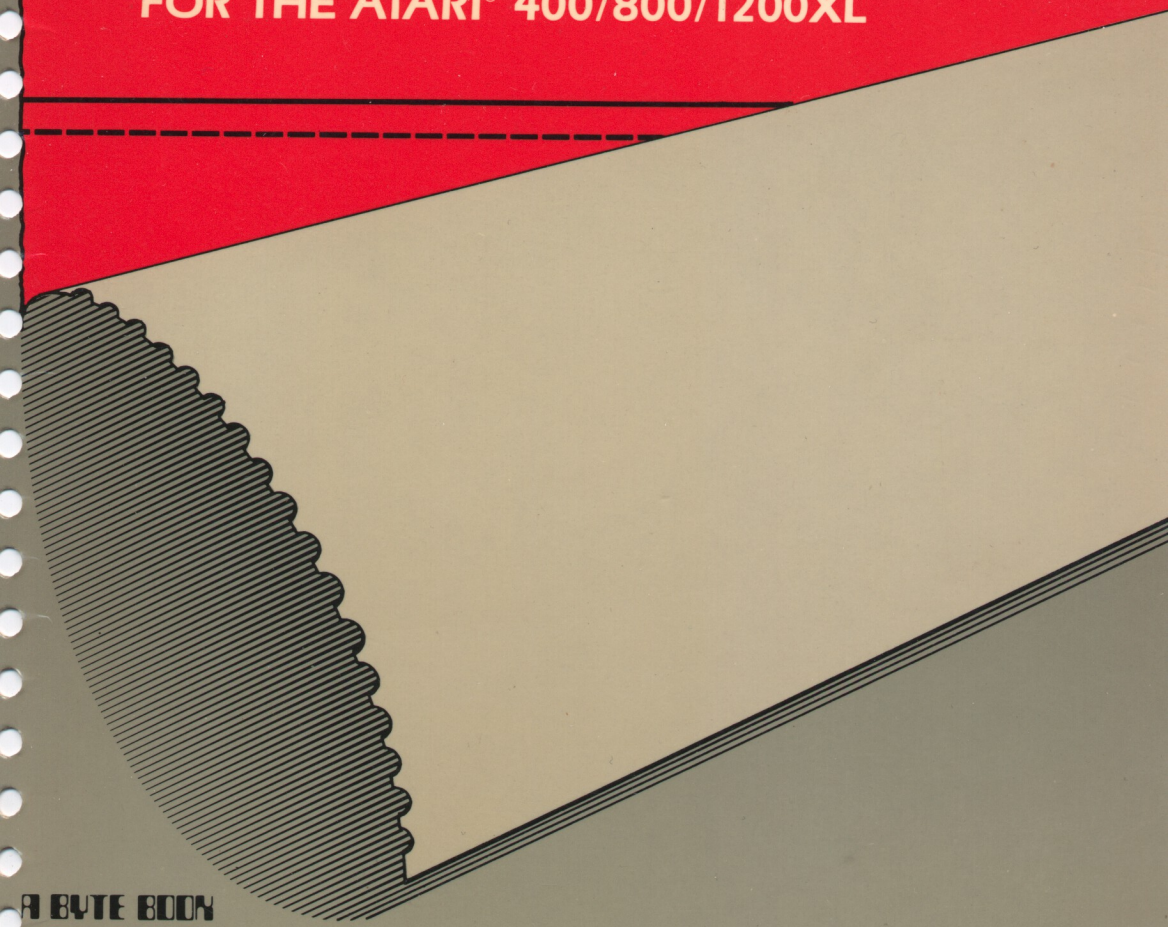
Hands-On BASIC

FOR THE ATARI® 400/800/1200XL



A BYTE BOOK

HERBERT PECKHAM
with WADE ELLIS, JR.
and ED LODI

HANDS-ON BASIC

FOR THE ATARI® 400/800/1200XL

HANDS-ON BASIC
FOR THE ATARI® 400/800/1200XL

HERBERT PECKHAM
with **WADE ELLIS, JR.**
and **ED LODI**

A Computer Literacy BOOK

McGRAW-HILL BOOK COMPANY
NEW YORK ST. LOUIS SAN FRANCISCO AUCKLAND
BOGOTÁ HAMBURG JOHANNESBURG LONDON MADRID
MEXICO MONTREAL NEW DELHI PANAMA PARIS
SÃO PAULO SINGAPORE SIDNEY TOKYO TORONTO

HANDS-ON BASIC: For the *ATARI*® 400/800/1200XL

1 2 3 4 5 6 7 8 9 0 H A L H A L 8 9 8 7 6 5 4 3

ISBN 0-07-049194-1

This book was set in Patina and Eras using the TEX composition system. The editor was Charles E. Stewart; the production supervisor was Joe Campanella; the book designer was Paul Quin; the cover designers were Oona Johnson and Paul Quin. Halliday Lithograph Corporation was printer and binder.

Library of Congress Cataloging in Publication Data

Peckham, Herbert D.
Hands-on BASIC for the Atari 400/800/1200XL.

(A Computer literacy book)
Includes index.
1. Atari 400 (Computer)—Programming. 2. Atari 800 (Computer)—Programming. 3. Atari 1200XL (Computer)—Programming. 4. Basic (Computer program language)
I. Ellis, Wade. II. Lodi, Ed. III. Title. IV. Series.
QA76.8.A8P43 1983 001.64'2 83-7973
ISBN 0-07-049194-1

CONTENTS

PREFACE

This book is a modification of *BASIC: A Hands-on Method*, which introduces students to BASIC on a number of different timesharing computers. The earlier book has been revised and modified to be used specifically on the personal computer manufactured by Atari, Inc. The thinking behind and justification for the original work remain unchanged and bear repeating.

Most BASIC programming texts have two serious drawbacks. First, almost all the texts presuppose a knowledge of mathematics that most of our intended readers do not have. Second, most texts require readers to spend little, if any, time on the computer. Typically, students try to study programming like any other subject and do not experiment with or execute programs on the computer. Our experience indicates that people understand text material better and more rapidly when it is preceded by a good deal of hands-on experimentation.

Most textbooks are used in classrooms and certainly many people learn programming in this traditional setting. However, personal computers will soon be in such widespread use that many people will learn programming outside the classroom. This text has been designed for anyone, in or out of the classroom, who wants to learn to program the ATARI 400, the ATARI 800, and the ATARI 1200XL computers.

This book is structured to make learning easy. Each chapter begins with a statement of objectives. Then discovery exercises let the student experiment with BASIC and see the language in action. Once students acquire a feel for BASIC, they can profitably proceed to a more traditional treatment of the concepts.

The text has twelve chapters and three appendices. Each chapter is a module of instruction that should require about one or two hours of computer work and perhaps one or two hours text

study. Reviews at the end of each chapter let students test their mastery of the objectives. The book can be used in different ways: as a self-study text, as the text for an open-entry, open-exit, self-paced course, and in tandem with a traditional lecture course.

People at any level from junior high through graduate school should be able to use this book to learn programming skills in BASIC rapidly and effectively. The student needs no knowledge of mathematics past introductory algebra, and the algebra used is mainly formula evaluation. Students with more advanced mathematical skills can apply them to independent work on the computer.

All students will need access to an ATARI 400/800/1200XL computer with a BASIC language cartridge, at least one disk drive, and a black-and-white or a color TV set. Although most of the work in the book can be done without a disk drive, the lack of a disk drive significantly limits the potential of the ATARI computer.

Two documents furnished with the ATARI 400/800/1200XL computer and three documents furnished with the ATARI disk drive are valuable to the student. The *Operators Manual* tells how to connect the parts of an ATARI BASIC system. The ATARI *BASIC Reference Manual* is a source of technical information. The *Disk Operating System II Reference Manual* describes the use of a disk drive with the ATARI 400/800/1200XL computer. Two other manuals concerning the disk drive, the *Owner's Guide* and *An Introduction to the Disk Operating System*, are also referenced.

Acknowledgments

I thank Wade Ellis, Jr. and Ed Lodi of the Computer Tutors of San Jose, California for their assistance in writing this adaptation of *BASIC: A Hands-on Method*. and for their typesetting and composition of the book using the TEX composition system. Additional thanks go to Atari, Inc. for their generous assistance.

Herbert D. Peckham

We would like to express our deep appreciation to several sources. The staff of the Context project at Stanford University were, as always, extremely helpful. Special thanks are due to Lincoln Ong who helped us immensely with transfering files, Dikran Karagueuzian for his efforts in getting the files to their final form, and Lisa Lodi for checking most of the book.

Finally, we are deeply grateful to our wives, Jane and Rose Marie, for their patience and understanding throughout the writing and production of this book.

Wade Ellis, Jr.
Ed Lodi

INTRODUCTION

Computers play a part in most of our daily activities. Without computers, businesses of all sizes, educational institutions, and various branches of government would be unable to handle the bewildering quantity of information that characterizes our society. Although the routine use of computers has become a significant part of everyday activities only in recent years, the trend will surely continue as the price of computers continues to drop. More and more people will need to know how to use computers if they are to participate fully in our society.

What Is BASIC?

You are about to embark on the study of the computer language called BASIC. BASIC is a very specialized language that permits communication between you and the computer. This language is not complicated and is certainly much easier to learn than Spanish or French. BASIC has a simple vocabulary of a few words, a grammatical structure, and rules of use just like any other language. Your main tasks will be to learn the vocabulary of BASIC, become familiar with its rules of grammar, and begin to see how the language lets you use the computer to do what you want. We have intentionally kept the level of mathematics in this book simple. Don't be too concerned if your mathematical skills are a bit rusty. As we proceed, you will have an opportunity to brush up on elementary mathematics.

One effective way to learn is to observe details and characteristics while performing a task. We will use this "discovery" strategy. You will begin each chapter with a session on the computer. After following the directions and observing the computer's response to your instructions, you will begin to acquire a feel for BASIC. Then you will study written material that summarizes what you have learned.

Where Did BASIC Originate?

The original version of BASIC was designed and written at Dartmouth College under the direction of Professors John G. Kemeny and Thomas E. Kurtz. In September 1963 they began to create a programming language written from the user's point of view. Much of the actual programming on the project was done by Dartmouth undergraduate students. The birthday of BASIC was May 1, 1964.

The success of this pioneering effort at Dartmouth attracted national attention, and other institutions became interested. The rest is history. What started as a project at a single college is now an established part of the computer industry. Today nearly every time-sharing computer supports some version of BASIC.

The enhanced versions have increased the power and capability of the language significantly. The most recent development is the adaptation of BASIC for use on small, inexpensive personal computers. ATARI BASIC for the ATARI 400/800/1200XL computer, the language presented in this book, is a powerful and flexible enhanced version of BASIC.

How To Use This Book

Each chapter begins with a brief statement of the objectives. Study these objectives carefully to get a clear picture of precisely where you're going.

The next section of each chapter is the discovery exercises. In that section you will record the computer output in the space provided, when appropiate, and try to answer any questions you are asked. These activities will lead you through the ideas involved and let you see BASIC working. Try to think about what will happen in the situations that are set up. Your relationship with the computer should be an active one. Whether your answers are correct is not important. The important thing is to think carefully about the questions and to try to answer them. The time you spend thinking about questions will save you time later on.

Following the discovery exercises is a complete discussion of the objectives. Since you will have already seen the ideas and concepts in action on the computer, your study of this material will be much easier and more profitable.

Typical programs are included in each chapter. These are discussed in detail to show how elements of programming are pulled together to produce a BASIC program.

Beginning with Chapter 5, we give a set of problems at the end of each chapter. You should work enough problems to satisfy yourself that you can write programs at the appropriate level. Solutions to the odd-numbered problems are in Appendix C.

Finally, each chapter has a practice test that lets you review your understanding of the material and discover needs for further study. The answers to the practice tests are contained in Appendix B.

CHAPTER 1

GETTING ACQUAINTED WITH YOUR ATARI 400/800/1200XL COMPUTER

Since the computer may seem a bit strange and complicated at first, we will proceed slowly. After a few sessions, routine operations will seem very natural and will cause you no trouble. Initially, though, be prepared for a certain "confusion quotient." Don't hesitate to review previously studied material if necessary.

1—1 OBJECTIVES

In this chapter you will become familiar with the computer and start learning how it operates. You will do no BASIC programming until the next chapter. Learning how to operate the keyboard and how to enter and modify information is easy, but it is fundamental to all that follows.

Connecting the Computer to Your TV Set and Turning it On and Off

Your ATARI 400/800/1200XL computer can be connected directly to a color or black-and-white TV set. To do this, you will need to connect the TV switch box that comes with the computer.

Using Direct Mode

One of the easiest ways to use the computer is in the direct mode. No programming is involved; the computer carries out instructions as they are entered. In due time you will learn how to do much more. For the present, however, simple operations in the direct mode are a good introduction to using the computer.

Learning Screen Editing

The information entered into a computer is rarley mistake free. You need to know how to edit— how to change or correct material that has been entered. A thorough knowledge of editing will save you a great deal of time later on.

1—2 DISCOVERY EXERCISES

Before beginning work on the computer, we must establish several important points. On a typewriter, the letter L is often used for the numeral 1. On the computer, however, the numeral 1 is with the other numeral keys along the top row of the keyboard. Similarly, the letter O is sometimes used for the numeral 0 on a typewriter, but on the computer the 0 is found on the top row of the keyboard.

■ **Don't use the L for the 1!**
Don't use the Oh for the 0!

The standard appearance of characters typed on ATARI computers is white on a black background. There is a key that causes characters typed to appear black on a white background instead. On the ATARI 1200XL, this key is next to the BREAK key at the top right hand side of the computer. On the ATARI 400/800 computers, it is the key with the symbol ⇞ next to the [SHIFT] key on the right hand side of the keyboard. Pressing this key causes any characters typed subsequently to appear in reverse video (background in white and characters in black). Avoid pressing this key until you become familar with using your ATARI computer. Reverse video can be turned off by pressing this same key again.

You will need to know the following locations. On the ATARI 1200XL computer, the SWITCH BOX receptacle is at the rear of the computer; the POWER IN receptacle is also at the rear; the POWER ON/OFF switch is at the rear, left hand side; and the CHAN switch is next to the POWER IN receptacle.

On the ATARI 400/800 computers, the POWER IN receptacle is on the right hand side of the computer; the POWER ON/OFF switch is also on the right hand side; and the CHAN switch is next to the POWER ON/OFF switch.

1. Now you are ready to begin work. Connect the TV switch box to your TV set and switch it to COMPUTER. Plug one end of the power cord into the POWER IN receptacle, and the other end into

a wall plug. Plug the cable found at the rear of the computer into the TV switch box (on the ATARI 1200XL, this cable must first be plugged into the SWITCH BOX receptacle).

2. Locate the RETURN key at the right hand side of the keyboard.

3. The BASIC language cartridge is installed by pressing it firmly into the appropriate slot. On the ATARI 1200XL, the slot is located on the left hand side of the computer. On the ATARI 400/800, the slot is located beneath the cartridge door found immediately above the keyboard. Be sure to close the door after inserting the BASIC language cartridge. Turn on the TV set and switch to channel 2. Now turn on the POWER ON/OFF switch. Be sure the CHAN switch is set to 2. After a few moments, you will see

```
READY
■
```

displayed on the screen. If not, reseat the BASIC cartridge, double check all the switch settings, and try again. The square ■ is called a cursor. It will be displayed on the screen most of the time.

4. Locate the keys on the right-hand side of the keyboard that have the following (as well as other) symbols on them: +, -, *, and /. Note that these are obtained without use of the SHIFT key.

5. Now type

```
PRINT 1+4
```

Has anything happened?

Now press RETURN and record what happened.

6. Now type

```
? 1+4
```

and press RETURN . What happened?

The question mark (?) is an alternate way to enter PRINT in ATARI BASIC.

7. Now you know how to make the computer do addition. Let's explore this further. Type

```
PRINT 20+54.7
```

and press RETURN . What happened?

8. Type

```
PRINT 2+4-3
```

and press RETURN . Record the output below.

9. Type

```
PRINT 12/2
```

and press RETURN . What happened?

What arithmetic operation does the / call for?

