINT "POSITION    NUMBER"
180 PRINT
190 FOR P=1 TO 10
200 PRINT TAB(3);P;TAB(14);R(P)
210 NEXT P
220 END
```

FIGURE 139

Listing for "Race"

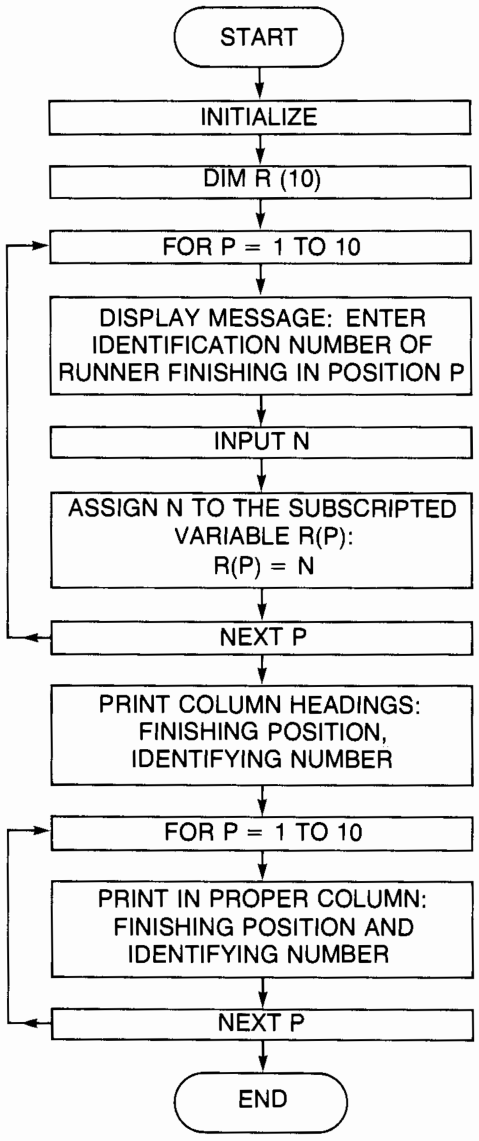

START
INITIALIZE
DIM R (10)
FOR P = 1 TO 10
DISPLAY MESSAGE: ENTER IDENTIFICATION NUMBER OF RUNNER FINISHING IN POSITION P
INPUT N
ASSIGN N TO THE SUBSCRIPTED VARIABLE R(P): R(P) = N
NEXT P
PRINT COLUMN HEADINGS: FINISHING POSITION, IDENTIFYING NUMBER
FOR P = 1 TO 10
PRINT IN PROPER COLUMN: FINISHING POSITION AND IDENTIFYING NUMBER
NEXT P
END

FIGURE 140
Flowchart for "Race"

```
FINISHING IDENTIFYING
POSITION    NUMBER

  1           4
  2           7
  3           6
  4           1
  5           8
  6           3
  7           5
  8          10
  9           9
 10           2
```

FIGURE 141
Output for "Race"

R ( ) = Array holding runner's identification numbers

P = First and second loop counter
= Runner's finishing position

N = Runner's identification number

PROBLEM: Write the "Wet" Program

Program Description "Wet" allows the user to ask and have displayed the daily rainfall in Drench, Oregon for any date in January 1978. The January rainfall data is in Figure 135, page 17-257. After giving the user the rainfall for his chosen date, the program prints all the dates whose rainfall was greater than the rainfall on the chosen date. If the chosen date happens to be the one that had the greatest rainfall for the entire month, an appropriate message is printed, since no date would have greater rainfall. Finally, the user is asked if he wishes to start again using another date, or if he wishes to stop.

Hint How will your program know if the rainfall on the chosen date is the heaviest rainfall in January? One way is to use a special variable as an internal indicator, called a **flag** by computer people. Let's call this variable C. Initialize C to zero, and increase it by one every time a date is printed whose rainfall is greater than the chosen date's rainfall.

After the chosen date's rainfall is displayed, have your program check to see if C is greater than zero. If C > 0, then the program did find at least one date whose rainfall was greater than the chosen date's rainfall. If C = 0, no date was found with greater rainfall; that is, the chosen date had the greatest rainfall for the month.

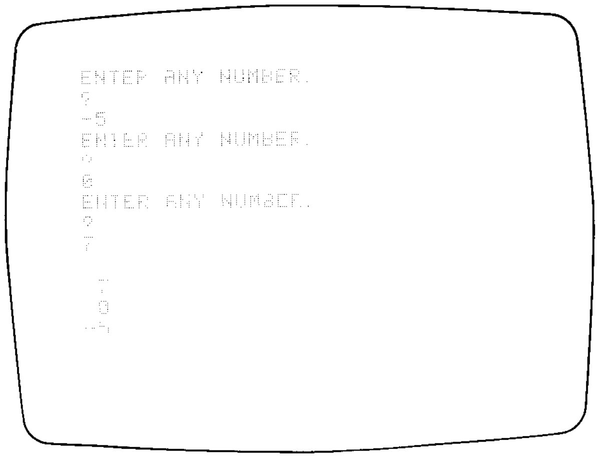
Your Turn Draw a flowchart and then write and run your program. If you're stuck, check out my flowchart, page H-47, then try to write "Wet." Compare your result with mine on page H-48.

Important When you're satisfied with your program, execute `STORE "WORK"` to store your program in your workspace on your BASIC Training cartridge. You'll be improving this program after I tell you how to sort numbers, and you'll save yourself grief if you have this first version available.

Sorting

Computers are made to order to handle a drudge job like sorting. Let's take a simple example. Say you enter three numbers into the HP-85, and ask it to display the three numbers in order of size, largest first, smallest last. Figure 142 shows the listing of "Sort," a program that will do the job. To help you understand how this sorting routine works, I'll give you a blow-by-blow description. You can also follow this description on the flowchart, Figure 143.

```
 10 ! SORT
 20 OPTION BASE 1
 30 CRT IS 1
 40 NORMAL
 50 CLEAR
 60 DIM N(3)
 70 FOR J=1 TO 3
 80 DISP "ENTER ANY NUMBER."
 90 INPUT N(J)
100 NEXT J
110 FOR K=1 TO 2
120 FOR L=1 TO 2
130 IF N(L)>N(L+1) THEN 170
140 T=N(L)
150 N(L)=N(L+1)
160 N(L+1)=T
170 NEXT L
180 NEXT K
190 DISP
200 FOR D=1 TO 3
210 DISP N(D)
220 NEXT D
230 END
```



```
ENTER ANY NUMBER.
?
-5
ENTER ANY NUMBER.
?
0
ENTER ANY NUMBER.
?
7

 7
 0
-5
```

FIGURE 142
Listing and output for "Sort"

N ( ) = Array for storing user entered numbers

J = First loop counter = number of numbers to be sorted

K = Second loop counter. The number of times this loop cycles (2) is one less than the number of numbers to be sorted (3)

L = Third loop counter. The remark for the K loop also applies here

T = Temporary storage

D = Fourth loop counter

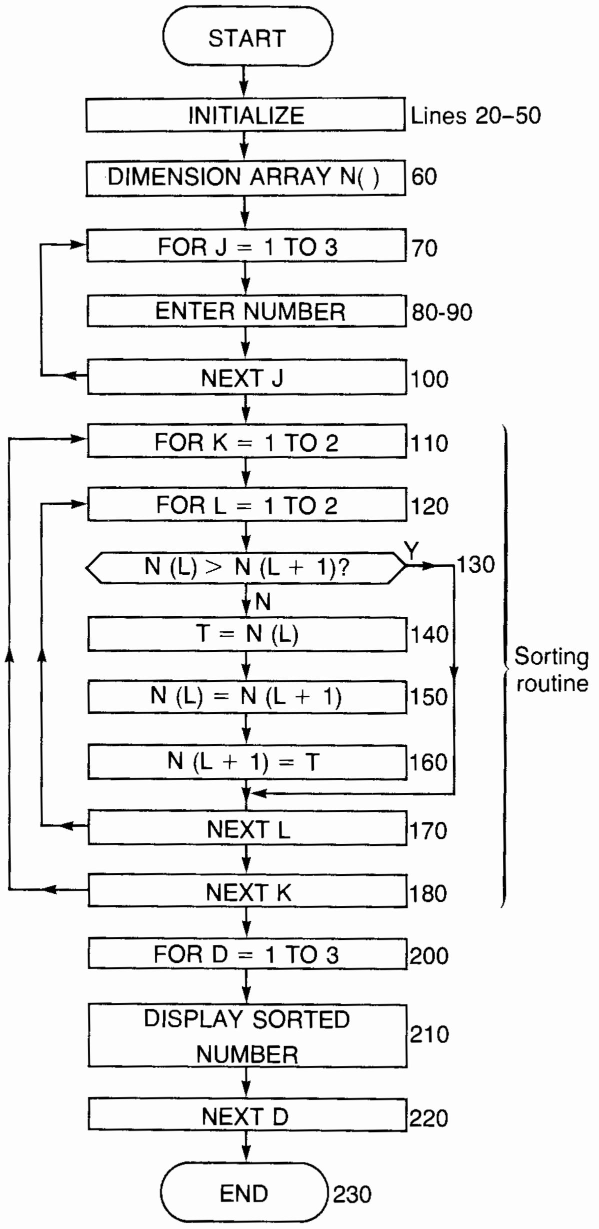


FIGURE 143
Flowchart for "Sort"

Blow-by-blow Description of "Sort"

```
60 DIM N(3)
```

Array N is dimensioned, but the subscripted variables are not initialized.

ARRAY N

Variable	Value
N(1)	
N(2)	
N(3)	

```
 70 FOR J=1 TO 3
 80 DISP "ENTER ANY NUMBER."
 90 INPUT N(J)
100 NEXT J
```

Each time through this loop, a value is entered and assigned to a variable of array N. The array is now initialized.

ARRAY N

Variable	Value
N(1)	-5
N(2)	0
N(3)	7

```
110 FOR K=1 TO 2
120 FOR L=1 TO 2
130 IF N(L)>N(L+1) THEN 170
```

Now:
K = 1
L = 1

Two nested FOR—NEXT loops are started. Statement 130 asks:

N(1) > N(1+1)?
N(1) > N(2)?
-5 > 0?

The answer is NO, so the next line is executed.

```
140 T=N(L)
```

T = N(1)
T = -5

This is the key, as will be evident as the plot unfolds.

IMPORTANT COMMENT

The final value for the two nested loops used in this type of sorting routine is one less than the number of items being sorted. In this example, *three* numbers are begin sorted, and the two nested loops start with

FOR K = 1 TO ②
FOR L = 1 TO ②

```
150 N(L)=N(L+1)
```

N(1) = N(1 + 1)
N(1) = N(2)
N(1) = 0

ARRAY N

Variable	Value
N(1)	0
N(2)	0
N(3)	7

Now the second value, 0, can be moved up to first position, since the variable T is temporarily saving the value, -5, originally assigned to N(1).

```
160 N(L+1)=T
```

N(1 + 1) = -5
N(2) = -5

ARRAY N

Variable	Value
N(1)	0
N(2)	-5
N(3)	7

Now the value saved by T can be put in position 2.

By using the ''temporary storage variable'' T, two adjacent numbers in the array can be switched.

```
170 NEXT L
```

L = 2

```
130 IF N(L)>N(L+1) THEN 170
```

N(2) > N(2 + 1)?

N(2) > N(3)?

-5 > 7?

The answer is NO, so the next line is executed.

```
140 T=N(L)
```

T = N(2)

T = -5

Again T is assigned -5 as the first step in switching -5 and 7.

```
150 N(L)=N(L+1)
```

N(2) = N(2 + 1)

N(2) = N(3)

N(2) = 7

ARRAY N

Variable	Value
N(1)	0
N(2)	7
N(3)	7

The second variable of array N is assigned the value of the third variable.

```
160 N(L+1)=T
```

N(2 + 1) = T

N(3) = -5

ARRAY N

Variable	Value
N(1)	0
N(2)	7
N(3)	-5

Now -5 and 7 are switched.

```
170 NEXT L
```

L = 3

Since 3 > 2, program execution proceeds to the next statement

```
180 NEXT K
```

K = 2

```
120 FOR L=1 TO 2
```

L = 1

```
130 IF N(L)>N(L+1) THEN 170
```

N(1) > N(1 + 1)?
N(1) > N(2)?
0 > 7?

Since the answer is NO, the next statement is executed.