10. If you make a typing error, you can move the cursor, ■ , back to the error by holding down the CTRL (CONTROL on the ATARI 1200XL) key and pressing the key with the symbol ← on it. Each time you press ←, the cursor will move one place to the left. When you reach the error, make the correction, and return to where you left off typing. To return, hold down the CTRL key and press the key with the symbol → on it. Then continue typing. However, you need only press RETURN after making the correction if the remainder of the line is complete and correct. When you press the RETURN key, the computer may give the message ERROR. If this happens, try to see what the problem is and retype the line.

    Type the incorrect line

    ```
    PRINC 1+4
    ```

    and press RETURN . What happened?

    Now type

    ```
    PRINC 1+4
    ```

    and don't press RETURN . Instead, move the cursor to the G using CTRL and ← and type T. Press RETURN . What happened?

11. Your TV screen should be fairly full now. Press the CLEAR key, holding down the SHIFT key at the same time. What happened?

    Pressing CLEAR clears the screen. If the screen is full and new lines are entered, old lines will scroll off the top.

12. Type

    ```
    PRINT 2*50
    ```

    and press RETURN . What happened?

What arithmetic operation is called for by the *?

13. Type

```
PRINT (2+3)*4-1
```

but don't press RETURN . What do you think will happen when you press RETURN ?

Press RETURN and record what happened.

14. Type

```
PRINT "(2+3)*4-1"
```

and press RETURN . What happened?

15. What will happen if you type

```
PRINT "BAD DOG"
```

and press RETURN ?

Try it and see if you were correct.

16. Let's move on to a different topic. First, clear the screen. If you have forgotten how, look back at step 11. Type and press RETURN .

```
QUIZ = 95
```

Then type

```
PRINT QUIZ
```

and press RETURN . What happened?

17. Take a few moments to examine the lines below.

```
LENGTH=10
WIDTH=6
HEIGHT=4
VOLUME=LENGTH*WIDTH*HEIGHT
PRINT VOLUME
```

What do you think will happen if you type in these lines?

Now type in the lines remembering to press RETURN at the end of each line. What happened?

18. Study the lines below briefly.

```
LENGTH=12
WIDTH=9
SQYDS=(LENGTH*WIDTH)/9
PRINT SQYDS,"SQYDS"
```

What will happen if you type in these lines?

Clear the screen and type in the lines, remembering to press RETURN after each line. What happened?

19. This concludes the discovery material for this chapter. Turn off your computer and TV set.

1-3 DISCUSSION

Connecting The Computer to Your TV Set and Turning it On and Off

See the *Operators Manual* for instructions on connecting your ATARI computer to the TV set. Except for parts of Chapter 3 and a few examples, a color TV set is not required.

The computer is simplicity itself to turn ON and OFF! As you have already seen, this is done with the switch at the left side of the computer (right hand side on the ATARI 400/800). Be sure the BASIC language cartridge is inserted before turning on the computer. The

```
READY
■
```

that appears on the screen indicates that you are in ATARI BASIC. This procedure will be referred to as bringing up ATARI BASIC.

One important point: If things get away from you, if you lose touch, or if the computer seems out of control, you have a way to regain control. Simply press the RESET key (SYSTEM RESET key on the ATARI 400/800). This should put you back where you were before things seemed to go out of control. If that doesn't work, you can also recover by turning the computer off and on again. If you use this remedy, however, you will lose any programs or information in memory. If neither of these remedies works, the BASIC cartridge is probaby not seated properly in the slot. Reseat it and try again.

Using Direct Mode

In the discovery activities, you learned how to do simple arithmetic operations using the computer like a simple calculator. This is also known as the direct mode. As we shall see in the next chapter, you can use BASIC to store statements and commands in a series of numbered lines and then direct the computer to perform all the statements at the same time. If, however, you type in the statements without line numbers, the computer assumes you want a direct or immediate answer and does what you asked it to do, if possible.

When you type in material, nothing happens until you press RETURN . The RETURN key tells the computer you are through typing a piece of information. In a few cases, the computer responds to a single keystroke and you do not have to press RETURN . However, such cases are the exception rather than the rule.

You have discovered that addition and subtraction are called for by + and −, which probably wasn't much of a surprise! Multiplication and division are indicated by * and /, respectively. Parentheses can be used to group operations any way you wish. A number of other clever operations are possible, but we will postpone discussion of these to later chapters. If you type

```
PRINT 5*3.2+6.3
```

and press RETURN, the computer will carry out the arithmetic and print the result.

If you type

```
PRINT "ABCDEFG"
```

and press RETURN, the computer is instructed to print out the collection of characters between the quotation marks— in this case the letters ABCDEFG. Such a collection is called a "character string," an important concept that we will return to throughout the book.

The computer can keep track of a number of pieces of information in the direct mode. Thus, if you type

```
A=2
B=3
PRINT A+B
```

the computer will print 5 on the screen. There is a very important point in connection with this concept. If you type

```
PRINT TAX
```

and press RETURN, the computer will display 0. Since you gave no value to TAX, the computer assigned the value 0 and printed it out.

The computer is very relaxed about names for quantities used either in the direct mode or in BASIC programs. You can use long names like DEPTH or RATE as well as short names like D or R. However, using long names can create problems. The long names must be typed correctly each time they are used. Also, no spaces are allowed in names. Certain words cannot be used for variable names, because they are reserved for use by the computer. See Appendix A of the *ATARI BASIC Reference Manual* for a list of reserved words.

This very brief introduction to the notion of variable names suffices for our discussion of the direct mode. We will discuss the concept more completely later in the book.

Recall from the discovery exercises that you can use ? as an abbreviation for the PRINT statement. You can also use PR. as an abbreviation for the PRINT statement. (Note: The period in PR. is required.) Many of the BASIC statements you will learn have abbreviations, but for clarity, we will use the full name in this book. If you wish to abbreviate the names, you will find a list of most of the statements and their abbreviations on the inside of the back cover of the book. In addition, you will find a list of most of the error messages and their meanings, on the inside of the flap of the book cover.

Learning Screen Editing

The ATARI 400/800/1200XL computer has line editing commands you can use to modify BASIC programs. However, errors can also be corrected in the direct mode. We will limit our discussion to the two ways of making changes in a line before you press the RETURN key

First, you can move the cursor to the left with the DELETE BACK SPACE key (DELETE BACK S key on the ATARI 400/800 computer). As the cursor moves left, the character under the cursor is erased. When the error is reached, you can make the correction and resume typing from that point.

The second, and perhaps more useful method is to move the cursor to the left or right by holding down the CTRL key and then pressing ← or →. Characters do not get erased in this activity. You make corrections by typing over the errors. If you need to delete a character, you can hold down CTRL and move the cursor to the character you want to delete, then press DELETE BACK SPACE. You can insert characters by moving the cursor to the character in front of which you wish to insert a character. Then continue to hold down the CTRL key and press the INSERT key. The computer will provide a space for the character you want to insert. (Note: The SHIFT key is not used with INSERT when you use the CTRL key.)

When all the corrections have been made, you can press the RETURN key regardless where the cursor is at the time. If there are many errors in a line, you may wish to press RETURN and retype the line.

You can use these simple editing commands in the direct mode to make changes or corrections. We will consider Further editing features in the next chapter.

1—4 PRACTICE TEST

Take the test below to discover how well you have learned the objectives of Chapter 2. The answers to the practice test are given in the appendix.

1. How do you let the computer know that you are through typing a line?

2. If you lose control of the computer, how can you regain it?

3. What symbol indicates multiplication on the computer?

4. How do you clear the screen display?

5. What operation does the symbol / indicate?

6. What will happen if you type

```
PRINT 3*4/6
```

and then press RETURN ?

7. What will happen if you type

```
PRINT "25/5+2"
```

and then press RETURN ?

8. If you typed PRING 2+3*4 and noticed your spelling error before you pressed RETURN , how could you correct the error?

CHAPTER 2

INTRODUCTION TO BASIC

Now you are ready to begin learning about programming in BASIC. In this chapter you will learn to write and execute some simple programs.

2—1 OBJECTIVES

Correcting Mistakes

If you're like everyone else, you'll make mistakes when entering information into the computer. You will learn how to correct mistakes even if you don't notice them until after you've typed several other lines.

Learning the Requirements for BASIC Programs

All BASIC programs have characteristics in common. You will look at some very simple programs to learn about these characteristics.

Telling the Computer What to Do

Commands tell the computer to do something to or with a BASIC program. These action words control a program. You will look at the LIST, RUN, and NEW commands.

Entering and Controlling Programs

This objective overlaps the one above. The main thing is to become comfortable entering and controlling programs. All the programs you will encounter initially are short and easy to handle.

Using Variable Names in BASIC

You must know how to name either numbers or strings of characters in BASIC programs. Fortunately, the computer has very relaxed rules about this.

2–2 DISCOVERY EXERCISES

In the discovery activities that follow, you will enter various programs. If you see a <RETURN> in the instructions, press the RETURN key. Remember from Chapter 1 that pressing the RETURN key tells the computer that you are through typing. Now go on to the activities below.

1. Turn on your computer and TV display and bring up ATARI BASIC. (See step 3 of the discovery exercises in Chapter 1.)

2. Now type in

```
100 LET A=1 <RETURN>
```

   This is the first line of a BASIC program.

3. Now type in the balance of the program as listed below.

```
110 LET B=8 <RETURN>
120 LET C=A+B <RETURN>
130 PRINT C <RETURN>
140 END <RETURN>
```

   If you make mistakes typing the program, either retype the line or correct it using the methods you learned in Chapter 1.

4. Clear the screen by pressing the key marked CLEAR. Remember to use SHIFT. What happened to the program you just typed in?

5. Fortunately, all is not lost. The computer remembers what you typed in even though the screen is blank. Type LIST and press the RETURN key. What happened?

6. On the TV display you should see the program you just entered. For the time being, ignore the line numbers at the beginning of each line. Just read the lines in the program and try to get a sense of what they mean. If you tell the computer to carry out the instructions, what do you think will happen?

Type RUN and press the RETURN key. What did happen?

7. Now type

```
110 LET B=5 <RETURN>
```

Clear the screen, type LIST, and then press the RETURN key. What happened to line 110 in the program?

8. What do you think will happen if you tell the computer to execute this program?

Type RUN, press the RETURN key, and record what happened. Were you right?

9. Now type

```
140 <RETURN>
```

Clear the screen and display the program using the LIST command. What happened to line 140?

If you want to delete a line in a BASIC program, how do you do it?

10. Now run the program. What happened?

Does the computer need the END statement that used to be in line 140?

11. Experiment a bit more. Often you will want to clear a program from the computer's memory. This is done with the NEW command. Type NEW and press the RETURN key. What happened?

Type LIST and press the RETURN key to see what the computer has in memory. Is anything there?

12. You have learned to clear out a program from memory, but now you have no program left! To get the program back, you must enter it again. Type in the program below.

```
100 LET A=1 <RETURN>
110 LET B=8 <RETURN>
120 LET C=A+B <RETURN>
130 PRINT C <RETURN>
140 END <RETURN>
```

Check all the lines to make sure you entered them correctly. If you need to change a line, retype it. If you have to retype lines, clear the screen by pressing CLEAR and redisplay the program by typing LIST and pressing RETURN .

13. Now type

```
125 LET D=B-A <RETURN>
135 PRINT D <RETURN>
```

Clear the screen and display the program. What has happened?

14. Take a few moments to study the program. What will happen if you RUN the program?

Type run, press the RETURN key, and record below what the computer did.

15. In the original program, the lines were numbered in intervals of ten (e.g., 100, 110, 120, 130, and 140). Why are there "gaps" in line numbers? (Hint: See step 13.)

16. How do you insert lines in a BASIC program? (Hint: See steps 13 and 15.)

17. Clear the program in memory by typing NEW and pressing the RETURN key. Enter the program below.

```
100 INPUT WHITE <RETURN>
110 LET RED=WHITE+2 <RETURN>
120 PRINT RED <RETURN>
130 GOTO 100 <RETURN>
140 END <RETURN>
```

18. This new program has several features that you have not seen before. Study the program carefully and think about what will happen if you run the program. What does the GOTO 100 in line 130 mean?

19. Run the program and record what the computer did.

Type the numeral 6 and press the RETURN key. What happened?

Type the numeral 10 and press the RETURN key. What happened?

20. What line in the program do you think is generating the question mark?

Describe in your own words what the program is doing. If necessary, experiment some more to make sure you are correct.

21. Now get out of the program. Press the BREAK key located at the top right hand side of the computer (upper right hand side of the keyboard on the ATARI 400/800). What happened?

22. Clear out the program in memory (see step 11). Type in the following program.

```
100 LET A=100 <RETURN>
110 PRINT A <RETURN>
120 LET A=A+1 <RETURN>
130 GOTO 110 <RETURN>
140 END <RETURN>
```

23. Run the program and record what happened.

When you get tired of watching the display, press BREAK. What happened?

24. Try it once more. Run the program and interrupt it after a few numbers appear on the screen. How do you stop a BASIC program running on the computer?

Now type CONT and press RETURN . What happened?

Stop the program by pressing BREAK.

25. Clear the screen and display the program in memory. (See step 5.) Type the line below. Note the absence of spaces in the line.

```
100LETA=1 <RETURN>
```

Clear the screen and list the program. What happened?

Now type the line below. Note the extra spaces.

```
1 0 0 L E T A = 1 <RETURN>
```

What happened?

List the program. Do there seem to be more statements in the program now?

Clearly, some spaces are important in BASIC statements. Just note the fact for now. We will return to this matter later.

26. Let's try a program with some new features. Clear the program from memory by typing NEW and then pressing the RETURN key. Type in the program below.

```
100 PRINT "TYPE A NUMBER" <RETURN>
110 INPUT FIRST <RETURN>
120 PRINT "ONE MORE TIME" <RETURN>
130 INPUT SECOND <RETURN>
140 LET SUM=FIRST+SECOND <RETURN>
150 PRINT "THEIR SUM IS" <RETURN>
160 PRINT SUM <RETURN>
170 END <RETURN>
```

27. Study the program for a few moments. Now run the program. What happened?

Type the numeral 12, press the RETURN key, and record below what the computer did.

28. Now type the numeral 13, press the RETURN key, and record below what happened.

29. Now type in

```
LIST 120 <RETURN>
```

Move the cursor to the T in TIME by holding down CTRL and using the up arrow (on the same key with the minus sign) and the right arrow. Replace TIME" with NUMBER PLEASE". Run the program again.

30. This simple program illustrates that you can make BASIC programs print out curt or courteous messages as well as numbers.

31. Now let's look at a different topic. Clear the screen. Type NEW and press the RETURN key to clear the program from memory. Then enter the following program.

```
100 DIM A$(5)
110 LET A = 1 <RETURN>
120 LET A$ ="HOUSE" <RETURN>
130 PRINT A <RETURN>
140 PRINT "A" <RETURN>
150 PRINT A$   <RETURN>
160 PRINT "A$" <RETURN>
170 END <RETURN>
```

32. This program contains something new. Look at the A$ in line 120. Note that it is set equal to a word enclosed in quotation marks. The balance of the program has to do with variations on printing out A and A$. You will learn more about the DIM A$(5) statement in later chapters. Run the program and record the output.

33. Study the output carefully and identify what the computer printed in response to each of the PRINT statements. For the time being, just make the comparison. Later you will examine the subject in detail. Enter the following line:

```
165 PRINT B <RETURN>
```

34. Clear the screen and display the program with the LIST command. Note that B is mentioned only in line 165 in the PRINT statement. What do you think will happen if you run the program?

Now run the program and record what happened.

35. As you saw in Chapter 1, even though the value of B was not defined in the program, the computer assigned it a value of 0. This is an important fact to remember when you write programs.

36. This concludes the discovery activities for this chapter. Turn off the computer and go on to the next section.

2–3 DISCUSSION

Correcting Mistakes

Since most of us make mistakes while typing, we need to be able to correct errors. Suppose you make a mistake typing a line. How you corrected it depends upon whether you have pressed the RETURN key yet. If you are on the line containing the error, you can move the cursor left using both the CTRL key (CONTROL key on the ATARI 1200XL) and the left arrow key and make corrections as you learned in Chapter 1. When all the corrections are made in a line, press the RETURN key.

If you are not on the line that contains an error, you can either retype the line to correct it or use LIST and the line number. Then move to the error using the up arrow and left or right arrow keys. You can make changes by typing over characters and using the INSERT and DELETE keys, if necessary. Remember that you must hold down the CTRL key when performing any editing function. In addition, you must press the RETURN key while you are still on the line for the corrections to be made. You should practice correcting a line until you feel you have mastered correcting mistakes.

Learning the Requirements for BASIC Programs

You have inferred several important facts about BASIC programs. As a point of reference, we will use the program you used earlier.

```
100 LET A=1
110 LET B=8
120 LET C=A+B
130 PRINT C
140 END
```

Each BASIC program consists of a group of lines called statements. Each statement must have a line number. The program above has three BASIC statements: LET, PRINT, and END. The first two statements will be treated fully in the next chapter. For the time being, their function in this program is clear. The END statement is not required with the ATARI computer. The computer will execute

the program with or without the END statement. However, it is a good practice to use it, especially in long programs, so there will be no confusion about where the program ends. Make it a rule to use the END statement in all programs even though the computer doesn't require it.

Generally the line numbers in a BASIC program are not consecutive (e.g., 100, 101, 102, etc.) because you may want to insert additional statements later if you discover errors or want to modify the program. If there were no intervals between lines, you may have to retype the entire program to make a change. "Gaps" in the line numbers allow you to insert statements simply by typing in the new statements with line numbers not already in the program.

As you saw in the discovery exercises, spaces are generally important in BASIC program statements. The computer will let you know whenever spaces are used incorrectly. If you write readable statements, you should have no problems with spaces.

In BASIC the order in which you enter the lines makes no difference. If you type

```
140 END
120 LET C=A+B
110 LET B=8
130 PRINT C
100 LET A=1
```

and display this new program, the computer will sort out the statements and display them in numerical order. In the same way, if you were to execute the program as typed above, the computer would sort the statements into the proper order before it executed them.

You can remove a BASIC statement from the program by typing the line number and pressing the RETURN key. You can modify statements by retyping lines or editing them as we have shown. You can add new statements by using line numbers not already in the program. Thus, you can add, remove, or change BASIC statements as you wish. This ability to change programs easily is a desirable characteristic of BASIC on the ATARI computer.

Telling the Computer What to Do

There is an important distinction between statements (lines in a BASIC program) and commands. Commands tell the computer to do something with a program. You have seen several of these in the computer work, and we will briefly review the use of each.

When you enter a BASIC program, it goes into memory. Quite often you need to see the program contained in memory, perhaps because you want to see changes made in the program, or perhaps because you simply need a copy of the program. In any case, you use the LIST command. The wise programmer makes valuable use of this command. If a program doesn't work as it should, your first step should be to display the program in memory. You may have the computer furnish a copy of the program stored in memory. Use this copy to troubleshoot the program.

When you turn off the computer, the contents of memory are cleared out automatically. When the computer is on, however, it is possible to mix up programs. Suppose you are working on one program and decide to go to another. If you don't clear the first program out of memory, the second program will enter over the first, and as a result parts of both programs may be in memory. The way to avoid this is to clear out (or erase) a program with the NEW command when you are through with it.

A BASIC program is simply a set of instructions on which the computer acts. However, the computer needs to be told to start this process. When the computer receives the RUN command, it goes to the program statement with the lowest line number and carries out the instructions. The computer then goes to the statement with the next-to-lowest number and keeps on carrying out instructions in numerical order, unless the program directs that a statement be done out of order.

Entering and Controlling Programs

So far, when you have typed programs or commands, the <RETURN> prompt appeared to remind you to press the RETURN key. Pressing RETURN should be a habit by now, so we will not use the <RETURN> prompt in the future. From now on, remember to press the RETURN key to let the computer know you are finished typing a statement or command.

Sometimes you need to control a program that is running. Certainly you would want to interrupt a program in a closed loop because otherwise the program will run forever. You can break into such a program by pressing the BREAK key. (Sometimes you must press BREAK twice.) The computer then interrupts its execution of the program and displays STOPPED AT LINE (whatever line was being processed when it was interrupted). The

computer resumes program execution when you type CONT and press RETURN .

You would also interrupt when the computer is in an input loop waiting for a number to be typed in. Again, press BREAK. The computer will jump out of program execution and displays the cursor ■.

Using Variable Names in BASIC

A common confusion for the beginner in BASIC is the distinction between the name of the variable and the data stored under that name. In the BASIC statement

```
100 LET A=2
```

the A names a variable. Variable means that different values can be assigned to A. Consequently, LET statements are often called assignment statements. In this case the variable A is assigned the value 2. What actually happens is that somewhere in the computer memory there is a location named A, and the computer stores the number 2 in that location, The fundamental distinction is between the name of a location in memory and the contents of the memory locations, much like the distinction between a post office box number and the contents of that box. The box number does not change, but the contents of the box may change at any time.

In ATARI BASIC, the use of LET in the assignment statement is optional. In this book, we will always use LET in assignment statements for the sake of consistency.

Consider the following statement.

```
130 LET C=A+B
```

This statement instructs the computer to get the numbers stored in locations named A and B, add them together, and put the sum in the storage location named C. The equal sign tells the computer to evaluate what is on the right and assign it to the variable name on the left. Now consider the BASIC statement

```
120 LET B=B+1
```

If we consider the statement above as an algebraic equation, we have

$$B = B + 1$$

If we subtract B from both sides of this equation, we have

$$0 = 1$$

which is very strange indeed! It is clear that in a BASIC statement the = sign does not mean what it does in an algebraic equation. Instead, the statement

```
120 LET B=B+1
```

instructs the computer to get the number stored in location B, add 1 to it, and put the result back in the storage location named B.

If you store a number in a location, anything that was stored there before is lost. Consider the following statements:

```
100 LET A=1
110 LET A=2*3
```

Line 100 instructs the computer to set up a location called A and put the number 1 in that location. Line 110 tells the computer to multiply 2 by 3 and store the product in location A. The 1 stored previously in location A is then lost.

To be precise, the variable A above is a "numeric" variable. The reason for including numeric in the name is that there is another type of variable called a "character string." You were introduced to this concept briefly in the discovery activities.

It is easy to distinguish between numeric and character-string variables. A, BOOK, M, and P would all identify numeric variables and name numeric quantities. A$, BOOK$, M$, and P$ all name strings of characters. The $ symbol identifies the name as a character-string variable. In the BASIC statement

```
100 LET B$="BARN"
```

B$ names a location in memory in which the character string "BARN" is stored. The quotation marks set off the string, but are not part of it.

Recall that the ATARI computer has very relaxed rules for variable names. It allows you to use "long" names for numeric variables as well as character strings. You can use up to 110 characters (including the $ character in the case of character strings) in long names. The computer has a set of words that are reserved for commands and statements and cannot be used to name variables. See

the *ATARI BASIC Reference Manual* for the list of reserved words. If you make a mistake and use one, however, the computer will often let you know.

Let's go over the important points once more. A variable name in BASIC identifies a storage location in memory that can contain a number or character string. The contents of the storage location (the value of the variable) may be changed, but the name of the location cannot.

The LET (or assignment) statement evaluates what is on the right side of the equal sign and assigns it to the storage location named on the left side. Thus,

```
100 LET D = A+B+C
```

instructs the computer to evaluate the expression (A+B+C) using the numbers stored in memory locations named A, B, and C. The computer stores the result in the memory location named D. We will return to the topic of character-string variables several times in this book.

2—4 PRACTICE TEST

1. How do you signal the computer that you are through typing a line or instruction?

2. Suppose that your computer is waiting for you to enter a number in an INPUT statement of a program. You decide that you want to jump the computer out of the program instead. How do you do this?

3. How do you interrupt a program that is running on your computer?

4. What is wrong with the following program?

```
100 LET A=1
110 LET B=3
120 LET C=B-A
PRINT C
130 END
```

5. What will happen if the program in question 4 were corrected and run?

6. How long can variable names be?

7. How do you insert a line in a BASIC program?

8. How do you replace a line in a BASIC program?

9. How do you remove a line from a BASIC program?

10. How do you display the program in memory?

11. How do you erase the screen?

12. How do you command the computer to execute the program in memory?

13. How do you erase a program in memory?

14. What is the difference between a numeric and a character-string variable?

CHAPTER 3

GRAPHICS

3—1 OBJECTIVES

In this chapter we will discuss the graphics statements that are available in the ATARI BASIC language.

Using Text and Graphics Modes

ATARI BASIC has three text modes and six graphics modes. You will learn how to enter graphics modes to begin creating drawings with your computer.

Drawing Lines

Straight lines are the basic component of many kinds of graphics. You will learn to draw lines easily and quickly as a first step in learning to make complex drawings on the display screen.

Creating and Positioning Shapes

You will learn to use your ability to draw lines to create shapes and place them where you wish on the screen.

Using Animation

Moving figures on the screen make computing more interesting and greatly increase the effectiveness of graphics displays. You will learn how to make figures move about the screen.

Using Color

Colors make graphics more attractive. You will learn to draw with colors and change background and border colors.

3-2 DISCOVERY EXERCISES

1. Turn on the computer and bring up ATARI BASIC. Remember to press RETURN after you have typed a line. Type

```
GRAPHICS 8
```

What happened?

2. Now type

```
PLOT 160,80
```

What happened?

3. Type

```
COLOR 1
PLOT 160,80
```

What happened this time? (Look at the center of the screen.)

4. Now type

```
PLOT 310,5
PLOT 310,159
PLOT 5,159
```

Where do the points appear on the screen? (Look at the corners of your screen. If your TV set is improperly aligned, some or all of these points may not appear.)

5. What should you type in to display or plot a point in the upper left-hand corner of the screen?

Try it and see if you were right.

6. Now type

```
GRAPHICS 8
```

Are the points still there?

7. Type

```
10 PRINT "HELLO"
20 PRINT "SAILOR"
30 END
```

and list the program. What happened?

8. Now type

```
GRAPHICS 0
```

and again list the program. What happened this time?

GRAPHICS 0 puts you back into standard text mode.

9. Type

```
GRAPHICS 8
PLOT 5,5
DRAWTO 310,5
```

What happened?

10. Type

```
DRAWTO 310,159
DRAWTO 5,5
```

Draw the figure that appears on the screen.

Are all the lines solid?

Now type

```
DRAWTO 320,5
```

What happened?

Error 141 means the point (in this case 320,5) is off the screen. The first number in the DRAWTO statement cannot be larger than 319. Actually you may not be able to see the allowable corner points such as 0,0 and 319,0 if your set is not adjusted very well.

11. Clear the graphics screen by typing

```
GRAPHICS 8
```

12. What statements would you enter to draw a box around the entire graphics region? (See steps 3 and 4.)

Enter the statements to see if you were correct.

13. Let's draw a series of lines. Clear the graphics screen with the GRAPHICS 8 statement. Type

```
PLOT 5,5
DRAWTO 50,5
DRAWTO 50,50
```

Draw the shape that appears on the screen.

14. Type

```
PLOT 5,5
DRAWTO 5,100
DRAWTO 100,100
```

What happened?

15. Clear the graphics screen. (See step 11.) Type

```
PLOT 10,10
DRAWTO 50,50
```

What happened?

Now type

```
PLOT 10+100,10+100
DRAWTO 50+100,50+100
```

Are the lines the same length?

Compare the two sets of statements that draw these lines.

16. Type

```
DRAWTO 200,150
```

Where did this line start?

17. Where will the line start that is drawn by

```
DRAWTO 10,10
```

Try it and see if you were right.

18. Clear the graphics screen. Type

```
COLOR 0
PLOT 0,0
DRAWTO 50,50
```

Was anything drawn on the screen?

19. Type

```
COLOR 1
PLOT 0,0
DRAWTO 50,50
```

What happened?

20. Type

```
COLOR 0
PLOT 0,0:DRAWTO 50,50
```

What color was used to draw this line?

COLOR 1 sets the drawing color to white. COLOR 0 sets the drawing color to the background color. Drawing a line with background color over a white line has the effect of erasing the white line.

Note that you can put more than one statement on a line if you separate them with a colon.

21. Clear the graphics screen and type

```
COLOR 1:PLOT 10,10
DRAWTO 10,30:DRAWTO 30,30
DRAWTO 30,10:DRAWTO 10,10
```

What was drawn?

How long is the edge of the figure?

Where did the figure start?

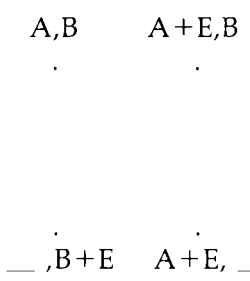
22. Now let's use programs to draw shapes. Type

```
GRAPHICS 0
```

and then type the following program.

```
10 GRAPHICS 8
20 LET A=0
30 LET B=0
40 PRINT "EDGE LENGTH ";
50 INPUT E
60 COLOR 1
70 PLOT A,B:DRAWTO A+E,B
80 DRAWTO A+E,B+E
90 DRAWTO A,B+E:DRAWTO A,B
100 GOTO 20
110 END
```

Study the program. Use the points in the PLOT and DRAWTO statements in lines 70, 80, and 90 to fill in the blanks in the following diagram. Connect the points with lines as indicated in program lines 70, 80, and 90.



Run the program. Input 20, 40, 70, 100 and 150 at the question mark prompts. What is the starting point for each figure?

Jump the program out of the input loop (BREAK).

23. Enter text mode by typing

```
GRAPHICS 0
```

Change lines 20 and 30 as follows.

```
20 PRINT "STARTING POINT ";
30 INPUT A,B
```

Display the program. Run the program. Enter 20,70 for the starting point and 70 for the edge. What happened?

Now enter 100, 150 for the starting point and 70 for the edge. What happened?

Avoid illegal quantities in DRAWTO statements if you wish the program to continue executing. When you plot a point, the largest first number you can use is 319, and the largest second number is 191.

24. Now that you are able to draw shapes by using the DRAWTO statement, let's make the box move on the screen. Enter text mode (see step 23) and display the program. Add lines 55, 56 and change line 100 as follows:

```
55 LET A=A+1
56 LET B=B+1
100 GOTO 55
```

Display the program. Check it against the complete program below.

```
10 GRAPHICS 8
20 PRINT "STARTING POINT ";
30 INPUT A,B
40 PRINT "EDGE LENGTH ";
50 INPUT E
55 LET A=A+1
```

```
56 LET B=B+1
60 COLOR 1
70 PLOT A,B:DRAWTO A+E,B
80 DRAWTO A+E,B+E
90 DRAWTO A,B+E:DRAWTO A,B
100 GOTO 55
110 END
```

Run the program. Enter a starting point of 0,0 and an edge length of 20. What happened?

Are you in text mode now?

Enter text mode (see step 23) and change lines 55 and 56 as follows.

```
55 LET A=A+3
56 LET B=B+2
```

Run the program entering 0,0 and 20 again at the input prompts. Does the figure move faster?

25. Again, enter text mode and add the following lines to erase the box right after it is drawn.

```
91 COLOR 0:PLOT A,B
92 DRAWTO A+E,B:DRAWTO A+E,B+E
93 DRAWTO A,B+E:DRAWTO A,B
```

COLOR 0 causes the DRAWTO statements in lines 92 and 93 to be drawn in the background color. Run the program entering 0,0 for the starting point and 20 for the edge. What happened?

The box appears to move down the screen. In direct mode, type

```
PRINT A+E,B+E
```

Which of these numbers was the illegal quantity?

---

26. If you are not using a color television set, go to step 31. Let's investigate using color to draw lines. Enter text mode and clear the memory and the screen. If you wish to check the color settings on your display, type and run the following direct mode statements.

```
GRAPHICS 3
COLOR 1
PLOT 0,0
DRAWTO 20,0
COLOR 2
PLOT 0,2
DRAWTO 20,2
COLOR 3
PLOT 0,4
DRAWTO 20,4
```

GRAPHICS 3 puts the screen into low resolution graphics, in which there are four colors available for drawing lines. These statements will display three horizontal bars whose colors are orange, light green, and dark blue, respectively.

Now type in the following direct mode statements, noting the display as you do.

```
COLOR 0
PLOT 0,2
DRAWTO 20,2
```

COLOR 0 is black, the same as the backround.