```
140 T=N(L)
```

T = N(1)
T = 0

Here we go again. Now we're switching 0 and 7.

```
150 N(L)=N(L+1)
```

N(1) = N(1 + 1)
N(1) = N(2)
N(1) = 7

ARRAY N

Variable	Value
N(1)	7
N(2)	7
N(3)	-5

```
160 N(L+1)=T
```

N(1 + 1) = T
N(2) = 0

ARRAY N

Variable	Value
N(1)	7
N(2)	0
N(3)	-5

We know the HP-85 has finished the job. Does the HP-85 know it?

```
170 NEXT L
```

L = 2

```
130 IF N(L)>N(L+1) THEN 170
```

N(2) > N(2 + 1)?
N(2) > N(3)?
0 > -5?

Yes, so on to 170

```
170 NEXT L
```

L = 3

Since 3 > 2, proceed to 180

```
180 NEXT K
```

K = 3

Since 3 > 2, on to 190

```
190 DISP                            Output
200 FOR D=1 TO 3
210 DISP N(D)                          7
220 NEXT D                             0
230 END                               -5
```

The program displays its good work and ends.

This sorting routine works equally well for putting the smallest number on top and the largest on the bottom. Just change the conditional in statement 130 from > to <.

There is no special limit on the number of numbers that can be sorted with this scheme.

Often, this sorting routine is called a "bubble sort," since the largest (or smallest) number "bubbles" to the top as the program runs.

More on TRACE VAR and TRACE ALL

You may recall that `TRACE ALL` includes the functions of `TRACE` and `TRACE VAR`. The variable list for `TRACE VAR` may include either or both simple numeric variables and *array variables* used in the program. Array variables are shown on the variable list as, for instance, E( ). The command `TRACE VAR E(),B,F,X` would, when executed with `RUN`, trace the values of *all* subscripted variables of array E, and the values of numeric variables B, F and X. To emphasize, if array E had 1000 elements, executing `TRACE VAR E()` and `RUN` would trace the values of every one of the 1000 subscripted variables associated with array E. Since `TRACE ALL` includes the functions of `TRACE` and `TRACE VAR`, executing `TRACE ALL` and `RUN` would trace every subscripted variable of every array in the program, as well as every simple variable.

Another point: The programmability of `TRACE`, `TRACE VAR` and `TRACE ALL` adds to their bug killing power. In "Sort," whose listing is in Figure 142, page 17-266, you might wish to trace all variables only through the three key element-swapping statements 140, 150 and 160. A way to do this is to add these statements:

```
135 TRACE ALL
165 NORMAL
```

`NORMAL` cancels the effect of `TRACE`, `TRACE VAR` and `TRACE ALL`. Pressing (RESET) or executing `SCRATCH` also cancel `TRACE`, `TRACE VAR` and `TRACE ALL`, but they're not programmable.

PROBLEMS Using TRACE ALL

1. Enter "Sort," Figure 142, page 17-266, and print a complete line-by-line trace of the program, including the name and value of every variable (which, of course, includes every subscripted variable) whenever a variable changes value. Your printout should include the output from all `DISP` statements. The desired output for this problem is on page H-49.

2. Again using "Sort," print the same information as asked for in Problem 1 above, except limit the information to the three element-swapping statements, 140, 150 and 160. Page H-51 shows the output for this problem.

PROBLEM: Write the Augmented "Wet" Program

Program Description This enhanced version of "Wet" includes all the abilities of the original. At the end of the original version, the user is asked if he wishes to start over again using another date, or if he wishes to stop. This new version replaces the "stop" option with the option to have the rainfall amounts sorted by amount, smallest first, and then to have these amounts printed in order, smallest amount first, largest amount last.

Your Turn Add to your original "Wet" flowchart those items needed to chart the flow of this new "Wet." Then execute `LOAD"WORK"` to copy your earlier version into memory. Finally, modify it to satisfy the new description.

Note: It takes the HP-85 about 20 seconds to sort all 31 rainfall amounts. If you ask your program to sort these rainfall amounts, no output will occur until this 20 second sorting time has passed.

My flowchart: Page H-52.

My listing: Pages H-51 and H-52.

My output: Page H-53.

Summary of Chapter 17

- **Array:** An organized collection of numbers. Reference: Page 256.

- **One dimensional array:** A way of handling easily and efficiently a long (or short) list of numbers, each of which is represented by a single valued variable called a subscripted variable. Reference: Pages 256–17-259.

  - Example of an array:

ARRAY C5

Subscripted Variable	Subscript	Value
C5(0)	0	3
C5(1)	1	-17.3
C5(2)	2	431
C5(3)	3	0
C5(4)	4	.037
C5(5)	5	-6255.79

  - Array name: The name given to the entire list of values. It does *not* refer to an individual value. The array name may be any letter or letter followed by any single digit (the same choices as for a simple numeric variable).

    Examples: C5, A, M6, W, E, G1

    Reference: Page 17-258.

  - Array subscript: A number giving the position of a particular value in the array's list of values. A subscript may be a constant, a simple variable, an expression or another subscripted variable. Reference: Page 17-258.

  - Subscripted variable
    General form:

    [array name] ([subscript])

The name of something that may have one value or a succession of values in a program, one value at a time. Its definition and its use in a program are identical to the definition and use of a simple variable, except a subscripted variable is one of a related array of subscripted variables, while a simple variable is not.

Reference: Pages 17-258–17-259.

- Comparison of simple variables and array variables:

Simple Numeric Variable Terms and Examples	**Array Variable Terms and Examples**	**Definitions**
	Array variable A5, D, G7, M	The name of an organized collection of numbers, where each is represented by a subscripted variable.
Simple, one-valued, numeric variable A5, D, G7, M	Subscripted variable A5(3), D(37), G7 (D(37)), M(L + 3 ⋆ B)	The name of something that may have a single value or a succession of values in a program, one value at a time.
	Subscript (subscripts identified by arrows) $A(\underset{\uparrow}{3}),\ D(\underset{\uparrow}{37})$ $G(\underbrace{D(37)}_{\uparrow})$ $M(\underbrace{L + 3 \star B}_{\uparrow})$	The number part of a subscripted variable, like the 3 in R(3). It gives the array position or element where the value represented by R(3) is located.
	Element or position	A location for a value in an array. A ten element array would have ten locations, each available for one value. During the execution of a program, the values in these positions or elements may change, but the identifying number and location of each element remains the same. The element number is given by the subscript of the subscripted variable. For instance, the variable H(3) refers to the number in element 3 of array H.
Value 23, -6.21, 4003	Value 23, -6.21, 4003	A number

- `OPTION BASE 0`: **A BASIC word**
- `OPTION BASE 1`: **A BASIC word**

General forms:

[line number] `OPTION BASE 0`
[line number] `OPTION BASE 1`

- `OPTION BASE 0` means the first subscript of an array is *zero,* and the eighth subscript, for instance, is numbered *seven.*
- `OPTION BASE 1` means the first subscript of an array is numbered *one,* and the eighth subscript, for instance, is numbered *eight.*
- The HP-85 wakes up (when first turned on) with `OPTION BASE 0` in force. An `OPTION BASE 0` statement is used in a program only for documentation. If you want your subscripts' numbers to correspond to their positions; that is, if you want the first subscript of each of your program's arrays to be numbered one, you must have an `OPTION BASE 1` statement in your program. Its statement number must be lower than the number of any statement using an array variable or a subscripted variable.

  Reference: Pages 17-260–17-261.

- **Array dimension:** The maximum number of values an array may have is called its dimension. An array's dimension is equal to the number of elements in the array. Reference: Page 17-261.

- `DIM`: **A BASIC word**

  General form:

  [line number] `DIM` [array name] `(`[dimension]`),` [array name] `(`[dimension]`),` . . .

  The one or several dimension statements in a program define the dimensions of a program's arrays. They must appear earlier in the program than any statement using a dimensioned array variable or related subscripted variable *except* for the `OPTION BASE` statement. The `OPTION BASE` statement must preceed any dimension statement that dimensions an array. If an array's dimension is not defined in a program, its dimension automatically becomes 10 (`OPTION BASE 1`) or 11 (`OPTION BASE 0`).

  Example:

```
25 DIM H(100),R(50),J(5),F4(7)
```

  Reference: Page 17-261.

■ INTEGER: A BASIC word

General form:

> [line number] INTEGER [array name] ([dimension]) , [simple variable name] ,
> [array name] ([dimension]) , [simple variable name] , [simple variable name] , ...

INTEGER dimensions an array just like DIM does. In addition, it defines the listed simple variables and the listed array variables as having integer precision (see below). When an array variable is defined as having integer precision, it means all of its subscripted variables have integer precision.

Simple variables and array variables may be intermixed in any order in an INTEGER statement, and either kind of variable may be absent from an INTEGER statement.

In a program, any INTEGER statement or statements must appear *earlier* than any integer precision variable. But they must appear *after* any OPTION BASE statement.

Example:

```
15 INTEGER C6,A(30),M(80),K
```

Reference: Pages 17-261–17-263.

■ Sorting

A collection of numbers may be arranged in ascending or descending order of size using the following key sorting statements. Of course, any line numbers and any variable names may be used, and there is no special limit on the number of numbers that may be sorted by this technique.

Statement 520, as shown, puts the smallest of ten numbers at the top of the list. Changing the < symbol to > in statement 520 would put the largest number on top. The ten numbers are held in array N both before and after sorting.

```
500 FOR K=1 TO 9
510 FOR L=1 TO 9
520 IF N(L)<N(L+1) THEN 560
530 T=N(L)
540 N(L)=N(L+1)
550 N(L+1)=T
560 NEXT L
570 NEXT K
```

The upper limit of the two nested loops, 9 in this example, is always one less than the number of numbers being sorted.

Reference: Pages 17-266–17-273.

- **Tracing values of subscripted variables in a program.**

  - `TRACE VAR` [variable list] may be used as a command or BASIC word to trace not only the values of simple variables, but the values of all subscripted variables of any array whose variable name appears in the variable list, followed by a pair of empty parentheses.

    Example:

    ```
    50 TRACE VAR F,B,K(),L4(),D
    ```

  - `TRACE ALL`, used as a command or BASIC word, traces the values of all subscripted variables of all arrays as well as the values of all simple variables in the program.

    Reference: Pages 17-273–17-274.

Review Test for Chapter 17

Answers are on page 17-279.

1. What's wrong with this program?

```
 10 PRINTER IS 2
 20 CLEAR
 30 NORMAL
 40 OPTION BASE 1
 50 FOR C=1 TO 20
 60 A(C)=C*2
 70 NEXT C
 80 FOR P=1 TO 19
 90 PRINT P;".  ";A(P+1)*A(P)
100 NEXT P
110 END
```

2. This program sorts numbers according to size. Will the smallest or largest numbers appear at the top of the list?

```
10 OPTION BASE 1
20 DIM A(20)
30 FOR C=1 TO 20
40 DISP
50 DISP "PLEASE ENTER ANY NUMBE
   R."
60 INPUT A(C)
70 NEXT C
80 FOR K=1 TO 19
90 FOR L=1 TO 19
```

```
100 IF A(L)>A(L+1) THEN 140
110 T=A(L)
120 A(L)=A(L+1)
130 A(L+1)=T
140 NEXT L
150 NEXT K
160 FOR C=1 TO 20
170 PRINT A(C)
180 NEXT C
190 END
```

3. To cause the program in problem 2 to print the same list of numbers, but with the order reversed, what program change(s) (is, are) required?

4. This program is supposed to sort seven entered numbers. Why won't it work?

```
 10 OPTION BASE 1
 20 DIM A(7)
 30 FOR C=1 TO 7
 40 DISP
 50 DISP "PLEASE ENTER ANY NUMBE
    R."
 60 INPUT A(C)
 70 NEXT C
 80 FOR K=1 TO 7
 90 FOR L=1 TO 7
100 IF A(L)>A(L+1) THEN 140
110 T=A(L)
120 A(L)=A(L+1)
130 A(L+1)=T
140 NEXT L
150 NEXT K
160 FOR C=1 TO 7
170 PRINT A(C)
180 NEXT C
190 END
```

5. Without using the HP-85 (except to check your answer if you wish), what will statement 200 print when this program is run?

```
 10 OPTION BASE 1
 20 INTEGER A(8),C(6),E(5)
 30 DIM B(7),D(11)
 40 FOR C=1 TO 8
 50 A(C)=C+3.6
 60 NEXT C
 70 FOR C=1 TO 7
 80 B(C)=C+2
 90 NEXT C
100 FOR C=1 TO 6
110 C(C)=C+1.5
120 NEXT C
130 FOR C=1 TO 11
140 D(C)=C
150 NEXT C
160 FOR C=1 TO 5
170 E(C)=C-1.6
180 NEXT C
190 P=A(B(C(D(E(4)))))
200 DISP "P =";P
210 END
```

Hint: Write a table for each array showing subscripted variables and values.

Answers to Review Test Questions for Chapter 17

1. It needs a DIM statement located after `40 OPTION BASE 1` and before `60 A(C)=C*2`. A good choice would be

```
45 DIM A(20)
```

Reference: Page 17-261.

2. Largest.

Reference: Page 17-272.

3. In statement 100, change `>` to `<`.

Reference: Page 17-272.

4. Lines 80 and 90 should be:

```
80 FOR K=1 TO 6
90 FOR L=1 TO 6
```

Reference: Page 17-268.

5. `P = 10`

Reference: Pages 17-258–17-259; Pages 17-261–17-263.

Notes

Chapter 18

Fun and Games With Strings

Preview

In Chapter 18, you will:

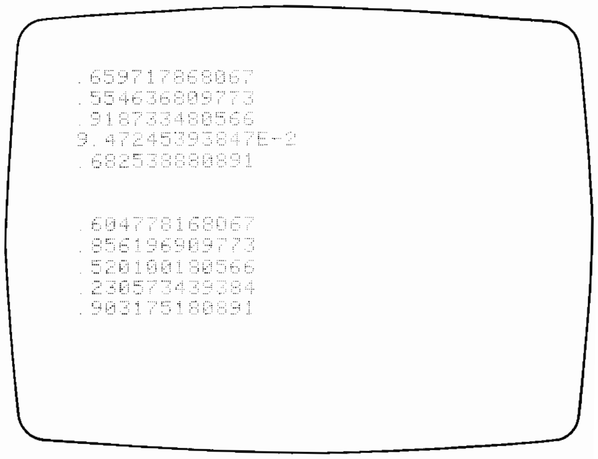
- Teach the HP-85 how to flip coins and roll dice.
- Learn how to increase your word power.
- Enter and run five example programs (one is only four lines long).
- Write *nine* (!) programs.

Games have always been a popular application for computers, and often games include an element of chance. BASIC allows you to deliberately introduce chance into your programs using a BASIC word and function you've previewed several chapters ago, `RANDOMIZE` and `RND`.

Figure 144 shows the listing and two typical outputs for "Randomize." `RANDOMIZE` and `RND` are formally introduced by this program. After this program, I'll discuss `RND`, then `RANDOMIZE`. But first, why don't you enter "Randomize," Figure 144, and run it a few times? Each time you run "Randomize," your output will be different. Almost certainly, none of your outputs will be the same as the two shown in Figure 144.

EXAMPLE: "Randomize" Program

```
10 ! RANDOMIZE
20 RANDOMIZE
30 FOR N=1 TO 5
40 DISP RND
50 NEXT N
60 END
```



```
.659717868067
.554636809773
.918733480566
9.47245393847E-2
.682538880891

.604778168067
.856196909773
.520100180566
.230573439384
.903175180891
```

FIGURE 144

Listing and two typical outputs for "Randomize"

RND: A Function

This function uses no argument. When executed repeatedly, the RND function generates a series of numbers that, in a practical sense, is completely random. The numbers range from zero to .999999999999. It is possible for the number to be zero, but it will never be exactly one. The sequence of numbers generated by RND does not exactly satisfy the strict mathematical definition of "random," but the HP-85 uses one of the best man-made generators available.

Random Number Seed

The HP-85 contains a random number generator that is activated by RND. This generator uses a number called a "seed" provided by the HP-85 or by the user which determines what sequence of numbers will be generated. When RANDOMIZE is *not* executed, the same seed is always provided by the HP-85 whenever the HP-85 is turned on, or when RESET is pressed. When RND uses the same seed, it generates the same sequence of random numbers.

EXAMPLE: "RND" Program

To discover for yourself how unrandom RND is when used without RANDOMIZE, enter the "RND" program listed below. Then press RESET and run the program. Repeat the same "press RESET, press RUN" sequence several times and compare the printed sets of five numbers.

```
 5 ! RND
10 FOR C=1 TO 5
20 PRINT RND
30 NEXT C
40 END
```

RANDOMIZE: A BASIC Word

To randomize this unrandom result, use RANDOMIZE.

If no number or numeric expression follows RANDOMIZE, the HP-85 supplies a different seed every time RANDOMIZE is executed. This seed is related to the HP-85's internal timing system, so as time changes, so does the seed delivered by the execution of RANDOMIZE. In this way, RANDOMIZE virtually guarantees a different sequence of random numbers each time a program containing RND is run. If a number or numeric expression follows RANDOMIZE, this number becomes the seed used by RND to generate its series of numbers. If this seed is the same number each time a program containing RANDOMIZE and RND is run, the same series of numbers is generated for each program execution. This is often desirable when testing a program containing RND.

Here are a few examples of RANDOMIZE [numeric expression]

```
120 RANDOMIZE .628731957
75 RANDOMIZE A*(3-A/L)
30 RANDOMIZE S
```

Random Integers Between Any Two Limits

To generate a series of random integers ranging from a small integer S to a larger integer L inclusive, use this expression:

```
INT((L+1-S)*RND+S)
```

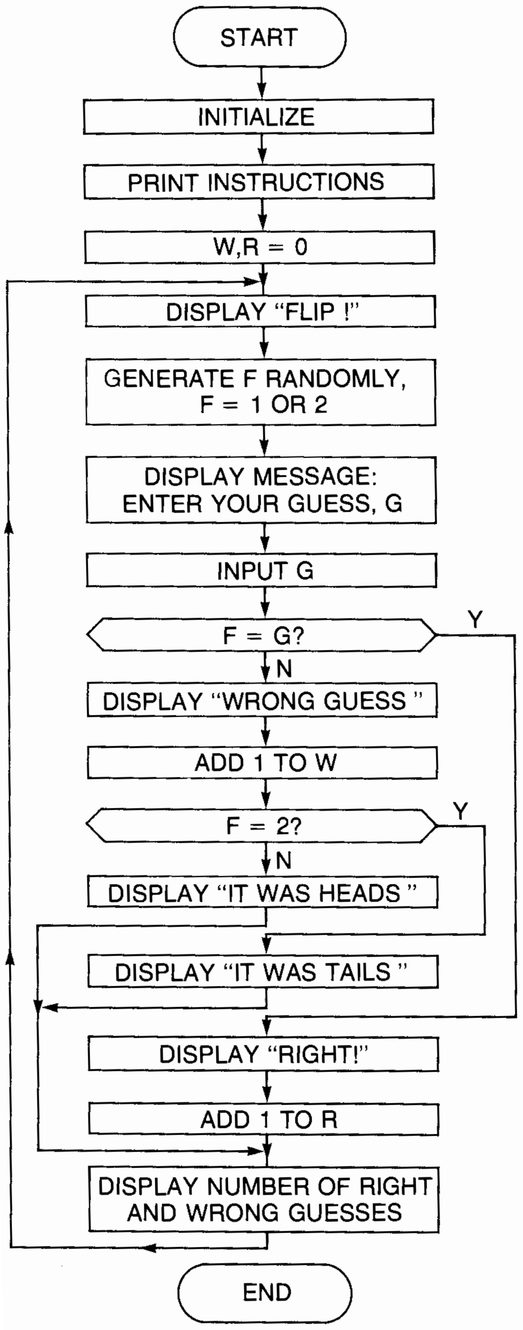
EXAMPLE: ''Flip'' Program

This is a coin toss guessing game. The flowchart and listing are shown in Figure 145. Enter and play this game if you wish. This statement:

```
270 F=INT(2*RND+1)
```

is the coin tosser. For RND values from zero to .499999999999 inclusive, F becomes one. For RND values from .500000000000 through .999999999999 inclusive, F becomes two.

Here's how	INT((L + 1 − S) * RND + S)
turns into	INT(2 * RND + 1):
The large integer is 2	L = 2
The small integer is 1	S = 1
So	INT((L + 1 − S) * RND + S)
becomes	INT((2 + 1 − 1) * RND + 1)
or	INT(2 * RND + 1)



```
 10 ! "FLIP"
 20 CRT IS 1
 30 PRINTER IS 2
 40 NORMAL
 50 CLEAR
 60 RANDOMIZE
 70 PRINT TAB(7);"COIN TOSS GAME
    "
 80 PRINT
 90 PRINT "CAN YOU OUTGUESS A CO
    MPUTER? I"
100 PRINT "WILL REPEATEDLY TOSS
    A COIN."
110 PRINT "EACH TIME YOU SEE 'FL
    IP' APPEAR,"
120 PRINT "I HAVE TOSSED IT. IF
    YOU THINK"
130 PRINT "IT CAME UP HEADS, TYP
    E A '1'"
140 PRINT "AND PRESS END LINE. I
    F YOU THINK"
150 PRINT "IT'S TAILS, TYPE '2'
    AND PRESS"
160 PRINT "END LINE. AFTER EACH
    GUESS, YOUR"
170 PRINT "SCORE WILL BE DISPLAY
    ED. WHEN"
180 PRINT "YOU WISH TO STOP, PRE
    SS PAUSE."
190 PRINT "HERE WE GO."
200 FOR P=1 TO 4
210 PRINT
220 NEXT P
230 W,R=0
240 DISP
250 DISP "FLIP!"
260 DISP
270 F=INT(2*RND+1)
280 DISP "WHAT'S YOUR GUESS";
290 INPUT G
300 DISP
310 IF F=G THEN 390
320 DISP "WRONG GUESS."
330 W=W+1
340 IF F=2 THEN 370
350 DISP "IT WAS HEADS."
360 GOTO 410
370 DISP "IT WAS TAILS."
380 GOTO 410
390 DISP "RIGHT!"
400 R=R+1
410 DISP
420 DISP "YOU'VE WON";R;"TIMES."
430 DISP "I'VE WON";W;"TIMES."
440 GOTO 240
450 END
```

FIGURE 145

Flowchart and listing for "Flip"

PROBLEM: Write the "Rand 1" Program

Write a program to generate random integers from one to 13, inclusive. Display five integers on one line each time the program is run. My version is on page H-53.

PROBLEM: Write the "Rand 2" Program

Modify "Rand 1" to display five random integers on one line from one to 100 inclusive.

Turn to page H-54 to see my listing.

PROBLEM: Write the "Rand 3" Program

Change "Rand 2" so that it displays eight random integers on one line between negative ten and positive twenty.

Page H-54 shows my program.

PROBLEM: Write the "Rand 4" Program

Write a program to display a set of 8 random integers for any range. The range limits are to be entered by the user. The integers are to be displayed on one line, unless the numbers are too large for eight of them to fit on one line. Write it for one unfamiliar with the HP-85. That is, write your `DISP` or `PRINT` statements for one who does not know what "enter" means.

See page H-54 for my flowchart and listing. Two different outputs are shown on page H-55.

PROBLEM: Write the "High Roller" Program

Program Description You and the HP-85 take turns rolling two dice. The high roller wins. The result of each roll is displayed. This display shows the value of each die and the sum. After the HP-85's roll is displayed, the display announces who won, or announces a tie. If no tie has occurred, the running totals of your wins and the HP-85's wins are displayed.

The program operates as an endless loop. At the start, displayed instructions include how to stop the program.

Your Turn Write your flowchart, and compare it against mine on page H-56. Maybe yours is better.

Now write your "High Roller" program. When you're finished, compare your program with mine shown on page H-55.

String Expressions

A string of characters within quotes is one kind of string expression you've used frequently. Examples:

```
"HELLO"
"MY NAME IS HP-85."
"12345"
"3*4/(7+3)"
```

The last two are strings, not numbers, since they are enclosed in quotation marks. They cannot be used in calculations. As you know, the only character or symbol that cannot be used in a string is the quotation mark itself. It is reserved for enclosing and identifying strings. A string of characters within quotes is often called a quoted or literal string, but it's more commonly called a string.

String Variables

BASIC has two kinds of variables:

Numeric variables: N, A3, C7, H(3)

String variables: N$, A3$, C7$

Here's a string variable in action:

Listing

```
10 D3$="STRING"
20 PRINT D3$
30 END
```

Output

```
STRING
```

Note how the assignment statement 10 assigns the ''value'' `STRING` to the string variable D3$.

A string variable name consists of a letter followed by a dollar sign or a letter followed by a single numeral followed by a dollar sign. There are 286 possible string variable names.

Strings Can Be Compared

Strings of characters and string variables can be compared in much the same way as constants and numeric variables. The following program demonstrates how this is done.

EXAMPLE: ''Spell'' Program

Limber your fingers and enter this program whose listing appears in Figure 146. After you've got it running OK, store it in your workspace by executing the `STORE"WORK"` command.

You'll be improving this program later, so it's wise to put it in a safe place, especially since you're going to scratch memory before you use ''Spell'' again.