27. Let's look again at high resolution graphics and color. Look at the screen after typing each of the following direct mode statements:

```
GRAPHICS 8
SETCOLOR 2,2,4
SETCOLOR 2,4,4
SETCOLOR 2,6,4
SETCOLOR 2,8,4
```

Did the background colors change?

28. Now type in

```
SETCOLOR 2,2,4
SETCOLOR 2,2,6
SETCOLOR 2,2,8
```

Did the background color get brighter?

29. Now type in

```
SETCOLOR 4,4,4
```

What happened to the border color?

30. Type in

```
SETCOLOR 4,6,4
SETCOLOR 4,8,4
```

Did the border color change?

31. Chapters 7 and 8 will present program flow controls that will help you avoid illegal quantity errors. For now, we have completed the discovery exercises. Turn off the computer.

3—3 DISCUSSION

Using Text and Graphics Modes

In the discovery exercises you saw two screen modes, text mode and graphics mode. Actually, your ATARI computer has three text modes and six graphics modes. To choose a particular mode you must know of the attributes of each mode and what task you wish to perform. (See Chapter 9 of the *ATARI BASIC Reference Manual* for more details.) We have used the standard text mode, GRAPHICS 0, in this chapter as well as in the previous ones. You enter the standard text mode every time you turn on your computer and bring up ATARI BASIC. It provides a screen of 24 lines of text and 40 columns.

To enter the high resolution graphics mode, use the GRAPHICS 8 statement. The high resolution graphics mode provides a display of 320 points across and 160 points down. There is also a narrow band at the bottom of the screen for 4 text lines. This combination is called a split screen. You can obtain a full graphics screen (320 points by 192 points by using the statement GRAPHICS 8+16 instead of GRAPHICS 8. Adding +16 to any of the graphics statements will give you a full screen rather than a split screen. A full screen allows you to plot more points at the bottom of the screen.

To enter the low resolution graphics mode, use the GRAPHICS 3 statement. This will provide 40 points by 20 points with a split screen.

- **Use GRAPHICS 8 to enter high resolution graphics mode.**
  Use GRAPHICS 3 to enter low resolution graphics mode.

Drawing Points and Lines

The PLOT statement is used to draw points. For example,

```
PLOT 0,0
```

plots a point in the upper left-hand corner of the screen in the current

color. The point may not be visible if your set is not properly aligned. The statements

```
PLOT 0,0
DRAWTO 319,159
```

plot a point at 0,0 and then draw a diagonal line across the screen in the current color. The statements

```
PLOT 160,80
DRAWTO 160,159
```

plot a point at 160,80 (the center of the screen) and then draw a line down to the point 160,159. The graphics pen is positioned at the last point it draws.

- **Use PLOT to draw points, DRAWTO to draw lines.**

Creating and Positioning Shapes

As you saw in the discovery exercises, creating shapes is a simple matter of drawing lines to produce the desired shape. For example,

```
GRAPHICS 8
COLOR 1
PLOT 0,40: DRAWTO 10,0: DRAWTO 20,40
DRAWTO 10,40: PLOT 10,40: DRAWTO 10,50
```

draws a pine tree when entered in direct mode.

To position this pine tree near the point 160,80, you would use the following statements:

```
GRAPHICS 8
COLOR 1
PLOT 160+0,80+40: DRAWTO 160+10,80+0
DRAWTO 160+20,80+40: DRAWTO 160+0,80+40
PLOT 160+10,80+40: DRAWTO 160+10,80+50
```

You can select a starting point A,B while the program is running by using an INPUT statement in the program. You can position the figure anywhere you desire on the screen. Take care that no point

has a first number larger than 319 or a second number larger than 191.

As you saw in the discovery exercises as well as in these last two examples, you can put multiple statements on a single line if separate them with the colon.

Using Animation

You can make a shape appear to move about the screen by first drawing it at a particular position in white (or color) and then redrawing it in the background color in the same position. Then repeat the first two steps using a slightly different starting position. If you continue to move the starting positions, the shape will appear to move along the path of the starting positions. For example, the program

```
100 GRAPHICS 8
110 LET A=0
120 COLOR 1
130 PLOT A+20,40: DRAWTO A+23,25
140 DRAWTO A+40,40: DRAWTO A+20,40
150 PLOT A+30,40: DRAWTO A+30,45
160 PLOT A+10,45: DRAWTO A+50,45
170 DRAWTO A+45,50: DRAWTO A+15,50
180 DRAWTO A+10,45
190 COLOR 0
200 PLOT A+20,40: DRAWTO A+23,25
210 DRAWTO A+40,40: DRAWTO A+20,40
220 PLOT A+30,40: DRAWTO A+30,45
230 PLOT A+10,45: DRAWTO A+50,45
240 DRAWTO A+45,50: DRAWTO A+15,50
250 DRAWTO A+10,45
260 LET A=A+15
270 GOTO 120
280 END
```

will cause a sailboat to move across the top of the screen. You might try this program on your computer. Notice that the value of the variable A (which is changed in line 260) sets the starting point. As the starting point changes, the position of the figure also changes.

Using Color

In high resolution graphics, lines can be drawn in one of two colors. COLOR 1 causes a line to be drawn in what might be called white (changing the background color causes variations in the color of the line). COLOR 0 causes a line to be drawn in the background color.

In low resolution graphics, there are four colors available for drawing lines. COLOR 1 draws lines in orange, COLOR 2 draws lines in light green, COLOR 3 draws lines in dark blue, and COLOR 0 draws lines in the background color. Again, changes in background color will change the color of the lines.

As you saw in the discovery exercises, the SETCOLOR statement can change the color and brilliance of the background and border colors. For example, the statement SETCOLOR 2,4,6 will make the background pink. The first number, 2, refers to the background. The second number, 4, indicates what color the background will be, and the third number, 6, gives the brilliance or luminance of the hue. The higher the third number, the greater the luminance. This number can be any even number from 0 to 14. Some of the possible color hues and their corresponding numbers are listed in the table below. See Table 9.3 of the *ATARI BASIC Reference Manual* for the complete set of colors available.

Number	Color
0	Gray
2	Orange
4	Pink
7	Blue
10	Turquoise
12	Green

The border colors are determined in the same way as the background colors. To change border colors, use 4 as the first number in the SETCOLOR statement. The statement SETCOLOR 4,12,6 would give a green border.

3–4 PRACTICE TEST

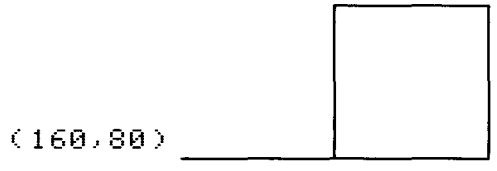
1. What figure will the computer draw when you enter the following direct mode statements?

```
GRAPHICS 8:COLOR 1
PLOT 160,60:DRAWTO 140,60
DRAWTO 140,100:DRAWTO 180,100
DRAWTO 180,60:DRAWTO 160,60
```

2. What figure will the computer draw when you enter the following direct mode statements?

```
GRAPHICS 8:COLOR 1
PLOT 160,80:DRAWTO 140,100
DRAWTO 180,100:DRAWTO 160,80
PLOT 140,100:DRAWTO 140,140
DRAWTO 180,140:DRAWTO 180,100
```

3. Write direct mode statements that will instruct the computer to draw the following diagram, with the point (160,80) as indicated.



4. Write direct mode statements that will instruct the computer to plot the following points in high resolution graphics.

   a. The center of the screen.

b. The upper left-hand corner.

c. The top center point.

d. The point (240,150).

5. a. How do you get into high resolution graphics mode?

b. What color statement should you use if you want to draw an orange line in low resolution graphics?

6. What does the following program do?

```
100 GRAPHICS 8
110 LET A=0
120 COLOR 1
130 PLOT 140,A: DRAWTO 200,A
140 COLOR 0
150 PLOT 140,A: DRAWTO 200,A
160 LET A=A+5
170 GOTO 120
180 END
```

CHAPTER 4

COMPUTER ARITHMETIC AND PROGRAM MANAGEMENT

4—1 OBJECTIVES

Now that you have learned how to turn on your ATARI computer, how to bring up ATARI BASIC, and how to communicate with the computer, you are ready to go on to more interesting tasks. You will need an ATARI disk drive and a blank diskette for some of these activities.

Doing Arithmetic on the Computer

Mathematics on the computer calls for only the simplest arithmetic operations. You will gain a clear understanding of how these arithmetic operations are done.

Using Parentheses in Computations

You must type all mathematical expressions on a single line to enter them into the computer. Some expressions can be handled this way only by organizing parts of the expressions in parentheses. You will learn to use parentheses effectively.

Using E Notation for Numbers

The computer must deal with both very large and very small numbers. E notation is used by the computer to describe such numbers. You will learn to recognize and interpret E notation.

Formatting a Diskette

You must prepare or "format" new or blank diskettes for use on your ATARI computer. You will learn how to format diskettes correctly in order to store programs.

Storing and Retrieving Programs

You have already seen some commands. You will learn additional commands that will permit you to store programs on and retrieve them from a diskette.

4-2 DISCOVERY EXERCISES

1. Turn on your computer and bring up ATARI BASIC. Recall from Chapter 1 the following symbols for arithmetic operations:

   + Addition
   -Subtraction
   * Multiplication
   / Division

   To review the arithmetic operations, type in the following program:

```
100 INPUT A
110 INPUT B
120 LET C=A*B-B/3
130 PRINT C
140 END
```

   Display the program and study it briefly. If we run the program now and enter 2 for **A** and 3 for **B**, what do you think will be typed out?

   _____

   Run the program and write down what happened.

   _____

2. Clear the program in memory. Note that the symbol ^ is on the same key with the * and requires you to use the SHIFT key. Now type

```
100 LET A=3*3
110 LET B=3^2
120 PRINT A
130 PRINT B
140 END
```

Display the program and make sure it is correct. Now run the program and record what was typed out.

Compare the numbers printed out to the expressions in the lines where they were computed. Notice that they are essentially the same. See if you can figure out what is taking place.

3. Type

```
100 LET A=3*3*3
110 LET B=3^3
```

Run the program. What was typed out?

4. Type

```
100 LET A=2*2*2*2
110 LET B=2^4
```

Run the program. What was typed out?

What is the ^ symbol used for?

5. Remember from your introductory algebra course (if you haven't had algebra, don't panic!) that when you want to multiply 2x2x2 for example, you can indicate this with an exponent. You would write the expression as

$$2^3$$

How would this expression be written in BASIC using the ^ symbol? (Hint: See steps 2, 3, and 4.)

6. Fill in the operators (symbols) that call for the following arithmetic operations:

Division

Exponentiation

Multiplication

7. Clear out the program in memory. Type

```
100 LET A=4+2*6/3
110 LET B=(4+2)*6/3
120 LET C=4+(2*6)/3
130 LET D=4+2*(6/3)
140 PRINT A
150 PRINT B
160 PRINT C
170 PRINT D
180 END
```

The two points of this program are (1) the order in which the arithmetic is done, and (2) the effect of the parentheses. If you look closely, you will see that the same numbers and operations are involved in each of the calculations in lines 100, 110, 120, and 130.

The only difference is the groupings within parentheses. Run the program and record what was typed out.

Study the program and the numbers the computer typed out until you see what is taking place in the program. The computer has established rules for such situations. We will go over these rules later in the chapter.

8. Clear out the program in memory. Now enter the following program:

```
100 LET A=3*100*100*100
110 LET B=3*100*100*100*100
120 LET C=3*100*100*100*100*100
130 PRINT A
140 PRINT B
150 PRINT C
160 END
```

Execute the program and record the output.

Can you explain the different form in which the numbers were typed out? (Hint: Count the number of zeros in the multipliers in lines 100, 110, and 120.)

9. Type

```
100 LET A=4/10
110 LET B=4/(10*10)
120 LET C=4/(10*10*10)
```

Execute the program and record the output.

Can you understand what is happening in the output? Count the zeros in the denominators in lines 100, 110, and 120.

10. Explain in your own words what it means when an E shows up in a number typed out by the computer.

If you still do not fully understand the purpose of the E notation, don't worry. We will return to it later.

11. Before you can proceed with storing and retrieving programs, you must format a diskette. (If you do not have a disk drive, go to the discussion.) Turn off your computer and connect the disk drive (see steps 1-6 in the first few pages of the *Owner's Guide* that came with the disk drive). When you are asked to insert a diskette at step 5 in the *Owner's Guide*, insert the diskette labeled Master Diskette II that came in the package.

12. Close the disk drive door. If you removed the BASIC cartridge, replace it and turn on the computer to bring up ATARI BASIC. Note: Wait until the the busy light goes off before proceeding.

13. Now type

```
DOS
```

and in a short time you will see a "menu" on the screen. This menu gives you several options.

14. You are now ready to format a diskette and place the DOS files on it. First replace the diskette in the drive with the diskette to be formatted. This can be either a new diskette or an old one. Warning: all programs and information on this diskette **will be lost** when the diskette is formatted.

15. We note from the menu that I is the letter you need to select. So type I and press RETURN . Type in 1 and Y, respectively, in response to the next two questions. After about 45 seconds, the diskette will be formatted, and the display will again show

```
SELECT ITEM OR RETURN FOR MENU
```

16. The next menu item you want is H. Type in H and press RETURN . Again, type in a 1 and Y in response to the next two questions. The computer will display

```
WRITING NEW DOS FILES
```

17. To see what files are stored on the diskette just formatted, type in A and press RETURN twice. You should now see

```
  DOS      SYS 039
  DUP      SYS 042
626 FREE SECTORS
```

18. We are now ready to store and retrieve programs. Get back to ATARI BASIC by typing B and RETURN . Then type in the following program:

```
100 LET A=2
110 LET B=3
120 LET C=A*B
130 PRINT C
140 END
```

19. Display the program to make sure it is correct. Run the program.

20. Now type

```
SAVE "D:PRODUCT"
```

This moves the program in memory to diskette under the name PRODUCT. You may choose any name of eight characters or less beginning with a letter.

21. You can obtain the directory of files on this diskette by first typing DOS and pressing RETURN . When the busy light on the disk drive goes off, type in A and press RETURN twice. You should now see that the program PRODUCT has been added to the directory. Is it there?

22. Get back to ATARI BASIC by typing B and RETURN . Now display the memory. Is anything there?

23. Type in the following program.

```
100 LET D=2*6-8/4
110 PRINT D
120 END
```

Run the program. Think of a name for this program other than PRODUCT. Limit the name to eight or fewer characters The first character must be a letter, but the others can be digits or letters. Record the name below.

Move the program in memory to diskette under the name you have just chosen. (See step 20.) Now display the program in memory. Is the program you just saved to diskette still in memory?

24. Obtain the directory of programs stored on diskette (See step 21.) Are the two programs you just entered there?

Now, you should have stored the two programs you just entered. The name of the first program is PRODUCT; you wrote the name of the second in step 22. To simplify the discussion, we will refer to this second program as **PROGRAM 22**. You, of course, must use the name you selected for the second program.

25. Get back into ATARI BASIC with the correct menu selection. To move PRODUCT from disk to memory, type

```
LOAD "D:PRODUCT"
```

Display this program and verify that it is the right one. Now move **PROGRAM 22** from disk to memory. Display the program in memory. Which one is there now?

What happened to the program PRODUCT that was in memory when you moved **PROGRAM 22** into memory from diskette?

Now type

```
RUN "D:PRODUCT"
```

What number is displayed on the screen?

What program is now in memory?

Display the program to see if you were correct.

26. We now have PRODUCT in memory. Remove the program PRODUCT from diskette by typing DOS. Then select D from the menu and in response to

```
DELETE FILE SPEC
```

type in PRODUCT. Then type in a Y in response to

```
TYPE "Y" TO DELETE
D1:PRODUCT    ?
```

27. Obtain a directory of the programs stored on diskette. PRODUCT should not be there, but **PROGRAM 22** should be. Inspect the listing of programs. Is everything the way it should be?

28. Get back to ATARI BASIC. Now try to move PRODUCT from disk to memory. (See step 25.) What happened?

Error 170 means the file is missing. Clear out **PROGRAM 22** from the diskette. (See step 26.) Turn off the computer.

4-3 DISCUSSION

Doing Arithmetic on the Computer

We are concerned with five arithmetic operations. These are addition, subtraction, multiplication, division, and exponentiation (+, −, *, /, ^). The first four are familiar to you. The last (exponentiation) may be unfamiliar, but is not nearly as complicated as its name suggests.

The exponentiation operation is represented by the ^ symbol. Exponentiation merely means "raised to the power." Therefore, 3^4 means "3 raised to the fourth power," or 3x3x3x3, giving 81 as the result. (The computer actually gives 80.99999834 due to the way it handles exponentiation.)

You need to understand the order in which the computer performs arithmetic operations. Consider the following expression:

$$2+3\text{^}2/5-1$$

If the computer simply performed operations starting at the left as they occur in the expression, it would add 2 plus 3 (giving 5), raise 5 to the second power, (giving 25), divide by 5 (giving 5), and subtract 1 producing an answer of 4. However, suppose addition and subtraction are done first, then exponentiation, then multiplication and division. This order would give 5 raised to the second power (25) divided by four, for an answer of 6.25.

Different rules for the order of arithmetic operations could produce other answers. However, there are well-defined rules in BASIC for the order and priority of arithmetic operations. They are:

Operations are performed from left to right, using the following priority rules.

The priority for arithmetic operations is

1. Exponentiation
2. Multiplication and division
3 Addition and subtraction

Now going back to the example:

$$2+3\text{^}2/5-1$$

First you scan left to right for exponentiation. Since there is an exponentiation indicated (3^2), it is done first. Now the expression is:

$$2+9/5-1$$

You scan from left to right for exponentiation again, and finding none, look for the operations with the next highest priority, multiplication and division. The division is therefore done next, with the following result:

$$2+1.8-1$$

Since there are no more multiplications or divisions in the expression, you scan from left to right for addition and subtraction. The addition gives

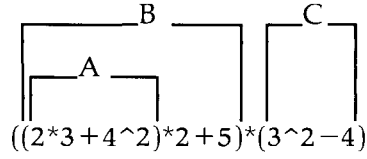
$$3.8-1$$

and the final subtraction produces the answer of 2.8.

Review the rules for order and priority of arithmetic operations until they become second nature to you.

Using Parentheses in Computations

The rules for order and priority are not the whole issue in arithmetic operations. Consider the following example:



The difference between this expression and the ones you have been studying is that parentheses are used here to group parts of the expression. We will go through this example in great detail to show you how the computer attacks the arithmetic.

The computer starts by scanning from left to right and meets the left parenthesis of **B**. It then looks inside to see if there are any more left parentheses and finds the one for **A**. The next parenthesis met is a right parenthesis for **A**. At this point, the computer has isolated the first group of operations to be done. This is:

```
2*3+4^2
```

and is evaluated using the order and priority rules. The result is 22 (check it). Now the problem has become:

```
     B       C
  ┌─────┐ ┌─────┐
(22*2+5)*(3^2-4)
```

On the next scan, the computer isolates the right parenthesis of **B**, does the arithmetic inside, and the problem is now

```
       C
    ┌─────┐
49*(3^2-4)
```

Since there are only the **C** parentheses left, the computer does the arithmetic inside, giving

```
49*5
```

which produces the answer of 245.

Thus, if parentheses are nested, the computer works out from the deepest set, scanning from left to right. When a set of parentheses is removed, the arithmetic operations inside are done according to the order and priority rules given in the preceding section. A very good rule of thumb for the beginner is to use extra parentheses if there could be any confusion about how the computer will evaluate an expression. Too many cannot hurt, but too few certainly can.

Using E Notation for Numbers

BASIC prints numbers in different forms. In particular, BASIC uses the E notation for very large or very small numbers. Examples of the E notation are 2.145E+06 or 6.032E−07.

The reason you need this special notation is that the computer usually prints out only nine digits for a number. A problem arises if you want to print out a variable whose value is 45612800000, eleven digits. The computer will print this out as 4.56128E+10. The E+10 means that the decimal point belongs ten places to the right of its present position. You can express very small numbers in the same way. A variable whose value is 0.0000000683 would be printed out as 6.83E−08. The E−08 means that the decimal point belongs eight places to the left. The table below should help you understand how to convert from decimal to E notation or from E notation back to decimal notation.

Decimal Form	E Notation
2630000	2.63E+06
263000	2.63E+05
26300	2.63E+04
2630	2.63E+03
263	2.63E+02
26.3	2.63E+01
2.63	2.63
0.263	2.63E−01
0.0263	2.63E−02
0.00263	2.63E−03
0.000263	2.63E−04
0.0000263	2.63E−05
0.00000263	2.63E−06

To change from E notation to decimal notation, look at the sign following the E. If the number is +, move the decimal point to the right as many places as the number. If the sign after the E is −, move the decimal point to the left. To convert from decimal to E notation, just write E + or − however many places the decimal has moved left or right, respectively.

Actually, you shouldn't get very tense about the E notation, since you will rarely use it. The main reason for bringing it up is that the computer may print out numbers in the E notation. Consequently, you should be able to recognize what is happening.

Formatting a Diskette

As you saw in the discovery exercises, formatting a diskette (new or old) is a straightforward task. With the computer turned off, you turn on the disk drive. When the busy light goes off, you place the

Master Diskette II into the drive and close the door. (Note: Make a copy of this diskette so that you can use the copy and preserve the original. See *An Introduction to the Disk Operating System*, page 12). Then you turn on the computer. DOS will be loaded. If the BASIC cartridge is not in place, the menu of disk drive operations will appear on the screen. If the BASIC cartridge is in place you can display the menu of disk drive operations by typing in DOS. At this point, you replace the diskette in the disk drive with the one to be formatted. To accomplish this, select I from the menu, 1 for the disk drive request, and Y to complete the process. This clears **all** programs and information from the diskette. Copying the DOS files to a formatted diskette will allow you to get back and forth between disk operations and BASIC. To copy the DOS files right after formatting the diskette, select H from the menu and again respond with 1 to the disk drive request, then with Y to complete the process.

Storing and Retrieving Programs

When you turn off the computer, you lose the program in memory. If every time you turned on the computer, you had to type in programs you wanted to use, you would get very little work done. Fortunately, with the ATARI computer you can type in long or complicated programs, troubleshoot them, and then store the programs on a formatted diskette for future use. To retrieve a stored program you need only turn on the disk drive, place the disk you need into the drive, turn on the computer, and bring up ATARI BASIC.

When you wish to save a program on diskette, you need to give it a name. You can use any name provided it starts with a letter and is eight characters or less. From the second character on, any letter or digit may be used in the name. For instance, you could use PRODUCT or TEST1. You should choose meaningful names to help you remember what each particular program does.

- **To move a program from memory to diskette, type**
  SAVE "D: ‹name of program›"

- **To move a program from diskette storage to memory, type**
  LOAD "D: ‹name of program›"

- **To move a program from diskette storage to memory and execute it, type RUN "D: ‹name of program›"**

You might not want to keep a specific program on a diskette forever. The disk drive operations menu provides a way to clear programs from diskette storage. To clear out a program on a diskette, select the appropriate letter in the menu of the disk drive operations menu. As you have seen you can obtain this menu by typing DOS and RETURN . You can get back to ATARI BASIC by selecting B from the menu.

You must be very careful when using the SAVE command. For instance, suppose you wish to move a program from memory to diskette and inadvertently type

```
SAVE "D:PRODUCT"
```

If PRODUCT is the name of another program saved on the diskette in the disk drive, then the program on diskette will be replaced by the program in memory. This is fine if the program in memory is an update of the program on diskette. If not, however, you have destroyed a program you wanted to keep. You can avoid this kind of problem by making back-up or spare copies of diskettes. Keeping a record of all the files you save on a diskette can also help you avoid this situation.

4—4 PRACTICE TEST

1. Write down the symbols used to signify the following arithmetic operations in BASIC expressions:

   a. Multiplication

   _____

   b. Exponentiation

   _____

   c. Division

   _____

2. When evaluating arithmetic expressions, what is the computer's priority of operations?

   a. First

   b. Second

   c. Third

3. When the computer scans arithmetic expressions, in what direction does it search?

4. Write a BASIC statement equivalent to the following expression. Number the line 100.

$$A = (4 + 3B/D)^2$$

5. If the computer runs the following program, what will it type out?

```
100 LET A=2
110 LET B=3
120 LET C=(A*B+2)/2
130 PRINT C
140 END
```

6. Convert the following numbers to E notation:

   a. 5160000

b. 0.0000314

7. Convert the following numbers to decimal notation.

a. 7.258E+06

b. 1.437E-03

8. In the expression below, give the order in which the computer will perform the operations.

```
100 LET A=(6/3+4)^2
```

9. What commands does the computer need to carry out the following operations?

a. Moving a program from diskette storage to memory

b. Moving a program from memory to diskette storage

c. Clearing out a program in diskette storage

d. Clearing out a program in memory

e. Displaying the program in memory

f. Executing the program in memory

g. Displaying the names of all the files saved on diskette

10. Suppose you are typing a line into the computer and have not yet pressed RETURN. How do you correct a single character?

CHAPTER 5

INPUT AND OUTPUT

5—1 OBJECTIVES

In this chapter you will get down to the business of writing programs. You will also increase your knowledge of BASIC by looking at some details about input and output.

Getting Numbers into a BASIC program

There are only three ways to enter numbers into the computer for a BASIC program. Since the computer is concerned mainly with numbers, you need to understand how to input these numbers.

Printing out Variables and Strings

After information is computed, it must be printed out. There are different kinds of output, but usually you will want to output strings of characters as well as numbers. This string output is handled essentially the same as numbers, but needs special attention.

Spacing the Printout

The computer has a built-in spacing mechanisms, but you can use punctuation to signal the computer to space output as you desire, for legibility and sense.

Using the REM Statement

The wise programmer includes comments in programs to help explain or interpret what is being done. The REM statement in BASIC lets you do this.

Working with Program Examples

Your ultimate goal is to learn how to write and troubleshoot programs. In this chapter you will begin with some simple program assignments.

5–2 DISCOVERY EXERCISES

1. Turn on the computer and bring up ATARI BASIC. Enter the following program:

```
100 INPUT A
110 INPUT B
120 INPUT C
130 LET D=A+B+C
140 PRINT D
150 END
```

What do you think will happen if you run this program?

Run the program. When the first question mark appears, (the INPUT prompt for A), type in 2. Likewise, when the second question mark appears, type in 3, and finally, at the last question mark, type in 5. Record the output.

2. Note that in the program in step 1 we have three INPUT statements (lines 100, 110, and 120). Type

```
100
110
```

What does this do to the program?

Display the program in memory and see if you are right. Then type

```
120 INPUT A,B,C
```

Display the program. What happened?

3. Run the program and when the INPUT prompt (?) appears, type in

```
2,3,5
```

What happened?

Can you input more than one variable at a time in a BASIC program?

4. Run the program, and when the INPUT prompt appears type

```
2,3
```

What happened?

What is the computer waiting for?

Type

```
5
```

What happened?

5. Run the program and when the INPUT prompt appears, type

```
2,3,5,1
```

What happened?

6. Can you type in more numbers than are called for at an INPUT statement?

What will happen if you do?

7. Can you type in fewer numbers than are called for at an INPUT statement?

What will happen if you do?

8. Type

```
120 READ A,B,C
```

Display the program. What happened?

Run the program and record the output.

Error 6 means the computer is out of data.

9. Now type

```
125 DATA 2,3,5
```

and display the program. What happened?

10. Run the program and record the output.

Based on what you have just seen, when a BASIC program contains a READ statement, there must be another type of statement in the program. What is that statement?

11. Name two different methods (other than using a LET statement) for getting numbers into a program. (Hint: See steps 1 and 8.)

12. Display the program in memory. Delete the DATA statement (line 125). Type

```
145 DATA 2,3,5
```

and display the program again. What happened?

13. Run the program and record the output.

Does it make any difference where the DATA statement is in the program?

14. Clear the program in memory. Enter the program below

```
100 READ A,B
110 LET C=A/B
120 PRINT C
130 GOTO 100
140 DATA 2,1,6,2,90,9,35,7
150 END
```

What do you think will happen if you run the program?

Try it and see if you were correct. Record the output.

Is the Error 6 (out of data) message associated with the READ statement or the DATA statement?

15. Delete the DATA statement in line 140 from the program, and enter the following statements:

```
105 DATA 10,2
115 DATA 100,50
125 DATA 50,5
```

Display the program in memory. What happened?

16. If you run the program, what do you think will be displayed?

Run the program to see if you were correct. Record the output.

17. Can you have more than one DATA statement in a BASIC program?

Does it make any difference where the DATA statements are in the program?

18. Clear out the program in memory. Enter the following program:

```
100 LET A=10
110 PRINT A
120 END
```

What will happen if you run this program?

Run the program and record the output.

19. Now type

```
110 PRINT "A"
```

and display the program in memory. What happened?

What will happen if you run the program?

Run the program and record the output.

20. Type

```
110 PRINT "HOUND DOG = ";A
```

and display the program in memory. What do you think will happen if you run this program?

Run the program and record the output.

21. Type

```
110 PRINT "B = ";A
```

Display the program and study it carefully. What do you think will happen if you run the program?

Try it and see if you were right. Record the output.

22. Type

```
95 REM DEMO PROGRAM
```

Display the program. What happened?

Run the program, and record the output.

Does the REM statement in line 95 have any effect on the program?

23. Clear out the program in memory. Enter the following program:

```
100 REM METRIC CONVERSION PROGRAM
110 REM CONVERT LBS. TO GRAMS
120 PRINT "INPUT NO. OF LBS. ";
130 INPUT P
140 LET G=454*P
150 PRINT P;" POUNDS IS ";G;" GRAMS"
160 GOTO 120
170 END
```

Display the program to see if it is correct. Note the semicolons. Study the program carefully and try to guess what will happen if you run it. Run the program. When the INPUT prompt is typed out, enter any number you desire. Note what is typed out. Repeat this process several times, then jump the computer out of the INPUT loop. If you have forgotten how, see Chapter 2, step 21 of the discovery exercises. What is the purpose of the REM statement?

24. Type

```
115 INPUT P
130
160 GOTO 115
```

and then display the program in memory. What happened?

Will the program work in this form?

Run the program and at the INPUT prompt, type 1. What happened?

Jump the program out of the INPUT loop.

25. Clear the program in memory and enter it again, modified as follows:

```
100 REM METRIC CONVERSION PROGRAM
110 REM CONVERT LBS. TO GRAMS
120 PRINT "INPUT NO. OF LBS.";
130 INPUT P
140 PRINT P;"POUNDS IS ";G;"GRAMS"
150 LET G=454*P
160 GOTO 120
170 END
```

Can you run the program in this form?

Run the program and at the INPUT prompt, type 2. What happened?

What is wrong. Remember that if a variable is not defined in your program, your computer will define it as 0.

Jump the program out of the INPUT loop.

26. Clear out the program in memory. Enter the following program:

```
100 READ A
110 PRINT A
120 GOTO 100
130 DATA 10,12,8,9,73,60,82
140 END
```

Run the program and record the output.

27. Type

```
110 PRINT A,
```

Note that all you have done is to insert a comma after the A in line 110. Execute the program and record the output.

28. Replace the comma after A with a semicolon by typing

```
110 PRINT A;
```

Run the program and record the output.

29. If a variable in a PRINT statement is not followed by any punctuation marks, what happens after the number is printed out? (Hint: See step 26.)

Suppose the variable is followed by a comma?

What happens if the variable is followed by a semicolon?

30. This concludes the computer work for now. Turn off the computer.

5—3 DISCUSSION

In this chapter you have begun to get away from the mere mechanics of controlling the computer and to concentrate more on writing and troubleshooting programs. This skill doesn't come naturally to most people, and consequently we will give the topic a great deal of attention, both now and in later chapters.

Getting Numbers Into a BASIC Program

In Chapter 1 you learned that one way to get numbers into a program is to assign values to a variable in the program itself. For example,

```
100 LET A=6
```

introduces the value 6 into a program and stores the number under the variable name A. This method has limitations, but there are two other ways to get numbers into a BASIC program: INPUT statements and READ and DATA statements. Let's look first at the INPUT statement and how it is used.

When the computer runs a line such as

```
260 INPUT G
```

it will type out a question mark as a prompt that input is expected from the terminal, then it will stop and wait for you to type in the number. In the case above, the number typed in will be known as G.