```
 10 ! "SPELL" AN EXAMPLE
 20 CLEAR
 30 CRT IS 1
 40 NORMAL
 50 T$="INCREDIBLE"
 60 DISP "ON YOUR COMMAND, A WOR
    D WILL    FLASH ON THE SCREE
    N. IT WILL"
 70 DISP "APPEAR IN THE CENTER O
    F THE TOP ROW FOR A LITTLE O
    VER 0.1 SEC."
 80 DISP "WHEN IT DISAPPEARS, YO
    U WILL BE ASKED TO SPELL IT
    CORRECTLY."
 90 DISP
100 DISP "SHARPEN YOUR EYEBALLS
    AND GOOD  LUCK! PRESS CONT W
    HEN READY."
110 DISP
120 PAUSE
130 CLEAR
140 DISP TAB(10);T$
150 WAIT 100
160 CLEAR
170 DISP "TYPE THE WORD YOU JUST
     SAW AND  PRESS END LINE."
180 INPUT A$
190 DISP
200 IF A$=T$ THEN 320
210 DISP "SORRY, YOU'RE A LITTLE
     OFF. IF  YOU WOULD LIKE TO
    TRY AGAIN,"
220 DISP "PRESS Y, THEN END LINE
     IF YOU  WOULD RATHER SEE T
    HE CORRECT"
230 DISP "SPELLING, PRESS S, THE
    N END     LINE."
240 INPUT A$
250 IF A$="S" THEN 280
260 CLEAR
270 GOTO 140
280 DISP
290 DISP T$
300 DISP
310 GOTO 360
320 DISP "GOOD SHOW!"
330 WAIT 300
340 DISP
350 GOTO 360
360 DISP "THANKS FOR YOUR EFFORT
    "
370 END
```

FIGURE 146

Listing for "Spell"

Now I'll describe how the strings and string variables work in the "Spell" program.

In line 50, which is an assignment statement, the string variable T$ is assigned the "value" `INCREDIBLE`. This string variable is used first in line 140. Statement 140, together with 150 and 160, flashes the word `INCREDIBLE` on the screen. Line 180 is an `INPUT` statement asking for a string entry. When a series of characters, say `INCREDABLE`, is entered in response to line 180's question mark, the string variable A$ is assigned the value `INCREDABLE` by statement 180. In line 200, A$ is compared to T$, which means `INCREDABLE` is compared to `INCREDIBLE`. Since they are not equal, the program does *not* branch to line 320. Instead, the "Sorry" message of lines 210-230 is displayed.

When `INCREDABLE` was entered in response to lines 170 and 180, it could have been typed with or without quotation marks. Quotes *must* be used if a single string entry includes a comma, like `PARIS, FRANCE`. If you entered `PARIS, FRANCE` in response to lines 170 and 180:

1. You would be a terrible speller, and
2. The HP-85 would think you had entered two strings, and would give you an error message.

In the absence of quotes, any comma entered in response to a string variable `INPUT` statement is taken as a string boundary. If quotes are used, only a comma between two quoted strings is considered to be a string boundry. So if you entered "PARIS, FRANCE" in response to lines 170 and 180, you would still be a horrible speller, but the HP-85 would accept your entry as one string.

Discover something else about string inputs. Change line 50 to

```
50 T$="500"
```

Now run "Spell" twice. The first time, enter "500" with quotes. The second time, enter 500 without quotes. How does the HP-85 know you intend 500 without quotes to be a string of three characters instead of a number which could be used in calculations? The answer lies in line 180. If a string variable is used in an INPUT statement, almost any group of characters entered in response to the ? is considered a string, including numerals.

I say "almost any group of characters" because the comma and quotation mark are special cases, and because there is a restriction on the number of characters that can be entered as one string. I'll tell you more about string length later. Now I'll tell you more about the comma and quotation mark.

How would you enter the number 1,000,000 as a single string? This situation is very similar to the Paris, France case discussed above. You would enter it enclosed within quotes, like this: "1,000,000". If you entered 1,000,000 without quotes, the HP-85 would think you had entered three strings, 1 and 000 and 000. Remember, in the absence of quotes, any comma is taken as a string boundry.

Mixed Numeric and String Inputs

As you know, one INPUT statement can.ask for several numerical inputs. In addition, an INPUT statement can ask for several string inputs and also for a group of intermixed numeric and string inputs. For example:

```
210 INPUT A,B,C,D(4)
335 INPUT A$,B$,C$,D$
500 INPUT A$,B,C$,D(4)
```

When responding to an INPUT statement that asks for several inputs, be sure to separate entries with commas and enclose with quotation marks all strings that include commas. Also, enter your inputs in the right order. In responding to example statement 500 above, you would enter a string, number, string and number in that order.

Strings Can Be Combined

As demonstrated below, the semicolon can be used to combine strings and string variables into a single statement.

EXAMPLE: "Small Program" Program

To illustrate this, here is a more sophisticated relative of the old "Semicolon, Comma" program you used in Chapter 6. Enter this program listed in Figure 147. Note the single space in line 10 between the last L in SMALL and the quotation mark.

```
 10 A$="SMALL "
 20 B$="PROGRAM"
 30 DISP "ENTER VALUES FOR A, B
    AND C."
 40 INPUT A,B,C
 50 PRINT "SEMICOLON:"
 60 PRINT A$;B$;"-10"
 70 PRINT "15";"20"
 80 PRINT A;B;C
 90 PRINT B;C;A
100 PRINT
110 PRINT "COMMA:"
120 PRINT A$,B$,"-10"
130 PRINT "15","20"
140 PRINT A,B,C
150 PRINT B,C,A
160 GOTO 30
170 END
```

```
SEMICOLON:
SMALL PROGRAM-10
1520
-10  15  20
 15  20 -10

COMMA:
SMALL                PROGRAM
-10
15                   20
-10                   15
 20
 15                   20
-10
```

FIGURE 147

Listing of "Small Program" and output for A=−10, B=15 and C=20

Now run "Small Program" and confirm the output shown in Figure 147. That is, enter values for A, B, and C of -10, 15 and 20. Then enter a number of other values, both negative and positive, and observe the outputs.

Here are some truths demonstrated by this program:

1. A **semicolon** between two string variables or quoted strings in a PRINT or DISP statement results in the strings being printed or displayed with *no space* between them.

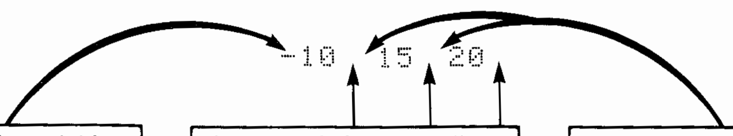
   When line 60 was executed, a space appeared between SMALL and PROGRAM only because a space was built into A$ in line 10. No space was built into the string variable B$ in line 20 or the string "-10" in line 60, so when line 60 was executed, PROGRAM and -10 were printed with no space between them.

   Remember: If you want one or more spaces to appear between strings when you use quoted strings and/or string variables in a DISP or PRINT statement, include the space(s) in your quoted string or in the assignment statement that defines your string variable (like line 10).

2. A **comma** between string variables and/or quoted strings in a PRINT or DISP statement results in the strings being printed or displayed with a *wide space* between them.§ Most often, such wide spacing is not desirable, so remember to use the semicolon.

§ See page 84 in your Owner's Manual for details about this wide spacing.

3. Review a bit of Chapter 16 by looking at the output of line 80 in detail, when A, B, and C have the values -10, 15 and 20:



When the numeric variable value is negative, a minus sign appears before the printed or displayed value.

The space following a displayed or printed value of a numeric variable is always left blank.

When the numeric variable value is positive, a space appears before the printed or displayed value.

4. A **semicolon** between two numeric variables in a `PRINT` or `DISP` statement results in the values being printed or displayed with *no additional space* between them beyond the spaces discussed in 3 above.

5. A **comma** between two numeric variables in a `PRINT` or `DISP` statement results in the values being printed or displayed with a *wide space* between them.

Null String

A special kind of string is two quotation marks enclosing nothing, as shown in line 190 below. Note that the two quotation marks are right next to each other. There is not even a space between them. A null string is entered in response to an `INPUT` statement by pressing only (END LINE).

The following program segment illustrates one use for a null string. In line 225, A$ is tested to see if it is a null string.

```
140 DISP "IF YOU WOULD LIKE SOME
    INSTRUC-"
150 DISP "TIONS, PRESS Y, THEN E
    ND LINE."
160 DISP "TO SKIP INSTRUCTIONS,
    PRESS END"
170 DISP "LINE ONLY."
180 DISP
190 A$=""
200 INPUT A$
205 DISP
```

```
210 CLEAR
215 DISP
220 DISP
225 IF A$="" THEN 430
230 DISP "THIS PROGRAM COMPLETEL
    Y"
240 DISP "SIMULATES THE CLASSICA
    L RIVER"
250 DISP "CROSSING PROBLEM IN TH
    E"
260 DISP "FOLLOWING FORM:"
```

If it is, the program branches around the instructions. Presumably someone using this program the fourth time would not wish to read the instructions again. Friendly programming suggests it should be easy for a user to bypass the instructions. Pressing only one key, (END LINE), is one easy way.

Dimensioning String Variables

Here are the promised words about the length of strings. If a string variable represents a string over 18 characters long, including spaces, it must be dimensioned in a `DIM` statement using *square brackets only.* For example:

```
10 DIM A$[20],E$[42]
```

Some observations:

1. Dimensioning strings to be 18 characters or less is optional but desirable, for these reasons:

   a. To conserve HP-85 memory. For each undimensioned string, the HP-85 reserves memory for 18 characters.

   b. To show a user of your listing in a clear way all the string variables your program uses.

2. A `DIM` statement may combine dimensions for strings and arrays. Use commas to separate each item. For example:

```
40 DIM Z$[80],R(40),W(5),D$[7]
```

3. A `DIM` statement must be located in a program before any other reference to the dimensioned variable(s) occur(s) in that program.

Strings in DATA Statements

After you see quoted strings used in `DATA` statements in the following example, I'll discuss the subject in more detail.

EXAMPLE: "Guess," a Weather Forecasting Program

Program Description The user enters the name of the day whose weather he wants forecasted. He then enters two magic numbers, and the HP-85 will print two statements:

1. The weather forecast for the chosen day.
2. The percent chance the forecast is correct.

Inspect my flowchart, Figure 148 and my listing for "Guess," Figure 149.

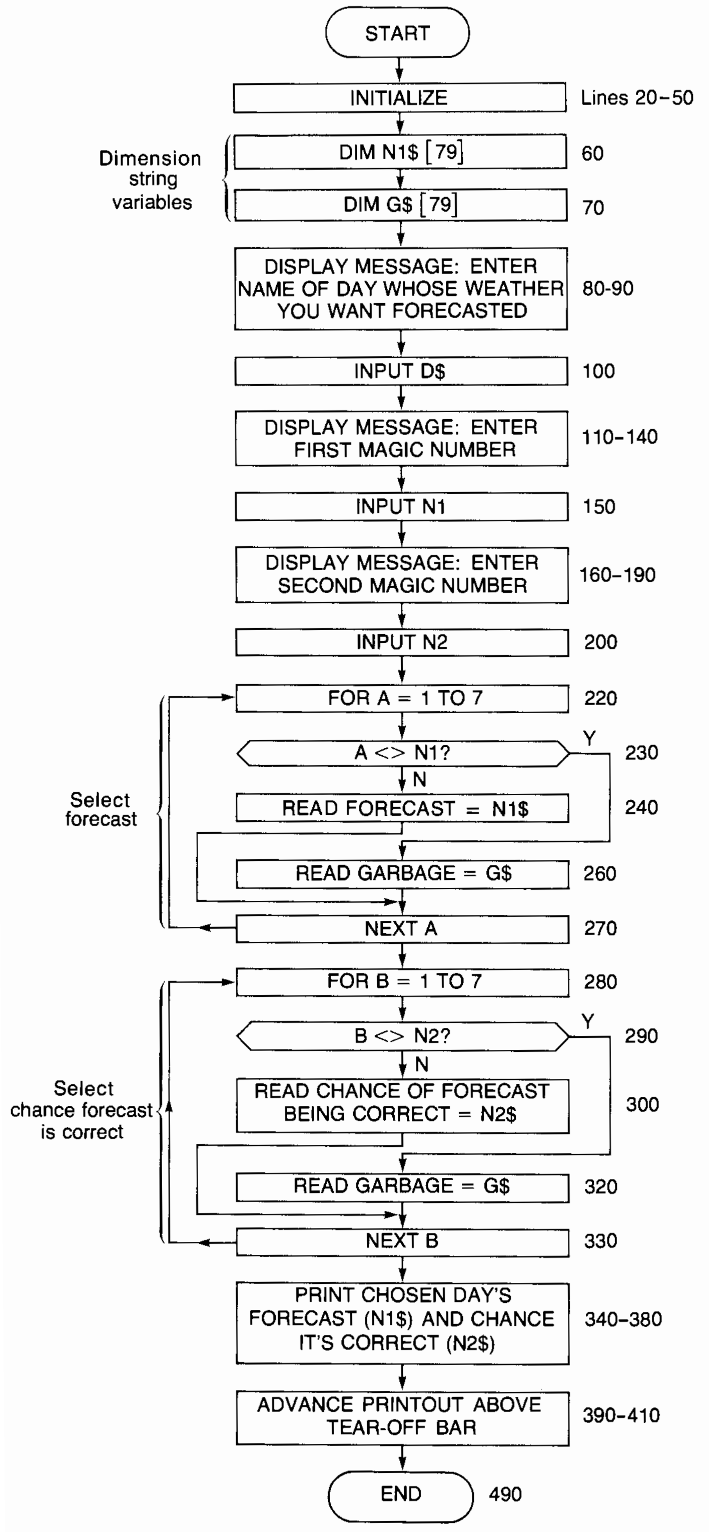

START
INITIALIZE
Lines 20–50
Dimension string variables
DIM N1$ [79]
60
DIM G$ [79]
70
DISPLAY MESSAGE: ENTER NAME OF DAY WHOSE WEATHER YOU WANT FORECASTED
80-90
INPUT D$
100
DISPLAY MESSAGE: ENTER FIRST MAGIC NUMBER
110–140
INPUT N1
150
DISPLAY MESSAGE: ENTER SECOND MAGIC NUMBER
160–190
INPUT N2
200
Select forecast
FOR A = 1 TO 7
220
A <> N1?
Y
230
N
READ FORECAST = N1$
240
READ GARBAGE = G$
260
NEXT A
270
Select chance forecast is correct
FOR B = 1 TO 7
280
B <> N2?
Y
290
N
READ CHANCE OF FORECAST BEING CORRECT = N2$
300
READ GARBAGE = G$
320
NEXT B
330
PRINT CHOSEN DAY'S FORECAST (N1$) AND CHANCE IT'S CORRECT (N2$)
340–380
ADVANCE PRINTOUT ABOVE TEAR-OFF BAR
390–410
END
490

FIGURE 148

Flowchart for "Guess"

```
10  ! GUESS
20  CRT IS 1
30  PRINTER IS 2
40  NORMAL
50  CLEAR
60  DIM N1$[79]
70  DIM G$[79]
80  DISP "WHAT DAY'S WEATHER DO
    YOU WANT"
90  DISP "FORECASTED (ENTER NAME
     OF DAY)";
100 INPUT D$
110 DISP
120 DISP "WHAT IS YOUR FIRST MAG
    IC"
130 DISP "FORECASTING NUMBER (CH
    OOSE 1,2,"
140 DISP "3,4,5,6 OR 7)";
150 INPUT N1
160 DISP
170 DISP "WHAT IS YOUR SECOND MA
    GIC"
180 DISP "FORECASTING NUMBER (CH
    OOSE 1,2,"
190 DISP "3,4,5,6 OR 7)";
200 INPUT N2
210 DISP
220 FOR A=1 TO 7
230 IF A<>N1 THEN 260
240 READ N1$
250 GOTO 270
260 READ G$
270 NEXT A
280 FOR B=1 TO 7
290 IF B<>N2 THEN 320
300 READ N2$
310 GOTO 330
320 READ G$
330 NEXT B
340 PRINT
350 PRINT D$;"'S WEATHER WILL BE
    "
360 PRINT N1$;"."
370 PRINT "THE CHANCE OF THIS FO
    RECAST"
380 PRINT "BEING CORRECT IS ";N2
    $;"%."
390 FOR P=1 TO 4
400 PRINT
410 NEXT P
420 DATA "TERRIBLE. FREEZING RAI
    N FOLLOWEDBY A SEVERE COLD S
    NAP"
430 DATA "HEAVY SNOW, ACCUMULATI
    NG TO TWO METERS"
440 DATA "BEAUTIFUL. SUNNY, VISI
    BILITY 200MILES, HIGH 23 DEG
    REES C"
450 DATA "STRONGLY INFLUENCED BY
     HURRICANEADOLF. RAINFALL 20
     CENTIMETERS, WINDS 180 KM/H
    R"
460 DATA "RAIN, SLEET AND SNOW",
    "SEVERE THUNDERSTORMS, WINDS
          GUSTING TO 70 MPH"
470 DATA "EVEN HOTTER. HIGH WILL
     APPROACH 50 DEGREES C"
480 DATA "70","30","90","20","50
    ","100","10"
490 END
```

FIGURE 149

Listing for "Guess"

Program Discussion Lines 120 through 200 ask the program's user to select two magic numbers. For the first magic number, he chooses one number out of seven. The DATA statements contain a string for each of the seven possible choices. If he selects, for instance, three as his first magic number, the program must move the DATA statement pointer through all seven strings, even though only one will be used. Otherwise, the pointer will not be positioned properly for the second magic number selection. Look at the first magic number loop starting at line 220 and using loop counter A. If A does not equal the first magic number, N1, the program branches to READ statement 260, whose only purpose is to move the DATA statement pointer.

More About Strings in DATA Statements

As you know, DATA statements may contain a number of numeric constants, like the daily rainfall figures in your "Wet" program. In addition, DATA statements may contain strings, as the "Guess" program shows. Each DATA statement string must be separated from another data string or numeric constant by a comma. A data string may be enclosed in quotation marks or it may be left unquoted. However, remember that *quotes must be used if the data string includes a comma.*

Quotes around DATA statement strings are like parentheses in numerical expressions. Sometimes they're necessary and sometimes they're not. But they never hurt. So use them freely to be safe and to make things clearer for you.

More Truths about DATA Statements

1. As you know, strings and numbers may be mixed together in a DATA statement. However, when a READ statement is executed, the READ variable must agree with the type of DATA statement item being read. That is, when the DATA statement pointer points to a string, like "MAY", the next READ statement executed must use a string variable, like P$. Likewise, when the pointer points to a number which is intended for calculations, like 352, the next READ statement executed must use either a simple numeric variable, like N, or a subscripted variable, like G(7).

2. Every string or numeric constant in a DATA statement must be complete. That is, you may not continue a string or constant from one DATA statement to the next.

PROBLEM: Write an Improved "Spell" Program

Program Description Execute LOAD"WORK" to copy your earlier "Spell" program into memory. Now make these improvements:

1. Modify lines 230 and 250 so the program's user can see the correct spelling by pressing only (END LINE). Remember, pressing only (END LINE) in response to an INPUT statement enters a null string.

2. Add a DATA statement or statements that contain ten spelling words of your choosing. A READ statement in your program should read each word before it is flashed on the screen. Remember, a DATA statement and all other statements may contain no more than 95 characters. That's three full lines less one character position.

3. If a word is spelled correctly, advance to the next word. If it's spelled wrong, give the user the option of trying to spell the same word again or advancing to the next word. If he chooses to continue before spelling the word correctly, show him the correct spelling of the word he missed before asking him to flash the next word.

4. Keep track of the number of words flashed. After the user spells the last word correctly or gives up, tell him the test is over, and give him the option of taking the test over again.

Your Turn Do you best at adding these changes to the "Spell" program you started earlier. My listing for this second "Spell" version is on page H-57.

When you're satisfied with your version, execute STORE"WORK" to record it into your workspace.

In the next problem, you'll improve "Spell" further. If the HP-85's memory somehow gets scrubbed as you're working on the next problem, you'll be very happy to have this backup on your cartridge.

PROBLEM: Write a Further Improved "Spell" Program

Program Description Starting with the improved "Spell" you just finished, make these changes:

1. Give the user the option of seeing or skipping the initial instructions.

2. Add scoring. Give ten points for each of the ten words spelled correctly, whether on the first try or later. At the end of the test, express the total score as a percent.

Your Turn Make a healthy stab at putting these improvements into your "Spell" program. To see my third version of "Spell," turn to pages H-57 and H-58.

After you complete this third edition of "Spell," store it as a backup copy by executing `STORE"WORK"`.

Now get ready for the final enhancement.

PROBLEM: Improve "Spell" Program One More Time

Program Description Start with the most recent improvement of "Spell" which is in your workspace, and which may still be in memory, and add these abilities:

1. Modify the scoring. Give ten points only if the word is spelled correctly on the first try. If it is spelled correctly on any later try, give five points. Modify your instructions to reflect this scoring change.

2. Increase the time the word stays on the screen by 0.1 second each time the user tries again to spell it correctly.

Your Turn Put these final changes into your "Spell" program. My final version of "Spell" is on pages H-58 and H-59.

PROBLEM: Write the "THROW" Program

Here is a dice game you play with the HP-85 which gives you the advantage of an option denied to the HP-85. It works like this:

A game consists of 12 rolls of three dice. You roll first, and you and the HP-85 take turns. The scoring is as follows:

Each die different:	Sum of dice
One pair:	3 times sum of dice
Three of a kind:	18 times sum of dice

Whoever get the highest score after 12 rolls wins the game.

The option given to you alone comes when you roll a pair. You then have a choice of either:

1. Taking your score and turning the dice over to the HP-85, or

2. Refusing your pairs score, keeping the dice, and rolling the odd die in an effort to get three of a kind with its higher score.

However, every throw of that single die counts as one of your 12 rolls. Since the HP-85 never throws a single die after rolling a pair, but instead always turns the dice over to you, the HP-85 might have several rolls left after you finish your last roll. In that case, you just sit there and watch the HP-85 add to its score while your score stays fixed.

Program Description The "THROW," or "To Throw or Not to Throw" program simulates this game. Let's say you're the human player. The program offers you the option of reading or skipping instructions. You are then invited to roll the dice. The three values of the dice are displayed. If they are all different or three of a kind, your current total score is also displayed, and the HP-85 then rolls. If you throw a pair, your score is not shown. Instead, you are offered the choice of ending your turn, seeing your current total, and letting the HP-85 throw; or rolling the odd die in a try for three of a kind. If you roll again, the screen shows the number of spots on each of the three dice. Of course, the count for two of the dice would not change. If the try is successful, your new total score is displayed, and the HP-85 rolls. If it fails, you are again offered the same two choices. If you keep trying for three of a kind in this way and always fail, you would use up all of your 12 rolls. In any event, your turn would end with the display of your current total score or your final score. Then the HP-85 rolls, the number of spots on each of its dice is displayed, followed by the display of its total score. Then it's your turn once more, unless one of two situations exist:

1. You have already rolled your 12th roll, in which case the HP-85 continues to roll until it completes its 12th roll, ending the game.
2. The HP-85 has just completed its 12th roll, in which case the game is over.

Each score that's displayed is a cumulative score; that is, the current total score for that player. The end of the game is indicated by having each score announced by a different message, one that presents the idea of final score.

The program uses string variables for words or phrases that appear in several program statements.

Hints

1. Use a separate assignment statement using `RND` to assign to each die's variable the result of that die's roll. If the human player, after rolling a pair, elects to try for three of a kind, use a separate statement using `RND` to generate the new roll of the odd die. Determine which of the three die variables does not belong to the pair, then assign the new roll count to that variable.

   Next, display the result of the most recent roll by executing the statement that displays the values of the three die variables. Two of these values would not change from the last time the same statement was executed. If the new roll of the odd die showed the same count as before, all three displayed values would not change.

2. The phrases I use to announce each cumulative score and the final scores are:

   YOUR TOTAL SCORE:
   YOUR FINAL SCORE:
   THE HP-85'S TOTAL SCORE:
   THE HP-85'S FINAL SCORE:

   I use five string variables in different combinations for these messages.

3. I use two counters. One counter counts the human's rolls, the other counts the HP-85's rolls.

4. I use a **flag**, which is a special variable that acts as a traffic cop directing program flow. The flag's value starts as zero and is assigned the value one after the human rolls his 12th roll. If the human's 12th roll is a pair, he is not supposed to roll again for three of a kind. The flag's value of one prevents the human from rolling again.

5. If the HP-85 has just rolled and has at least one more roll left, and if the human has already completed his last roll, the HP-85 should immediately roll again. In this case, I use the flag's value of one plus the HP-85's roll counter to make the next roll an HP-85 roll.

Your Turn The writing of this program can benefit from a flowchart. I suggest you draw one. Play several imaginary games using your flowchart to see if it works. The flowchart for my program is on page H-61.

When you're ready to check your program against mine, see my listing on pages H-59 and H-60.

I've recorded my version on your BASIC Training cartridge in case you'd like to see how my version works. If you would like to run my version, *be sure to execute* `STORE "WORK"` *to avoid having your only copy disappear when you load my version.*

To get my version into the HP-85's memory, execute `LOAD "THROW"`.

Summary of Chapter 18

- `RND`: **A function**

  When executed repeatedly, `RND` generates a series of pseudo-random numbers. Each number X of this series is between 0 and 1 (1.00000000000). If `RND` is used in a program without `RANDOMIZE` (see below), the same sequence of random numbers is generated every time the program is run just after

  1. The HP-85 is switched ON, or
  2. RESET is pressed.

  Reference: Pages 282–18-283.

- **Random number seed:** The HP-85's random number generator uses a number called a "seed" to determine what sequence of numbers will be generated by `RND`. Reference: Page 18-283.

- `RANDOMIZE`: **A BASIC word**

  General form:

  [line number] `RANDOMIZE` [optional random number seed]

Examples:

```
115 RANDOMIZE
283 RANDOMIZE 0.71365927
20 RANDOMIZE PI/9
```

- When RANDOMIZE is used *without* a seed, the HP-85 supplies its own seed derived from its internal timing system. Executing RANDOMIZE before RND in a program virtually assures a *different* sequence of random numbers every time the program is run.
- When RANDOMIZE is used *with* a seed, then when RANDOMIZE is executed before RND in a program, the *same* sequence of random numbers is generated every time the program is run.

  Reference: Pages 18-283–18-284.

- **Random integers between any two integers.**

  To generate a sequence of random integers from a smaller integer S to a larger integer L inclusive, use this RND expression:

```
INT((L+1-S)*RND+S)
```

Examples: To generate random integers

1. From 1 to 2 (coin flip), use:

```
INT(2*RND+1)
```

2. From 1 to 6 (die throw), use:

```
INT(6*RND+1)
```

3. From -5 to 5, use:

```
INT(11*RND-5)
```

Reference: Page 18-284.

- **String:** A quoted or literal string is a series of characters, symbols and/or spaces within quotation marks. It's more commonly called a string. Reference: Pages 18-286–18-287.
- **String variable name**

  A letter followed by a dollar sign, or a letter followed by a single numeral followed by a dollar sign.

  Examples: C$, M3$, W0$, A$

  A string variable name is assigned the "value" of a quoted string in an assignment statement, just as a simple numeric variable name or subscripted numeric variable name is assigned a numeric value.

  Reference: Page 18-287.

- **Comparing strings:** Strings and string variables can be compared much as numbers and numeric variables can be compared. Reference: Pages 18-287–18-288.

- **Strings and** `INPUT`

  When a string is entered in response to an `INPUT` statement, quotation marks must be typed if the string includes one or more commas. Otherwise, the use of quotes is optional.

  One `INPUT` statement can use intermixed string variables, simple numeric variables and/or subscripted variables. Examples:

```
150 INPUT F4(4),P$,K
35 INPUT A$,B$,A,B
```

  When values are entered in response to such `INPUT` statements, the values must be entered in the order of the variables. For instance, when responding to line 150, a number, a string and a number must be typed onto the HP-85's screen in that order, separated by commas, then (END LINE) is pressed.

  Reference: Pages 18-288–18-289.

- **Strings in** `DISP` **and** `PRINT` **statements**

  In a `DISP` or `PRINT` statement, strings, string variables, numeric variables and constants can be combined. They should be separated by **semicolons** to avoid the generally undesirable wide spacing produced when commas are used as separators.

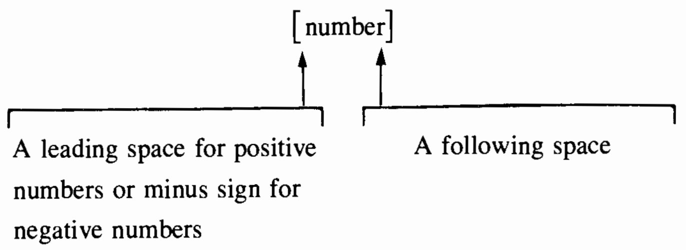
  Example:

```
45 DISP A$;F3;"NEVER";V3$;G3(6)
```

  Reference: Pages 18-289–18-290.

- **Numbers in** `DISP` **and** `PRINT` **statements**

  When a numeric constant, variable, or expression is used in a `DISP` or `PRINT` statement, the following extra character and/or space(s) are always displayed or printed in addition to the number:


[number]
A leading space for positive numbers or minus sign for negative numbers
A following space

  Reference: Page 18-291.

- **Null string**

A null string is two quotation marks right next to each other. In this example, D$ is assigned the value of the null string:

```
105 D$=""
```

A null string is entered in response to an `INPUT` statement by pressing only one key: (END LINE).

Reference: Page 18-291.

- **`DIM` statement used for string variables**

If a string variable is over 18 characters, including spaces, it must be dimensioned in a `DIM` statement using **square brackets**. One dimension statement can dimension both string variables and array variables. String variables with 18 or less characters may be dimensioned in a `DIM` statement. The HP-85 assigns a dimension of 18 characters to all string variables not dimensioned in a `DIM` statement. Use commas to separate items.

Example:

```
15 DIM A$[30],A(30),G3$[5]
```

Reference: Pages 18-291–18-292.

- **Strings in `DATA` statements**

`DATA` statements may include strings and numbers intermixed, separated by **commas**. If a string *includes* a comma, the string must be enclosed by quotes. If the string does *not* include a comma, the string may be enclosed in quotes or it may omit quotes.

When a `DATA` statement is read, the `READ` statement variable must correspond to the type of data (string or numeric) being read.

One data string may *not* be split between two `DATA` statements.

Reference: Pages 18-292–18-295.

Review Test for Chapter 18

This test covers material in earlier chapters as well as in Chapter 18. It is on your BASIC Training cartridge. Execute `LOAD"TEST18"`. The program will direct you to the answers.

Take heart! This is the last test in this course! Of course, Chapter 19 has a few interesting programs for you to write.

Cannibals and Missionaries

Preview

In Chapter 19, you will:

- Learn how, in one statement, you can tell your program many different places it can go.
- Learn how to tell your program to go on a journey and then return.
- Write a flexible arithmetic quiz program in six stages.
- Write, as your final masterpiece, a program which completely simulates the classical river crossing problem.

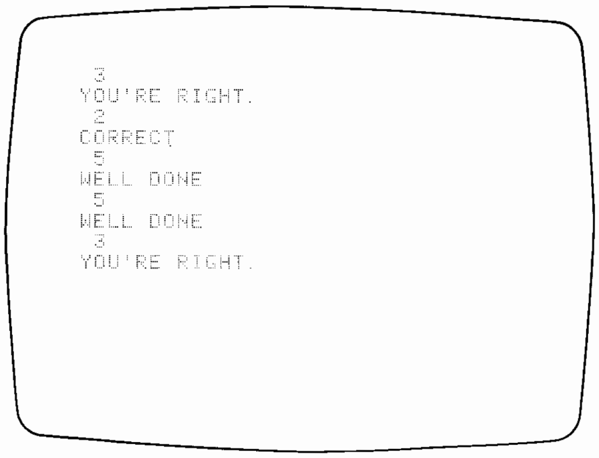
ON ...GOTO: A BASIC Word

Sometimes you reach a "Have you stopped beating your wife? (or husband?)" point in your program where a simple YES or NO won't do. There are times when you'd like your program to take one of three, four or more paths, depending on circumstances. You guessed it—the BASIC word `ON ... GOTO` can accommodate your needs.

EXAMPLE: "Congratulations" Program

Look at the listing in Figure 150, which is a renumbered and slightly modified portion of Program "REVIEW" from your BASIC Training tape. See how statement 50 directs the program to one of five congratulatory messages depending on the whim of statement 30.

```
 10 ! CONGRATULATIONS
 20 R=0
 30 N=INT(5*RND+1)
 40 DISP N
 50 ON N GOTO 60,80,100,120,140
 60 DISP "YOU DO GOOD WORK."
 70 GOTO 150
 80 DISP "CORRECT"
 90 GOTO 150
100 DISP "YOU'RE RIGHT."
110 GOTO 150
120 DISP "EXCELLENT"
130 GOTO 150
140 DISP "WELL DONE"
150 R=R+1
160 END
```

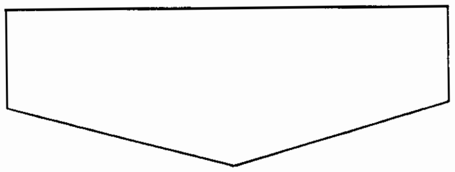