An INPUT statement may call for more than one variable. For example,

```
420 INPUT A,B,C,D
```

In this case the computer uses the same INPUT prompt (?) but now you must type in four numbers separated by commas. If you enter fewer than four and press the RETURN key, the computer will type another question mark and wait for the remaining numbers to be input. If you enter more than four numbers, the program will continue running using only the first four numbers you typed in.

The last method of providing numerical input is to use the READ and DATA statements. The computer handles the statement

```
100 READ A,B,C,D
```

the same way it handles an INPUT statement with two exceptions, First, the computer does not stop, and second it reads the numbers called for from DATA statements in the program.

Consider the following program:

```
100 READ A,B,C,D
110 LET E=A+B+C+D
```

```
120 PRINT E
130 DATA 25,3,17,12
140 END
```

The program reads the four numbers from the DATA statement and prints out the sum of the numbers. It makes no difference where the DATA statement is in the program. There can be more than one DATA statement, and they need not be grouped at the same place in the program. Because numbers are called for by READ statements, they are taken in order from the DATA statements, beginning with the lowest-numbered statement. If you need more numbers than are available in DATA statements, the computer will type

```
ERROR-    6 AT LINE <line #>
```

and then halt. Recall that many of the error messages are summarized on the inside of the cover.

You will become familiar with the advantages and disadvantages of each of these methods as you spend more time writing programs.

- **You can put numbers into a BASIC program with: (1) LET statements; (2) READ and DATA statements; and (3) INPUT statements.**

Printing out Variables and Strings

Output from the computer is quite simple. The computer can print out either the numerical value of a variable or a string of characters. To illustrate, suppose we have a variable named X and the number 2 is stored in that location. The program

```
100 LET X=2
110 PRINT "X"
120 PRINT X
130 END
```

shows the difference between string and variable output. Line 110 prints out the character X, because X is enclosed in quotation marks. Line 120 prints 2, because that is the number stored in location X.

The rule is clear. Any set of characters contained within quotation marks is called a string. The computer prints out strings exactly as they are listed; it does not attempt to analyze or detect what is in the string. The computer prints out the numerical value of any variable that is not in quotes.

It is possible to do computations with a PRINT statement. Thus

```
100 PRINT A+B+C,D
```

will cause the computer to print out the sum of the numbers stored in A, B, and C and the number stored in D. Of course, the variables A, B, C, and D would have to be defined or the computer will assign zeros for their values.

Spacing the Printout

BASIC has a built-in standard spacing mechanism that prints four numbers equally spaced on one line. Where possible, this standard spacing is used by the computer when quantities in a PRINT statement are separated by commas. The comma signals the computer to move to the next print position on the line. If the computer is already at the last position on a line and encounters a comma in a PRINT statement, it does a carriage return and prints the number on the first position on the next line. Thus

```
100 PRINT A,B,C,D,E
```

would cause the numerical values of A, B, C and D to be printed equally spaced across a line in the four standard positions. The numerical value of E would be printed on the next line and indented two spaces. If four or more numbers are to be printed and if one of the numbers has more than eight digits, then the number of columns will be reduced to three.

■ **Commas in PRINT statements produce 4 columns per line.**

Another type of spacing is produced by the semicolon between variables. For example, in the statement

```
100 PRINT A;B;C
```

the numbers will be printed close together and space needs to be provided. However, such spacing is relatively easy to do. For example, the direct mode statement

```
PRINT A;" ";B
```

will produce a single space between the two numbers

The semicolon can also be used in an INPUT statement to cause the program to stop and place the input prompt at the end of the statement. For example, when the statement

```
130 PRINT "WHAT IS THE PRICE? ";
```

is executed, the input prompt will be placed at the end of the question and await your input.

You can add vertical spacing to output by using an empty PRINT statment as follows:

```
100 PRINT
```

The computer looks for the quantity to be printed and finds none. It then looks for punctuation and, finding none, orders a carriage return and moves the cursor down one line. You can add as many empty lines to the printout as you wish by using empty PRINT statements.

Using the REM Statement

The REM ("remark") statement is different from the statements you have seen previously. As soon as the computer encounters the characters REM following the line number, it ignores the balance of the statement and goes on to the next line. The REM statement simply provides information to help the programmer or someone reading the program to follow what is happeneing in the program. The wise programmer will use REM statements liberally.

Below we present the same program with and without REM statements. You can decide which program is easier to follow.

With REM statements:

```
100 REM COMPUTE AVERAGE OF FOUR NUMBERS
110 REM INPUT THE FOUR NUMBERS
120 INPUT A,B,C,D
```

```
130 REM COMPUTE THE AVERAGE
140 LET X=(A+B+C+D)/4
150 REM PRINT OUT THE AVERAGE
160 PRINT X
170 END
```

Without REM statements:

```
100 INPUT A,B,C,D
110 LET X=(A+B+C+D)/4
120 PRINT X
130 END
```

- **You can describe what is happening in a program with a REM statement.**

5—4 PROGRAM EXAMPLES

Study each of the following examples until you are certain you understand all the details. You might want to enter the programs into the computer and run them to verify that they work as intended.

Example 1 - Unit Prices

Your problem is to write a program to compute unit prices of supermarket items. If you let T stand for the total price, N for the number of units, and U for the unit price, you can compute the unit price with the following relationship:

$$U = T/N$$

For example, if a case of twelve large cans of fruit juice costs $6.96, the unit cost per can would be:

$$U = 6.96/12 = \$0.58$$

You want the program to produce the following output:

```
WHAT IS THE TOTAL PRICE? (You enter value of T)
NUMBER OF UNITS? (You enter value of N)
UNIT PRICE IS (Computer types out value of U)
```

Break the example apart to see how the program is related to what you want to see in the output.

```
WHAT IS THE TOTAL PRICE? (entry of T)
        100                      200
NUMBER OF UNITS? (entry of N)
        300              400
500: Compute unit price.
UNIT PRICE IS (output of U)
      600              700
```

You will write each line of the program so that the numbers below each statement will be the line numbers in the program. In line 100 you will use the PRINT command to tell the computer to type out the message indicated.

```
100 PRINT "WHAT IS THE TOTAL PRICE? ";
```

Note the semicolon outside the closing quotation marks. The reason for this is that you do not want a carriage return; you want the printed line to hold there for the INPUT prompt on line 200:

```
200 INPUT T
```

Use a PRINT statement to get the computer to print out the message in line 300.

```
300 PRINT "NUMBER OF UNITS? ";
```

Handle the input for the total number of items the same as the total price.

```
400 INPUT N
```

Next you compute the unit price in line 500.

```
500 LET U=T/N
```

Use a PRINT statement to handle the next line, a message followed by the unit price.

```
600 PRINT "UNIT PRICE IS ";U
```

Finally, add an END statement.

```
700 END
```

Now put the whole program together.

```
100 PRINT "WHAT IS THE TOTAL PRICE? ";
200 INPUT T
300 PRINT "NUMBER OF UNITS? ";
400 INPUT N
500 LET U=T/N
600 PRINT "UNIT PRICE IS ";U
700 END
```

Study the program to make sure you see the purpose of each line as related to the original statement of the problem.

Example 2 - Converting Temperature

The relationship between temperature measured in degrees Fahrenheit and in degrees Celsius is

$$C = (5/9)(F - 32)$$

where C stands for degrees Celsius and F stands for degrees Fahrenheit. If, for example, F is 212, then C is

$$C = (5/9)(212 - 32) = 100$$

As in the first example, you will write the program after deciding how you want the output to appear. Suppose you want to see the following:

```
INPUT NO. OF DEGREES F
?  (You enter value of F)
(Value of F) DEGREES F IS (answer) DEGREES C
```

Split the output up into parts that will be generated by the lines in the program.

```
INPUT NO. OF DEGREES F
          100
```
200: Entry of F
300: Compute C
(Output of F) DEGREES F IS (Output of C) DEGREES C
```
                    400
```

The corresponding program is

```
100 PRINT "INPUT NO. OF DEGREES F"
200 INPUT F
300 LET C=(5/9)*(F-32)
400 PRINT F;" DEGREES F IS ";C;" DEGREES C"
500 END
```

This program is a bit different from the first example. In line 100 there is no puncuation following the string. Thus the INPUT prompt generated by line 200 will be printed out on the line following the initial string. The PRINT statement in line 400 prints out (1) the value of F, (2) a string, (3) the value of C, and (4) a second string. The semicolons in line 400 are used to put space between the variables and strings in the PRINT statements.

Example 3 - Sum and Product of Numbers

Suppose you want to compute the sum and product of two numbers. when the program is run, you want to see:

```
INPUT A? (You enter value of A)
INPUT B? (You enter value of B)
SUM OF A AND B IS (Computer prints out sum)
PRODUCT OF A AND B IS (Computer prints out product)
(Blank lines are inserted by the computer.)
INPUT A? (You enter a second value of A)
(etc.)
```

Since you studied the first two examples in great detail, you can proceed more rapidly with this problem. The first line of the output can be handled by the following statements:

```
100 PRINT "INPUT A ";
110 INPUT A
```

Note that the message printed out in line 100 is window dressing for the program and has nothing to do with the actual calculations. The input instruction that is important to the computer is in line 110. However, such messages are important to you because they tell you what to do. You can generate the second line of desired output in the same manner.

```
120 PRINT "INPUT B ";
130 INPUT B
```

Use the following lines to generate the sum and product of the two numbers.

```
140 PRINT "SUM OF A AND B IS ";A+B
150 PRINT "PRODUCT OF A AND B IS ";A*B
```

The spacing between the output of the original set of numbers and the output obtained when the program loops back, as well as the looping instructions, can be obtained with three statements.

```
160 PRINT
170 PRINT
180 GOTO 100
```

Of course, the final line should be the END statement.

```
190 END
```

The whole program is:

```
100 PRINT "INPUT A ";
110 INPUT A
120 PRINT "INPUT B ";
130 INPUT B
140 PRINT "SUM OF A AND B IS ";A+B
150 PRINT "PRODUCT OF A AND B IS ";A*B
160 PRINT
170 PRINT
180 GOTO 100
190 END
```

You could also compute the sum and product of the two numbers using LET statements as in the following version.

```
100 PRINT "INPUT A ";
110 INPUT A
120 PRINT "INPUT B ";
130 INPUT B
140 LET S=A+B
150 PRINT "SUM OF A AND B IS ";S
160 LET P=A*B
170 PRINT "PRODUCT OF A AND B IS ";P
180 PRINT
190 PRINT
200 GOTO 100
210 END
```

Both forms of this program will keep looping back until you jump the the program out of the INPUT loop.

5—5 PROBLEMS

1. Write a program that will read the four numbers 10, 9, 1, and 2 from a DATA statement, putting the numbers in A, B, C, and D, respectively. Add the first two numbers putting the sum in S. Then compute the product of the last two numbers, putting the result in P. Print out the value of S and P on the same line.
2. Write a three line program that will call for the input of four numbers and then print back the numbers in reverse order. For example, if you type in 5, 2, 11, 12, the computer should type back 12, 11, 2, 5. The program must work for any set of four numbers that you decide to type in. Use only three lines in your program.
3. Write a program to read variables A, B, C, and D from numbers of your choice in a DATA statement and print out the numbers vertically.
4. What will be output if you run the following program?

```
100 READ X,Y,Z
110 DATA 2,5,3
120 LET T=X*Y+Z
130 LET S=Y^2
140 PRINT T,S
150 END
```

5. What is wrong with this program?

```
100 LET A=2
110 READ B
120 LET A=A+C/B
130 DATA 3
140 PRINT A
150 END
```

6. Explain in your own words what the following program does.

```
100 INPUT A,B
110 LET S=A+B
120 LET T=A-B
130 LET U=A*B
140 PRINT S,T,U
150 END
```

7. One of the ratios used to judge the health of a business is the acid-test ratio. The acid-test ratio is the sum of cash, marketable securities, and receivables, divided by current liabilities. Write a program that requests input of the necessary quantities and computes and outputs the acid-test ratio.

8. Write a program to count and print out by fives beginning with 0. The first few numbers will be 0, 5, 10, 15, and so on. Interrupt the program manually when 40 or 50 numbers have been printed out.

9. The intended output of the program below is 1, 3, 5, 7, 9, and so forth. The program below has an error. What is wrong?

```
100 LET A=1
110 PRINT A;" ";
120 LET A=A+2
130 GOTO 100
140 END
```

10. If an object is dropped near the surface of the earth, the distance it will fall in a given time can be determined by

$$S = 16T^2$$

where S is the distance fallen (in feet) and T is the time of fall (in seconds). Write a program that, when executed, will produce the following output:

```
TIME OF FALL (SEC) ?   (You enter T)
OBJECT FALLS (Computer types out S) FEET
```

11. The volume of a box can be computed as

$$V = LWH$$

where L, W, and H are the length, width, and height. If these are all measured in centimeters, for example, the volume will be in cubic centimeters. You want a program that will produce the following output:

```
LENGTH (CM) ?   (You enter L)
WIDTH (CM) ?   (You enter W)
HEIGHT (CM) ?   (You enter H)
VOLUME IS (Computer types out V) CUBIC CM
```

The program below is incorrect and will not produce the output you want. What is wrong?

```
100 PRINT "LENGTH (CM) ";L
110 PRINT "WIDTH (CM) ";W
120 PRINT "HEIGHT (CM) ";H
130 INPUT L,W,H
140 LET V=L*W*H
150 PRINT "VOLUME IS"
160 PRINT V
170 PRINT "CUBIC CM"
180 END
```

12. In the program below, the INPUT statement calls for two numbers, A and B. Supply the missing statements so that when A and B are printed out, the values have been interchanged.

```
100 INPUT A,B
110
120
130
```

```
140 PRINT A,B
150 END
```

13. Suppose the odometer on your car reads R_1 miles when the gas tank is full. You drive until the odometer reading is R_2 at which point G gallons of gasoline are required to fill the tank. The miles per gallon you get on the trip is

$$M = (R_2 - R_1)/G$$

Write a program to figure out the mileage for the following data:

R_1	R_2	G
21423	21493	5
05270	05504	13
65214	65559	11.5

14. There is an old tale of a wise man who invented the game of chess and as a reward asked to receive 1 grain of wheat on the first square of the chess board, 2 grains on the second, 4 grains on the third, 8 on the fourth, and so on. Write a program to print out the number of the square and the number of grains on that square. The program should involve a loop using a GOTO statement and should be interrupted at the keyboard when you have seen enough. How many grains of wheat will be on the 64th square? Run the program and find out.

15. It is known that a DATA statement contains examination grades for a class of ten students. Write a program of no more than four statements (counting the DATA and END statements) to compute and print out the class average. Try out the program on sample data of your choice.

16. Simple interest on an investment is computed according to the following rule:

$$I = (P)(R/100)(T/365)$$

where P is the principal invested at an annual interest rate R (expressed in percent) for a time T (expressed in days). Write a program that will generate the display shown below:

```
WHAT IS THE PRINCIPAL
? (You type in the principal)
WHAT IS THE ANNUAL INTEREST RATE (%)
? (You type in the interest rate)
WHAT IS THE TERM IN DAYS
? (You type in the term)

FOR AN INVESTMENT OF
(Computer types out the principal)
AT AN ANNUAL INTEREST RATE OF
(Computer types out the rate)
PERCENT INVESTED FOR
(Computer types out the term)
DAYS, THE INTEREST IS
(Computer types out the interest)
```

17. If compound interest is paid, the true annual interest rate is higher than the nominal rate which is quoted for the investment. The following BASIC formula computes this true annual interest rate:

```
T = ((1+R/(100*M))^M-1)*100
```

In this expression, T is the true annual interest rate in percent, R is the nominal interest rate in percent, and M is the number of times the interest is compounded per year. Write a program that will produce the following output:

```
QUOTED INTEREST RATE (PERCENT)
? (You type in the interest rate)
NUMBER OF TIMES COMPOUNDED PER YEAR
? (You type in times computed)
TRUE ANNUAL INTEREST RATE IS
(Computer types out answer)
```

18. If an amount of money P is left to accumulate interest at I percent compounded J times per year for N years, the value of the investment will be

```
T = P*(1+I/(100*J)^(J*N)
```

Write a program that will call for the input of P, I, J and N. Run the program as needed to get the value of $1000 invested at 8 percent for 2 years compounded

a. Annually (J = 1)
b. Semiannually (J = 2)
c. Monthly (J = 12)
d. Weekly (J = 52)
e. Daily (J = 365)

If a savings and loan company conducts a big advertising campaign about computing interest every day instead of each week, should you get excited?