```
 3
YOU'RE RIGHT.
 2
CORRECT
 5
WELL DONE
 5
WELL DONE
 3
YOU'RE RIGHT.
```

FIGURE 150

Listing and typical outputs for "Congratulations"

Figure 151 shows the flowchart for this "Congratulations" program. This flowchart includes the fourth and final flowchart symbol I'll give you, which is the ON...GOTO symbol:



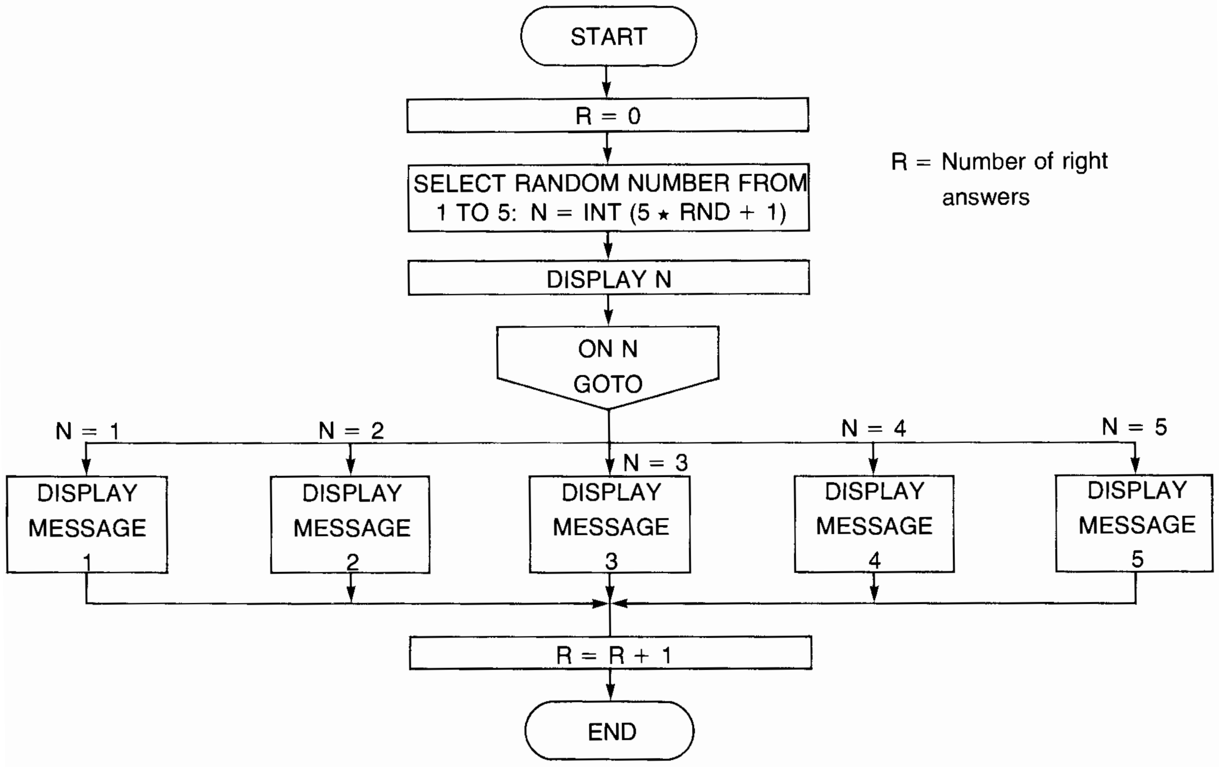


FIGURE 151

Flowchart for "Congratulations"

This key phrase, ON...GOTO, is called the computed GOTO statement, because the item between ON and GOTO can be an expression as well as a simple variable. "Congratulations" could be written with N in line 50 replaced by the expression for N in line 30. Also, when the expression in an ON...GOTO statement is evaluated, the result is rounded to the nearest integer. The program then branches to the line number whose position in the ON...GOTO statement list is given by this integer. For instance, when INT(5*RND+1) in statement 30 evaluates to 3, the "Congratulations" program branches to line 100, the third line number in the statement list 60, 80, 100, 120, 140 in line 50.

When the expression in an ON...GOTO statement is rounded to a value under one, an error is produced. An error is also produced if the rounded value is larger than the number of statements in the list.

PROBLEM about Rounding

May `INT(5*RND+1)` in statement 30 be simplified to `5*RND+1` because of the rounding function of `ON...GOTO`? If you wish to experiment, you might wish to enter the program in Figure 150 with one change: delete `INT` from statement 30. When you run your test program, run it 20 or 30 times to give the round function of `ON...GOTO` a good chance to generate all of its possible values.

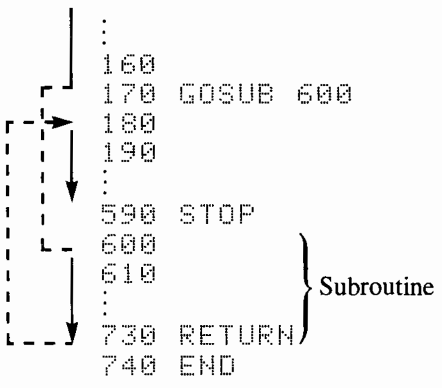
For the answer, see page H-62.

GOSUB—RETURN: A BASIC Word

Often programmers use a particular routine several times in a program. A handy way to direct program execution to that routine and back again, from several points in the program, is to use `GOSUB—RETURN`.

Here's how it works:

Path followed by program flow

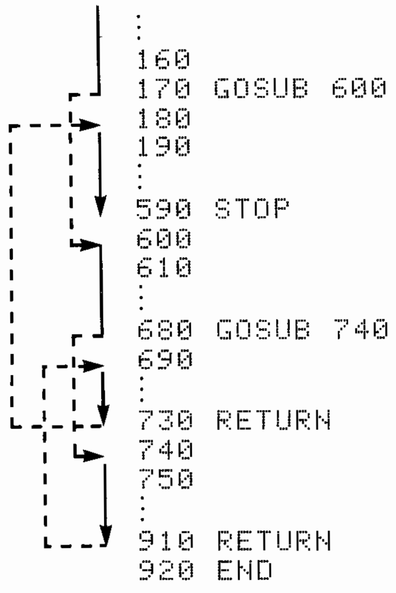


```
:
160
170 GOSUB 600
180
190
:
590 STOP
600
610
:
730 RETURN
740 END
```

Program execution jumps from 170 to 600, then proceeds through the subroutine to `730 RETURN`. At that point, program execution returns to 180, the statement immediately following `170 GOSUB 600`. A statement like 170 is sometimes referred to as a call to a subroutine. The first statement of a subroutine (600 in this example) may be any kind of statement, including `FOR—NEXT`, `ON...GOTO`, `!`, etc.

The subroutine referenced by 170 may contain another reference to a second subroutine. For instance:

Path followed by program flow.



```
:
160
170 GOSUB 600
180
190
:
590 STOP
600
610
:
680 GOSUB 740
690
:
730 RETURN
740
750
:
910 RETURN
920 END
```

When a second subroutine is referenced from within a first subroutine, the two subroutines are said to be nested. In the example just above, if statement 820, say, referenced a third subroutine, the nesting would be two deep.

The HP-85 allows nesting up to 255 levels deep, although available memory would no doubt be used up first. In practical terms, you can nest as many subroutines as you want, just as you can nest as many `FOR—NEXT` loops as you want.

EXAMPLE: "Press CONT" Program

Figure 152 shows a portion of program "CH3" from your BASIC Training cartridge.

```
1410 CRT IS 1
1420 PRINTER IS 2
1430 NORMAL
1440 CLEAR
1450 DISP TAB(9);"HEWLETT-PACKAR
     D"
1460 DISP TAB(6);"GETTING DOWN T
     O BASIC"
1470 DISP TAB(4);"CONTINUATION O
     F CHAPTER 3"
1480 DISP
1481 DISP TAB(13);"REMINDER" @ D
     ISP TAB(12);"----------" @
     DISP
1482 DISP "IF YOU PRESS"&R2$&"IN
     STEAD OF"
1483 DISP C2$[2,5]&", RELAX - TH
     E PROGRAM WILL"
1484 DISP "START OVER. IF YOU PR
     ESS ANY    OTHER KEY, YOU'R
     E STILL OK. JUST"
1485 DISP "PRESS"&C2$[1,5]&" AS
     THOUGH NOTHING HAD"
1486 DISP "HAPPENED."
1487 GOSUB 4000
1488 DISP
1490 DISP H$
1500 DISP
1510 DISP "DURING THIS LAST PART
      OF CHAPTER"
1520 DISP "3, LOOK FOR MY HELP M
     ESSAGES."
1530 DISP "I'VE SPRINKLED THEM T
     HROUGHOUT"
1540 DISP "THE TEXT YOU'LL SEE O
     N SCREEN"
1550 DISP "AND PRINTER. THEY GIV
     E PAGE REF-"
1560 DISP "ERENCES TO THE HELP S
     ECTION LO-"
1570 DISP "CATED IN THE SMALLER
     OF THE TWO"
1573 DISP "BOOKS INCLUDED WITH T
     HIS PAC."
1575 DISP
1580 GOSUB 4000
1590 DISP "RIGHT NOW, YOU'RE GOI
     NG TO MAKE"
1600 DISP "SURE YOUR TAPE IS NOT
      ERASED"
1610 DISP "ACCIDENTLY. FOR HELP,
      SEE PAGE"
:
4000 DISP
4010 DISP "TO PROCEED, PRESS"&C2
     $;
4020 DISP
4030 PAUSE
4065 CLEAR
4070 RETURN
4080 END
```

FIGURE 152

Portion of program "CH3"

The section of "CH3" shown in Figure 152 uses a few programming techniques which are not covered in this course, but which are fully covered in your Owner's Manual.

One programming technique shown in Figure 152 which is covered in this course is the use of the subroutine. Lines 1487 and 1580 direct program execution to line 4000, the first line of the subroutine that prints the message you have come to know and love. The `PAUSE` in this subroutine holds the display so you can read it.

Every time this message and `PAUSE` were executed when you ran "CH3," this subroutine was responsible. If you list "CH3," you'll see many `GOSUB 4000` statements.

Each time line `4070 RETURN` is executed, program execution returns to the line number immediately following the most recently executed `GOSUB` statement. At any point during the running of a program, the number of `RETURN` statements executed cannot exceed the number of `GOSUB` statements executed previously. For example, this sequence of execution is OK:

```
        GOSUB, RETURN, GOSUB, GOSUB, RETURN, RETURN
```

The following is a list of all the GOSUB and RETURN statements in a program. It represents an impossible sequence of execution:

```
        GOSUB, RETURN, GOSUB, RETURN, RETURN
```

The last RETURN is not associated with any GOSUB in the program. If the execution of this last RETURN were attempted, an error message like this would be displayed:

```
        Error 51 on line 60 : RETURN W/O
         GOSUB
```

PROBLEMS: Write the "MATH" Series of Programs

This series of programs is also called "Arithmetic Quiz 1," "Arithmetic Quiz 2," and so on, through "Arithmetic Quiz 6." This series of six programs plus one additional program constitute your final exam. As promised, no more quizzes!

Program Descriptions You'll write this program in six stages, where each new stage builds on the last. I'll describe the sixth and final version first, then the contribution of each earlier stage to the final program. I have a listing for the first stage and a program on your BASIC Training cartridge for the final stage.

The final version offers the user drills in addition, subtraction, multiplication, division and exponentiation. The math problems are constructed from randomly selected numbers whose upper and lower limits are chosen by the user. After every 10 problems, the user's score is given, and the user may chooose a new type of problem and/or new limits for the random numbers. The final version of "MATH" also offers a sixth choice besides +, −, ★, / and Λ, called pot luck. Pot luck randomly selects one of the five kinds of math functions for each problem.

All six versions operate as endless loops.

Here's a description of each of the six stages:

1. This program drills in addition only. The limits are fixed to be 99 and 2, which means the largest and smallest possible problems are 99 + 99 and 2 + 2.

   After every 10 problems, a percentage score for those 10 problems is displayed. Then another 10 problems are displayed one at a time, and so on. The program operates as an endless loop. At the beginning, the user is told what the number limits are (99,2), and then instructions are displayed. These instructions say something like: `TYPE YOUR ANSWER, THEN PRESS CONT. YOUR SCORE WILL BE SHOWN AFTER EVERY 10 ANSWERS.` The limits and instructions are shown only once for each running of the program.

2. Same as stage one, except subtraction problems are also offered. The user is asked to choose between addition and subtraction problems. The next ten problems that are displayed will be of his chosen type.

After his score on these ten problems is displayed, he is again invited to choose between addition and subtraction for the next ten problems. Subtraction problems use the same fixed 99 and 2 limits as do addition problems.

3. Same as stage 2, except the user may elect to accept the program's limits of 99 and 2, or he may choose his own limits for the random numbers from which the + and − problems are constructed. After the score for the last 10 problems is displayed, the current limits are displayed, and the user is given an opportunity to change them. Whenever the user is asked if he wishes to change his limits, it is easy for him to determine what his current limits are.

4. This version adds division problems as a third user choice. To avoid remainders with division problems, the program uses this scheme: Say A and B are the two random numbers to be used to construct the division problem. Q is defined as Q = A ⋆ B. Then the division problem is asked like this: What is Q/A? The correct answer is B.

   The program protects against division by zero.

   If the user does not change the division limits, the program gives limits for A and B of 15 and 2, using the definitions for A and B given above. These limits mean the numerator Q = A ⋆ B ranges from 15 ⋆ 15 = 225 to 2 ⋆ 2 = 4, and the denominator A has limits of 15 and 2. The program's choice for addition and subtraction limits remain the same, 99 and 2. After the user chooses the type of problem, he is asked if he wishes to change the limits for that kind of problem only.

5. Multiplication and exponentiation are added, so the user may now choose among all five types of problems. The program's limits for multiplication are the same as for division, 15 and 2. The user may choose new limits for ⋆ and / problems, but not one set for ⋆ and another set for /. The program uses two sets of limits for exponentiation problems. For the problem B Λ E = ?, the program's limits for B are 9 and 2, and the program's limits for E are 3 and 2.

   The program protects against zero raised to an exponent equal to or less than zero; that is, against 0 Λ 0 and 0 Λ (negative number).

6. The final problem choice is added: pot luck. When pot luck is chosen, the type of each problem is chosen randomly from among +, −, ⋆, / and Λ. When the user chooses pot luck, he is then asked about the limits for + and − problems, for ⋆ and / problems, and for Λ problems, but only if that kind of problem is presented during the next 10 problems, and then only when that kind of problem is presented for the first time.

   Confused? Let me explain it another way. The following table shows the types of problems that might be asked during a pot luck set of 10 problems. This table also shows when limit questions would be asked. Note that there are two types of limit questions:

   a. Do you want to change limits?

   b. If yes, what are your new limits?

This table assumes that the user chose pot luck, and that he also chose to change limits.

Problem Number	Type of Problem	Limit Question Asked?
1	+	Yes (+, − limits)
2	/	Yes (⋆, / limits)
3	⋆	No
4	−	No
5	Λ	Yes (Λ limits, both pairs)
6	+	No
7	⋆	No
8	/	No
9	⋆	No
10	−	No

Problems 4, 6 and 10 would use the limits chosen for problem 1. Problems 3, 7, 8 and 9 would use the limits chosen for problem 2. If problem 5 had been another kind of problem, say ⋆ instead of Λ, the Λ limits questions would not have been asked at all during these ten problems.

When the 10th problem is answered, and the score is given, the user then chooses again between +, −, ⋆, /, Λ and pot luck, and the cycle repeats.

Your turn If you wish, draw a flowchart for the first stage. Then write your first stage program and compare it with mine whose listing is on page H-62. Don't start work on the second stage until you're satisfied with your first stage program. After completing each stage, store it in your workspace (`STORE "WORK"`) to give you a backup if you need it.

When you finish your last stage, print a listing, *be sure to execute* `STORE "WORK"` *to preserve your program,* then execute `LOAD "MATH"` to get my version into memory. Run it and compare its operation against yours.

The road through all six stages may be long, but treating the completion of each stage as a separate goal makes the entire project less imposing. This illustrates a good technique to use when faced with writing a program having a complicated definition. Break the final definition into smaller, more manageable parts, and finish and test each part before starting the next.

PROBLEM: Write the "CAMIS" or "Cannibals and Missionaries" Program

This program was inspired by and is based on the HP-65 Users' Library Program No. 02286A by Mordecai Schwartz, M.D. The following description is a direct quote from the program documentation sent to us by Dr. Schwartz.

Program Description

"This program completely simulates the classical river crossing problem in the following form:

"Three missionaries and three partially civilized cannibals must cross a river with a boat that can hold no more than three passengers. At no time may cannibals outnumber missionaries at either bank or on the boat lest the cannibals regress to an earlier mode of behavior! Further, cannibal(s) left alone on the boat will run off with it after launching. Missionaries, cannibals and boat are all initially on the left bank.

"You have 3 choices:

1: Missionary boards

2: Cannibal boards

3: Boat leaves bank"

When choice one or two is made, the program checks to see if the person being loaded actually exists on the bank where the boat is currently moored. If not, a message is displayed, and the user is invited to try another choice. When the boat leaves a bank, the program checks to see if any of the problem rules are broken. If so, an appropriate message is displayed, and the user is invited to start all over again. If the user gets all three missionaries and all three cannibals on the right bank, the program presents a success message.

Your Turn This is a challenging program. A good way to start is to draw a flowchart. Tape several pieces of paper together to give yourself room. Try to solve the puzzle using your flowchart, and correct your flowchart if necessary. When you think you've got a workable flowchart, you may either take a look at my flowchart on page H-63, or if you're filled with confidence, you may move directly to the writing of your program. When you're finished writing, print a listing, *store your program in your workspace to preserve it,* then execute `LOAD"CAMIS"` to copy my version into memory. Finally, compare your program against mine.

Summary of Chapter 19

- `ON...GOTO`**: A BASIC word**

  General form:

  [line number] `ON` [numeric variable or expression] `GOTO` [line number] `,` [line number] `,` ...

  Example:

```
250 ON G/H GOTO 470,620,840,1130
,1710
```

  When an `ON ... GOTO` statement is executed, the numeric variable or expression is evaluated to the nearest integer. If the nearest integer is below one or greater than the number of line numbers in the list, an error message is given. If the evaluated nearest integer is from one to the number of line numbers in the list (five in the above example), the program branches to the line number whose position in the list corresponds to the evaluated nearest integer.

In the above example, if the nearest integer to the value of G/H were 3, the program would branch to line 840.

Reference: Pages 302–19-304.

- GOSUB—RETURN: **A BASIC word**

General form:

```
[line number] GOSUB [first line number of subroutine]
:
[last line number of subroutine] RETURN
```

Example:

```
:
295 GOSUB 400
300
:
400◄─────────── first line of subroutine
:
525 RETURN◄── last line of subroutine
:
```

When 295 GOSUB 400 is executed, program execution transfers to line 400. The statements in the subroutine are then executed, one after the other, until 525 RETURN is executed.

At that point, program execution switches to line 300, the statement immediately following 295 GOSUB 400.

In one program, many different GOSUB statements can transfer program execution to the same subroutine. Also, a subroutine may contain one or many GOSUB statements referencing other subroutines. These other subroutines may lie within the first subroutine or outside it.

When a subroutine itself contains GOSUB statements, the subroutines are said to be nested.

To avoid an error, the number of RETURN statements whose execution is completed or attempted must not exceed the number of GOSUB statements already executed in the same program.

Reference: Pages 19-304–19-307.

Chapter 20

Where Do I Go From Here?

You have finished the course. Congratulations!

If you want to get a good feeling about how much you've learned, load and run "CH3" again, just for fun.

You've got a lot of powerful BASIC tools in your kit, but you don't have them all. I'll give you some thoughts on where and how to get additional knowledge and experience.

1. Read your HP-85 Pocket Guide to get a feeling for the full power of the HP-85's BASIC. Especially browse through the lists of commands, BASIC statements, graphics statements and BASIC predefined functions.

2. Among the many important additional BASIC words or statements available are these two:

   a. IF — THEN [executable statement], Owner's Manual, page 105. Example:

   ```
   90 IF B=5 THEN A=G*(N+5)
   ```

   b. $\left.\begin{array}{l}\texttt{PRINT}\\\texttt{DISP}\end{array}\right\}$ USING— IMAGE, Manual page 161. These give detailed and exact control over where characters and spaces are printed and displayed. One example: Using these in a FOR — NEXT loop, you can print this: rather than this:

   ```
    8          8
    9          9
   10         10
   ```

3. The HP-85 can handle two dimensional arrays (tables of rows and columns). For instance, programs dealing with budget and expense records may be written using two dimensional arrays. One dimension (rows) could be budget or expense categories, while the other dimension (columns) could be months. Start on page 119 in your Manual for the two dimensional array story.

4. There is much you can do with string variables. See your Manual starting on page 124.

5. Multistatement lines: Using the "@" symbol as a separator (delimiter), several statements may be put into a single line. See Manual page 92.

6. Graphs can be constructed easily with labeled axes and additional text. To see the power of the HP-85's graphics, load and run the "KEYS" and "DEMO" programs on your BASIC Training cartridge. For full details on graphics, see your Owner's Manual, starting on page 197.

7. You've been introduced to some functions already, such as SIN(X), INT(X), ABS(X), and RND.

You can also define your own function in your program and use it in other parts of your program. The function definition may be many statements long. Your Manual has more on this starting on page 145.

8. The four keys at the upper left of your keyboard, plus KEY LABEL, represent another HP-85 power you should investigate. The eight special function keys, four unshifted plus four shifted, offer eight ways a running program may be interrupted from the keyboard and ordered to branch to a line number defined by your program. See Manual page 154.

9. You can ''print'' quoted strings and numbers on your tape as well as on your printer. This is done with `PRINT#`. The `READ#` statement allows you to read this information back from tape to your program. To use `PRINT#` and `READ#`, you must first initiate a data file on your cartridge with a `CREATE` statement, then ''open'' the file with an `ASSIGN#` statement. The whole story on data files is in your manual, starting on page 180.

10. The HP-85 has an internal timer that can be set like a clock. See the ''TIMER'' program in your Standard Pac for an example. You can use this internal timer in your programs. One use is to program a branching to occur after a certain time interval, which is independent of the execution of your other program statements. There are three mutually independent internal timers you may use in this way. See Manual pages 56 and 156.

11. For more programming practice, write the programs presented as problems in your Manual. Flowcharts and listings for these programs are in Manual Appendix F, page 309, and the problems themselves are scattered throughout the programming section of your Manual starting on page 71.

12. Another way to improve your programming skills is to study the listings of programs in your Standard Pac, as well as in other application pacs. You might pick up some useful programming tricks. You also might see places where our programs could be improved. We're not perfect.

13. Our Customer Support Department in Corvallis, Oregon ((503) 757-2000, address on inside front cover) would be pleased to give you programming help. You can appreciate that the time it takes to completely plan, write, debug and document even the simplest program makes it necessary for us to refuse requests for custom programming. However, we will help where we can to answer specific questions. Our ability to help you increases as you give us more detailed information, such as inputs, outputs and equations.

Part II: An HP-85 Demonstration

This program presents examples of graphics power, prints all of the characters and symbols grouped by function, and plays a little Bach.

The program will ask you to use the special function keys whose locations are shown on page 316. To see the functions assigned to these keys, press KEY LABEL at any time after the instructions are printed.

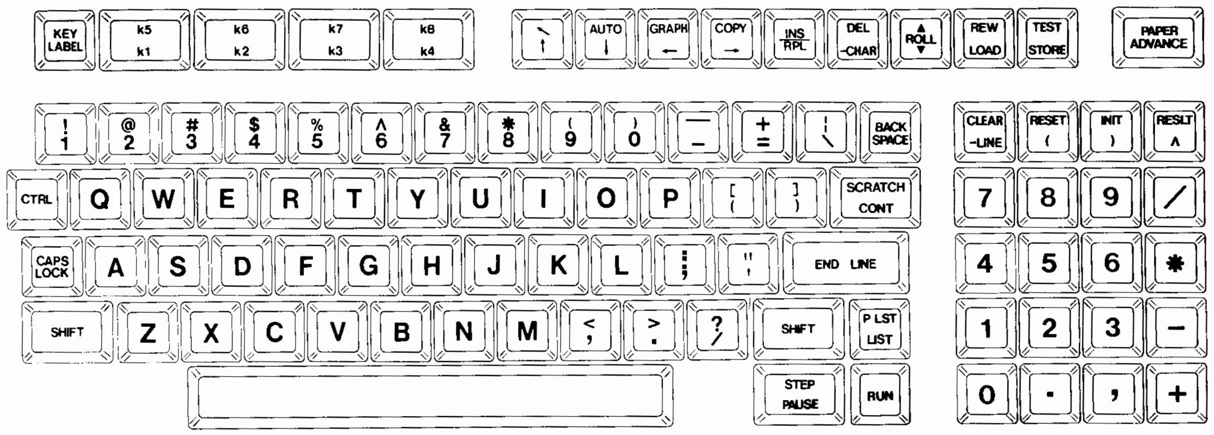
To start the program, insert your BASIC Training cartridge into your HP-85 and execute `LOAD"DEMO"` (press LOAD " [§] D E M O " END LINE). When the tape drive light goes out, press RUN and the program will print instructions.

[§] To type ", press SHIFT + ".

Part III: The HP-85 Keyboard

This program shows you on the screen where keys are located and what each key does.

The program will ask you to use the special function keys whose locations are shown below:



Special function keys $\boxed{\text{k5 k1}}$, $\boxed{\text{k6 k2}}$, $\boxed{\text{k7 k3}}$ and $\boxed{\text{k8 k4}}$, plus the $\boxed{\text{KEY LABEL}}$ key.

To select $\boxed{\text{k1}}$, press $\boxed{\text{k5 k1}}$.

To select $\boxed{\text{k2}}$, press $\boxed{\text{k6 k2}}$.

To select $\boxed{\text{k3}}$, press $\boxed{\text{k7 k3}}$.

To select $\boxed{\text{k4}}$, press $\boxed{\text{k8 k4}}$.

To select $\boxed{\text{k5}}$, press $\boxed{\text{SHIFT}}$ + $\boxed{\text{k5 k1}}$.

To select $\boxed{\text{k6}}$, press $\boxed{\text{SHIFT}}$ + $\boxed{\text{k6 k2}}$.

To select $\boxed{\text{k7}}$, press $\boxed{\text{SHIFT}}$ + $\boxed{\text{k7 k3}}$.

To select $\boxed{\text{k8}}$, press $\boxed{\text{SHIFT}}$ + $\boxed{\text{k8 k4}}$.

To see the functions assigned to these keys, press $\boxed{\text{KEY LABEL}}$ at any time after the instructions are printed.

To start the program, insert your BASIC Training cartridge into your HP-85 and execute `LOAD"KEYS"` (press $\boxed{\text{LOAD}}$ $\boxed{\text{"}}$§ $\boxed{\text{K}}$ $\boxed{\text{E}}$ $\boxed{\text{Y}}$ $\boxed{\text{S}}$ $\boxed{\text{"}}$ $\boxed{\text{END LINE}}$). When the tape drive light goes out, press $\boxed{\text{RUN}}$ and the program will print instructions.

§ To type ", press $\boxed{\text{SHIFT}}$ + $\boxed{\text{" ;}}$.

Notes

Index

Bold page numbers refer to summary pages.

HEWLETT
PACKARD

For additional information please contact your local Hewlett-Packard Sales Office.

00085-90052

Printed in U.S.A.