19. If an amount of money P is left to accumulate interest at a rate of I percent per year for N years, the money will grow to a total amount T given by

$$T = P*(1+I/100)^N$$

As an example, if P = $1000, I = 6 percent, and N = 5 years,

$$T = 1000*(1+6/100)^5 = 1338.23$$

Write a program that when executed will produce the following output:

```
INITIAL INVESTMENT ?  (You enter P)
ANNUAL INTEREST RATE (%) ?  (You enter I)
YEARS LEFT TO ACCRUE INTEREST ?  (You enter N)
TOTAL VALUE IS (Computer types out T)
```

5–6 PRACTICE TEST

1. What will be the output if you run the following program?

```
100 LET X=1
110 PRINT X,
120 LET X=X+1
130 GOTO 110
140 END
```

2. Describe three ways you can put numbers into a BASIC program.

3. In a PRINT statement, what is a collection of characters between quotation marks called?

4. What is the purpose of the REM statement?

5. If there is a READ statement in a BASIC program, what other type of statement must also be present in the program?

6. What will happen if you run the following program?

```
100 LET X=3
110 LET Y=4
120 PRINT "Y = ";X
130 END
```

7. How many standard print columns per line are provided for in BASIC when the print quantities are separated by commas?

8. How many DATA statements can you put in a program?

9. What will the output look like if you run the following program?

```
100 LET A=1
110 LET B=3
120 PRINT A,B
130 PRINT A;B
140 END
```

10. Suppose you're running the program:

```
100 INPUT A,B
110 LET C=A+B
120 PRINT C
130 END
```

In response to the INPUT prompt you type the numbers 10, 12, and 13 on the same line. What will be printed out?

11. You can convert miles to kilometers by multiplying by 1.609. Thus, 10 miles equals 16.09 kilometers, and so on. Write a program that will produce the following printout.

```
INPUT NO OF MILES? (You type in a number)
(Computer types your number) MILES EQUAL
(Computer types answer) KILOMETERS
```

CHAPTER 6

DECISIONS AND BRANCHING

6—1 OBJECTIVES

The power of the computer rests in large part on its ability to make decisions about quantities in programs. In this chapter we will explore this cabability and will continue the task of learning to program in BASIC.

Making Transfer Decisions in Programs

Decisions made in programs can cause the computer to jump to line numbers out of numerical order. Such a transfer to a program line may be unconditional or may depend on values of variables in the program. You will learn to use these conditional and unconditional transfer statements to make simple programs produce powerful and useful results.

Working with Program Examples

As in the previous chapter, you will continue to learn how to apply the techniques you study to BASIC programs.

Finding Errors in Programs

When first written, almost all programs have errors. You will learn the vital skill of troubleshooting programs.

6–2 DISCOVERY EXERCISES

1. Turn on the computer, bring up ATARI BASIC and enter the following program:

```
100 LET X=1
110 PRINT X
120 LET X=X+1
130 IF X<5 THEN 110
140 END
```

The < symbol in line 130 means "less than"; thus, the statement means as "If X is less than 5 then print X." Study the program carefully. What do you think will be printed out if you run the program?

Run the program and record the output.

2. Now type

```
100 LET X=2
```

Display the program in memory. What will the output be now?

Run the program and write down what the computer printed out.

3. Now make another change in the program to see if you understand what is happening. Type

```
120 LET X=X+2
```

Display the program and study it carefully. What do you think the program will do now?

Run the program and see if you were right. Record the output.

4. To explore a new idea, you need to make some changes in the program now in memory. Modify the program to make it agree with the one below or clear the program in memory and enter the one below.

```
100 LET X=1
110 PRINT X
120 LET X=X+1
130 IF X>=5 THEN 140
135 GOTO 110
140 END
```

Run the program and record the output.

Compare the output you recorded above to the output you recorded after step 1. Is there any connection?

5. Display the program in memory. Line 130 of this program is the assertion X > = 5, which is means "X is greater than or equal to 5." If, for example, X had the numerical value 6, the assertion would be true. If X had the value 3, the assertion would be false.

Now look at the program in step 4. If the program is executed, the computer starts with line 100, then goes to lines 110, 120, and 130. If the assertion in line 130 is true, which line number will the computer execute next?

6. Only two conditions have been used so far in the programs. They are

   < (Less than)
   >= (Greater than or equal to)

   How would you write the conditions for

   Greater than

   _____

   Less than or equal to

   _____

   Equal to

   _____

   Not equal to

   _____

   If you can fill in the blanks above without too much trouble, fine. If not, don't worry. We will review everything later. The important thing to grasp now is how the IF THEN statement works.

7. Let's explore some applications of the IF THEN statement. Clear the program in memory and enter the following program:

```
100 PRINT "INPUT EITHER 1, 2, OR 3 ";
110 INPUT Y
120 IF Y=1 THEN 150
130 IF Y=2 THEN 170
140 IF Y=3 THEN 190
150 PRINT "BLOOD"
160 GOTO 100
170 PRINT "SWEAT"
180 GOTO 100
190 PRINT "TEARS"
200 GOTO 100
210 END
```

Display the program and check that you have entered it correctly. Study the program briefly. Remember that when the computer executes the program and types out the INPUT prompt, you are supposed to type in either 1, 2, or 3. Which value of Y will let the computer reach line 120 in the program?

Which value or values of Y will let the computer reach line 130?

How about line 140?

8. Suppose you wanted the computer to type "SWEAT". What value of Y should you enter?

See if you were right. Run the program and enter the number you wrote down. What happened?

9. What value of Y will cause the computer to type BLOOD?

How would you make the computer type TEARS?

Check each of your responses above to see if you were right.

10. The program assumes that either 1, 2, or 3 will be typed in at the INPUT prompt. Think about the program a bit, then try to figure out what will happen if you type 4 in response to the input prompt. What do you think will happen?

Run the program, type 4 in response to the input prompt, and record the output.

You can easily explain what happened in the program by considering what the computer does when it encounters an assertion in the IF THEN statement. Remember, if the assertion is true, the computer goes to the line number following the THEN. If the condition is false, the computer goes to the next higher line number. Now jump the computer out of the INPUT loop.

11. Delete lines 150 through 190 and display the program to make sure the lines were deleted.

12. Now use another form of the IF THEN statement. Change lines 120, 130, and 140 as follows

```
120 IF Y=1 THEN PRINT "BLOOD"
130 IF Y=2 THEN PRINT "SWEAT"
140 IF Y=3 THEN PRINT "TEARS"
```

Run the program. Is a line number required after the THEN in an IF THEN statement?

---

13. Now jump the computer out of the input loop and add line 150 as follows

```
150 IF Y>3 THEN 210
```

Display the program. What will happen now if you enter 4 when you run the program?

---

14. You can control a program with a single keystroke by using the GET statement. Clear the program in memory and type in the following program:

```
80 DIM A$(1)
90 OPEN #1,12,0,"K:"
100 PRINT "PRESS A KEY"
110 GET #1,A
120 LET A$=CHR$(A)
130 IF A$="Q" THEN 160
140 PRINT A$
150 GOTO 100
160 PRINT "QUIT"
170 END
```

Do not be concerned with the statements in lines 80, 90, 110, and 120. They will be covered later. For now, just be careful to type them in correctly. Try pressing several keys (including the spacebar) before you press Q. Run the program. What happened?

Press Q to stop the program. Is Q printed out by line 130?

15. Now let's look at a use of the GET statement in a graphics program. Clear the memory and type the following program.

```
80 REM POINT MOVING PROGRAM
90 DIM Y$(1)
100 OPEN #1,12,0,"K:"
110 GRAPHICS 8
120 LET A=160
130 LET B=96
140 REM PLOTS A WHITE POINT AT (A,B)
150 COLOR 1
160 PLOT A,B
170 GET #1,X
180 LET Y$=CHR$(X)
190 REM PLOTS A BLACK POINT AT (A,B)
200 COLOR 0
210 PLOT A,B
```

```
220 IF Y$ ="D" THEN B=B+5
230 IF Y$ ="U" THEN B=B-5
240 IF Y$ ="L" THEN A=A-5
250 IF Y$ ="R" THEN A=A+5
260 IF Y$ ="Q" THEN 280
270 GOTO 150
280 END
```

16. This program allows you to use the U, D, L, and R keys to move a point around the screen. Run the program and move the point just off the screen by pressing U a number of times. As it disappears, you will get an error message. To see what quantity is illegal, in direct mode type in

```
PRINT A,B
```

Run the program again and move the point to the upper left-hand corner of the screen being careful not to go off the screen. Move the point about the screen to get the feel of this program. Type Q to end the program.

How far does the point move each time you press U, D, L, or R? (Listing the program in text mode may be helpful.)

---

17. Save this program on a diskette under the name POINT for use at a later time.

18. Enter text mode. Clear the screen and the memory. Type the following program.

```
100 LET S=0
110 INPUT Y
120 IF Y=11111 THEN 150
130 LET S=S+Y
140 GOTO 100
150 PRINT S
160 END
```

This program is supposed to add up numbers that are input. The input value that causes the sum to be printed out is 11111. It is not part of the sum.

19. Run the program and each time the INPUT prompt is displayed, type in one number from the following sequence of numbers (remember to press RETURN after each number).

```
3   1   6   5   11111
```

What value is printed out for S?

Is this value of S the sum of the numbers you input?

20. Let's try to find out why. Type the following line

```
135 PRINT "S = ";S
```

List the program. Run the program and input the same values as in step 19 (3,1,6,5,11111). Compare the values of S in line 135 to the values input in line 110.

Though you may have already discovered the logical error in the program, trace the execution of the program by going through the program as if you were the computer. Do this for the first one or two input values and then for the last two of the input values. Observe that the GOTO statement in line 140 goes to the wrong line, that is, line 100, where S is reset to zero each time.

21. Delete line 135 and retype line 140 as follows

```
140 GOTO 110
```

Run the program and input the same values (3,1,6,5,11111). Is the printout correct this time?

Run the program a final time with different values to verify its correctness.

22. Turn off the computer and go on to the discussion of the objectives.

6–3 DISCUSSION

Making Transfer Decisions in Programs

In this chapter we are concerned mainly with transfer statements, both conditional and unconditional, as well as their use in programs. Before getting to the programming, we will discuss each type of transfer statement.

a. Unconditional Transfers

From the very beginning of the book, we have been using unconditional transfer statements. The following program illustrates the use of the unconditional transfer statement:

```
100 LET Z=2
110 PRINT Z
120 LET Z=2*Z
130 GOTO 110
140 END
```

Recall that when ordered to execute a BASIC program, the computer goes to the statement with the lowest line number and then executes the statements in increasing line number order. The only way to interrupt this is with a transfer statement (or, as you will see in the next chapter, a loop command). In the program above, the computer would execute line numbers as follows: 100, 110, 120, 130, 110, 120, 130, 110, 120, 130, and so on. The point is that the statement in line 130 causes the computer to jump back to line 110 instead of going to 140. Note that there are no conditions attached to the statement in line 130. For this reason the GOTO statement is known as an unconditional transfer statement. It is also clear that the GOTO statement in this case puts the program into a loop, and there is no way out. The only way we can get the computer out of the loop is to interrupt the program from the keyboard while it is running.

- **GOTO is unconditional**

To sum up, if at some point in a program you want the computer to make an unconditional jump to another line without any conditions attached, use the GOTO statement. However, be careful that you don't get the program "hung up" in a loop.

b. Conditional Transfers

By now you have most likely established the connection between the IF THEN statements you saw in the discovery exercises and the notion of the conditional transfer statement. All conditional transfer statements have the same form. A description of this form and a sample IF THEN statement are given below:

Line # IF <relation> <condition> <relation> THEN Line #

```
240 IF 3*X-2>Y-Z THEN 360
```

■ **IF THEN statements state conditions.**

No matter what the assertion, all IF THEN statements have this same format. The IF and THEN as well as the two line numbers in the statement require no special explanation. However, the heart of the statement lies in the two expressions separated by the condition that forms the assertion. We must look at them very carefully.

Several conditions may be used in the IF THEN statement. The conditions and their meaning are listed below.

Condition	Meaning
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Except for the IF THEN statement above, you have seen only simple variables and constants in IF THEN statements. This is the type of assertion used in programs most often. Examples are

```
100 IF W<3 THEN 250
```

```
340 IF S>T THEN 220
```

There are instances, however, when you might want to use more complicated expressions in the IF THEN statements. In the statement

```
240 IF 3*X-2>Y-Z THEN 360
```

the first relation is

```
3*X-2
```

which is fine providing that X has a value. The second relation,

```
Y-Z
```

can also be used if Y and Z have values. Suppose that X has the value 1, Y is 10, and Z is 4. The computer will translate the statement

```
240 IF 3*X-2>Y-Z THEN 360
```

by substituting the values of X, Y, and Z. This changes the statement to

```
240 IF 1>6 THEN 360
```

Sooner or later, all IF THEN statement are reduced to this form, from which the computer must judge whether an assertion established by two numbers and a condition is true or false. In this case, the assertion 1 > 6 is false. However, the assertion 4 < 10 would be true. If the assertion is true, the computer will go to the line number following THEN. If the assertion is false, the computer will go to the next higher line number in the program.

- **A true IF THEN statement causes the computer to branch; a false one causes it to go to the next higher line number.**

When the "statement" following the THEN is a number, the computer will branch to the line with that number if the IF THEN statement is true. If the statement following the THEN is another statement, the computer will execute that statement and proceed to the next line.

With IF THEN (or conditional) transfer statements, you can make the computer branch anywhere you desire in a program. This ability gives the computer its great programming potential.

6—4 PROGRAM EXAMPLES

Example 1 - Printout of Number Patterns

The problem is to write a program that will make the computer print out the following number pattern:

```
 2      3      4      5
  6      7      8      9
   10
```

You must think about several characteristics of this pattern when you write the program. The first number is 2, and succeeding numbers are spaced across in the standard manner (four numbers to a line). Each number is 1 greater than the previous one. The last number is 10, then the computer should stop.

Several solutions are possible. This program, though not elegant, will work:

```
100 PRINT 2,3,4,5,6,7,8,9,10
110 END
```

You might check this program to see that is does in fact produce the correct number pattern. This program illustrates a very important concept. There is no such thing as the correct program. The only test that can be applied is "Does the program work?" Certainly some programs are cleverer or may accomplish the results more efficiently than others, but this is a separate issue.

Another way to approach the problem is to make the computer print the first number in the pattern. You also want to organize the program so that only a single print statement is required. The solution is to have the computer print the value of a variable that will change as it executes the program. You can start the program with the following segment:

```
100 LET X=2
110 PRINT X,
```

The value of X is set to 2, and this value is printed out in line 110. The comma causes the computer to space across to the next standard printing position. Now you must generate the next value to be printed out. Note that at any point in the pattern, the next

number is just 1 more than the present number. You can generate the next number with

```
120 LET X=X+1
```

Now all that remains is to give the computer a way to make a decision about whether to loop back to the print statement or to stop. As long as X is less than or equal to 10, you want to loop back, so you can use a conditional transfer statement.

```
130 IF X<=10 THEN 110
```

Finish the program with an END statement.

```
140 END
```

The complete program is

```
100 LET X=2
110 PRINT X,
120 LET X=X+1
130 IF X<=10 THEN 110
140 END
```

This program is simple and has little practical value other than to illustrate how a conditional transfer statement can get you out of a program at the proper time.

Example 2 - Automobile License Fees

In an attempt to force consumers to use lower-horsepower cars and conserve energy, the state adopts a set of progressive annual license fees based on the power rating of the car. The criteria and fees are listed below.

Horsepower	License Fee
Up to 50 hp	$0
More than 50 but 100 hp or less	30
More than 100 but 200 hp or less	70
More than 200 but 300 hp or less	150
More than 300 hp	500

You want a program that will produce the following output:

```
INPUT AUTO HP? (You type in horsepower)
LICENSE FEE IS (Computer types out fee)

INPUT AUTO HP? (You type in horsepower)
LICENSE FEE IS (Computer types out fee)

                (etc.)
```

Clearly, the only difficult part of the program will be instructing the computer to decide what the fee is. The IF THEN statement is made to order for this decision-making process. To get started, provide for input of the power rating. Use P to stand for the power rating of the car. The program can begin with

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
```

Now, you must work out a method to have the computer decide in which license category P lies. A logical way to do this would be to check upward from the low horsepower ratings. First, the computer can check whether P is 50 or less. If so, then the tax is 0.

```
120 IF P<=50 THEN _____(Fee is 0)
```

Notice that there is no line number following THEN. If the number in P is less than or equal to 50, we want the computer to jump to a statement that will assign the value 0 to the fee. The problem is that we don't know at this point what line number should be used for this statement. Consequently, we will leave it blank and insert the proper value later. The note at the right is a reminder of what the fee is supposed to be if the assertion is true and the branch is taken.

If the assertion in line 120 is false, the computer will go to the next higher line number. The statement in that line should have the computer test whether P falls in the next higher category.

```
130 IF P<=100 THEN _____(Fee is $30)
```

Again, we don't know what line number to use following the THEN but can fill it in later. We need three further branch statements to accommodate all categories of P. Now that the pattern is established, we can include them all at once.

```
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)
```

The program to this point is

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN _____(Fee is 0)
130 IF P<=100 THEN _____(Fee is $30)
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)
```

Now we can fill in the missing line number in line 120. Since the next line number in the program would be 170, we may as well use it.

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 170
130 IF P<=110 THEN _____(Fee is $30)
140 IF P<=200 THEN _____(Fee is $70)
150 IF P<=300 THEN _____(Fee is $150)
160 IF P>300 THEN _____(Fee is $500)
170 LET F=0
180 GOTO _____(PRINT statement)
```

Again, you don't yet know what line number to use in line 180. Use a reminder because you want the computer to jump to a PRINT statement after it determines the fee. If the assertion in line 120 is true, the computer jumps to line 170 and assigns the value 0 to F, which stands for the fee. We can fill in the missing numbers in lines 130, 140, 150, and 160 using the same pattern. The result is

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
```

```
120 IF P<=50 THEN 170
130 IF P<=100 THEN 190
140 IF P<=200 THEN 210
150 IF P<=300 THEN 230
160 IF P>300 THEN 250
170 LET F=0
180 GOTO _____(PRINT statement)
190 LET F=30
200 GOTO _____(PRINT statement)
210 LET F=70
220 GOTO _____(PRINT statement)
230 LET F=150
240 GOTO _____(PRINT statement)
250 LET F=500
```

The next line in the program would be 260, which you may as well use for the PRINT statement. The rest of the program is given below.

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 170
130 IF P<=100 THEN 190
140 IF P<=200 THEN 210
150 IF P<=300 THEN 230
160 IF P>300 THEN 250
170 LET F=0
180 GOTO 260
190 LET F=30
200 GOTO 260
210 LET F=70
220 GOTO 260
230 LET F=150
240 GOTO 260
250 LET F=500
260 PRINT "LICENSE FEE IS ";F
270 PRINT
280 GOTO 100
290 END
```

You may have noticed that the conditional transfer statement in line 160 is not necessary. To see why, consider each of the

assertions in the IF THEN statements. If the assertion in line 120 is false, P must be greater than 50. Likewise, if each of the following assertions is false, the computer goes to the next higher line number. Suppose the computer reaches line 150 and determines that the assertion is false. This directs the computer to line 160, but that jump is unnecessary. You already know that P must be greater than 300, and the computer can therefore print out the fee without any more testing. If you assign the license fee of $500 in line 160, you use a slightly different program:

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P<=50 THEN 200
130 IF P<=100 THEN 220
140 IF P<=200 THEN 240
150 IF P<=300 THEN 260
160 LET F=500
170 PRINT "LICENSE FEE IS ";F
180 PRINT
190 GOTO 100
200 LET F=0
210 GOTO 170
220 LET F=30
230 GOTO 170
240 LET F=70
250 GOTO 170
260 LET F=150
270 GOTO 170
280 END
```

Yet another solution for this problem is listed below. Study it to make certain you understand how it works.

```
100 PRINT "INPUT AUTO HP ";
110 INPUT P
120 IF P>300 THEN F=500
130 IF P<=300 THEN F=150
140 IF P<=200 THEN F=70
150 IF P<=100 THEN F=30
160 IF P<=50 THEN F=0
170 PRINT "LICENSE FEE IS ";F
180 PRINT
```

```
190 GOTO 100
200 END
```

All three versions of the program will work equally well, and you may have your own version. You can decide how you prefer to handle the branches. The main question is whether your program will work.

We have gone through this program in great detail because beginners often have difficulty writing programs that use transfer statements. You should study the program until you are convinced that it does accomplish what is desired. Remember to leave line numbers out when you do not know what they should be, then return later to fill in the proper numbers. Also, use comments at the right of each line that refers to an as yet undetermnind line number. Those comments will help you remember what you want to happen at that branch point in the program. In fact, consider using these comments any time you use an IF THEN statement. When you reread an old program, these comments remind you of what the program is supposed to accomplish.

Example 3 - Averaging Numbers

Suppose you wish to average the numbers in a DATA statement. The problem is that you don't know in advance how many numbers there are. So you will use a flag variable to mark the end of the data. The flag will be a number that is very unlikely to occur in the data. We will use the number 9999 as the flag here, but you could select any unlikely number.

Here is the way it will work. The DATA statement will always appear as follows:

Line # DATA (number),(number),....,(number),9999

Place the flag 9999 after the last number to be averaged. Each time the computer reads a number from the DATA statement in the program, it checks to see if it is 9999. If not, the computer reads the number as part of the data to be averaged. If the number is 9999 the computer goes on to the rest of the program.

An average is computed by dividing the sum of the numbers by the number of numbers. The program must give the computer a way to ascertain both these quantities. Use S to stand for the sum of

the numbers and N for the number of numbers. When the program is executed, you do not know what these values will be, so set them equal to 0. The computer will then develop their values as it reads numbers from the DATA statements.

Begin the program by setting up the initial values of S and N.

```
100 LET S=0
110 LET N=0
```

Next you can program the computer to read a number from the DATA statement and check for the flag value.

```
120 READ X
130 IF X=9999 THEN _____(Compute average)
```

Use the method introduced in the previous example of leaving a blank line number in the conditional transfer statement until you know what that line number should be. In this case, if the assertion X = 9999 is true, then the computer is signaled that all the numbers in the DATA statement have been processed and that it can now compute the average. If the assertion is false, the computer reads the number as part of the data and processes that number as follows:

```
140 LET S=S+X
150 LET N=N+1
```

In line 140, the value of X (the number just read) is added to the value in S. Remember that the sum of all the numbers to be averaged is being developed in S. In line 150, the number in N is increased by 1 to record the fact that another number has been processed. You are now ready to program the computer to process the next number. This statement does the job.

```
160 GOTO 120
```

Now you can fill in the missing number in line 130, since the next line number in the program would normally be 170. Line 170 is an instruction to compute the average, which you can identify by A. If a typical DATA statement is included, the complete program is

```
100 LET S=0
110 LET N=0
120 READ X
130 IF X=9999 THEN 170
140 LET S=S+X
150 LET N=N+1
160 GOTO 120
170 LET A=S/N
180 PRINT A
190 DATA 4,2,3,6,5,9999
200 END
```

Of course, you can have as many DATA statments as needed to accommodate the numbers to be averaged. Following the last number in the last DATA statement we put the flag 9999 to mark the end of the data. This gets the computer out of the READ loop and signals it to compute the average.

The conditional transfer statement, coupled with the flag variable, is a powerful programming tool.

6—5 *FINDING ERRORS IN PROGRAMS*

The ability to look at a program and determine whether it will accomplish what it is supposed to is certainly one of the most important skills a beginner can acquire. Probably more to the point, programmers need the ability to find out what is wrong and correct it when a program is not working as it should. Although the task seems difficult, it is really easy.

Two separate skills are involved in troubleshooting programs. First, you need to decide which variables you would like to see additional information about. You can insert PRINT statements into the program to get the computer to print the values of the variables you want to see. Second, you need to be able to follow the logic of the program by going through the program as the computer would (using pencil and paper, if necessary). For loops, it is generally sufficient to check the first couple of values and the last couple of values. These two abilities together allow you to find logical programming errors quickly.

6-6 PROBLEMS

1. Write a BASIC program that calls for the input of two numbers and prints out the larger.

2. Program a computer to read three numbers from a DATA statement and then print out the smallest.

3. Write a program that has the computer find and print out the sum of all the whole numbers between 1 and 100, inclusive.

4. What will happen if you run the following program?

```
100 LET S=0
110 LET X=1
120 LET S=S+X
130 LET X=X+2
140 IF X<100 THEN 120 150 PRINT S
160 END
```

5. In example 3 in this chapter, change line 190 as follows:

```
190 DATA 4,2,3,6,5,1111
```

   Study the program with this change and write down what will be output if the program is executed. You may wish to run the program to see if you are correct. If the answer is not correct, it may be helpful to insert line 155 as follows:

```
155 PRINT "S= ";S
```

   List the program before you run it again. Compare the values of X in the DATA statement with the values of S printed out. If you can figure out why S = 1131 in the last output, you will know the reason for the error. If not, try tracing the logic of the program by doing exactly what the computer would do.

6. Program a computer to find the average of all the positive numbers in a list whose end is marked with the flag 999. You do not know in advance what numbers will be part of the data; they will be typed into the computer when the program is run.

7. Suppose you are given a DATA statement that contains a list of numbers of unknown length. However, the end of the list is marked with the flag variable 9999. Write a BASIC program to compute and print out the sum of the numbers in the list between −10 and +10 inclusive.

8. Usually the markup of supermarket items depends on the unit cost of the item. Suppose this markup is based on the following schedule:

Unit Cost	Mark up
0 to $1.00	20%
$1.01 to $2.00	10%
over $2.00	5%

The unit cost is determined by dividing the case price by the number of items in the case. Write a program to compute label price, which is unit cost plus markup.

9. Suppose you agree to work for one cent the first day, two cents the second, four cents the third, eight cents the fourth and so on. If there are 22 working days in a month, write a program that will compute your wages (in dollars) for one month.

10. Consider the series

$$1 + 1/2 + 1/3 + 1/4 + \ldots$$

Write a program to find the sum of the first N terms. Use this to find the sum of the first 10, 100, and 1000 terms. What do you think will happen if you let the series run on forever?

11. Study the following program. Can you describe what the program does?

```
100 READ N
110 LET L=1
120 LET C=1
130 READ X
140 LET C=C+1
150 IF X<L THEN 170
160 LET L=X
170 IF C<N THEN 130
```

```
180 PRINT L
190 DATA 10
200 DATA 5,83,17,3,47
210 DATA 25,16,41,51,7
220 END
```

You can find out more about on how the program works by inserting line 165 as follows

```
165 PRINT "L IS ";L
```

and running the program.

12. The following program is intended to find the average of N numbers typed in at the terminal. As it stands the program is incorrect. What's wrong?

```
100 PRINT "HOW MANY NUMBERS"
110 INPUT N
120 LET S=0
130 LET C=1
140 PRINT "TYPE IN A NUMBER";
150 INPUT X
160 LET S=S+X
170 LET C=C+1
180 IF C<N THEN 140
190 LET A=S/N
200 PRINT "THE AVERAGE IS"; A
210 END
```

13. The discounted price of an item can be computed by

$$D = L*(1 - R/100)$$

where L is the purchase price and R is the discount rate in percent. Write a program that will produce the following output:

```
LIST PRICE ($)?  (You type in price)
DISCOUNT RATE (%)?  (You type in rate)
DISCOUNTED PRICE IS
(Computer types out price) DOLLARS
```

14. There is an interesting sequence of numbers called the Fibonacci numbers. The set begins with 0, 1. Then each succeeding number in the sequence is the sum of the two previous ones. Thus, the Fibonacci sequence is

    0, 1, 1, 2, 3, 5, 8,....

    Write a BASIC program to compute and print out the first twenty numbers in the Fibonacci sequence.

15. Write a program to accept the input of two numbers. If both the numbers are greater than or equal to 10, print out their sum. If both the numbers are less than 10, print out their product. If one number is greater than or equal to 10 and the other is less than 10, print out the difference between the largest and smallest.

16. An instructor decides to award letter grades on an examination as follows:

90−100	A
80−89	B
60−79	C
50−59	D
0−50	F

    Write a program to produce the following output:

    INPUT EXAM GRADE ?(You type in numerical grade)
    YOUR GRADE IS (Computer types out A, B, C, D, E, or F)

17. If you use 8 percent more electricity each year, in nine years your consumption will double. Thus your doubling time is nine years. There is an interesting rule called the "rule of seventy- two" that can be used to compute doubling times. If a quantity grows by R percent in a single period of time, then the number of periods for the quantity to double is given approximately by 72/R. We can compute the growth of a process directly on the computer. In a single growth period, a quantity Q grows according to the relation

    $$Q_{new} = Q_{old}(1 + R/100)$$

Thus we can keep track of the growth by repeated use of the relation above. When Q is twice the original value, the corresponding number of growth periods is the doubling time. Using this approach, write a program that will produce the following output:

```
GROWTH RATE (%) ?  (You type in R)
NUMBER OF GROWTH PERIODS TO DOUBLE IS
(Computer types answer)
```

Use the program to check out the accuracy of the rule of seventy-two for many different growth rates.

18. A set of integers (whole numbers) is chosen at random from the set 1, 2, 3, 4 and put in a DATA statement. The end of the set is marked with the flag 9999. Write a BASIC program that will compute and print out the number of 1s, 2s, 3s, and 4s in the set. Test your program on the following DATA statement:

```
DATA 3,1,2,1,4,4,1,2,2,2,3,9999
```

19. Write a program that draws short lines (10 units) to the left, right, up, or down depending on which of the keys L, R, U, or D is pressed. You should model your program after the program in step 15 of the discovery exercises. Be sure to delete lines 190 through 210 in that program since they draw lines in the background color. You will need a DRAWTO statement at the appropriate place. Try running the program to see how the program draws various lines and figures.

6–7 PRACTICE TEST

1. What will be output if you run the following program?

```
100 LET Y=3
110 LET X=2*Y
120 PRINT X
130 LET Y=Y+2
140 IF Y<=10 THEN 110
150 END
```

2. What will be output if you run the following program?

```
100 READ X
110 DATA 1,2,3
120 IF X<2 THEN 160
130 IF X=2 THEN 150
140 PRINT "GOOD"
150 PRINT "BETTER"
160 PRINT "BEST"
170 PRINT
180 GOTO 100
190 END
```

3. Suppose you decide to buy a number of widgets. The manufacturer is pushing sales and will reduce prices if widgets are purchased in quantity. The price reductions are given below:

#Purchased	Price per Widget
20 or less	$2.00
21 to 50	1.80
51 or more	1.50

Write a program that will produce the following output when executed:

```
HOW MANY WIDGETS ?  (You type in purchase quantity)
PRICE PER WIDGET IS (Computer types out unit price)
TOTAL COST OF ORDER IS (Computer types out total)
```

Then keep looping back through the program.

4. Write a program that will print out the number pattern shown below and then stop.

```
0  5  10
15 20 25
```

etc.

```
165 170 175
```

5. If you get a ticket for speeding, your fine is based on how much you exceeded the speed limit. Suppose the fine is computed as follows:

Amount over Limit	Fine
1–10 mi/h	$ 5
11–20	10
21–30	20
31–40	40
41 or more	80

Write a BASIC program that will produce the following output:

```
WHAT WAS THE SPEED LIMIT ?  (You type in)
SPEED ARRESTED AT ?  (You type in)
FINE IS (Computer types out fine) DOLLARS
```

CHAPTER 7

LOOPING AND FUNCTIONS

7—1 OBJECTIVES

Using Built-in Looping Statements

You have already learned how to loop programs using either the unconditional or conditional transfer statements. Now you will learn special BASIC statements that take care of looping automatically. These statements simplify the programming task and provide flexibility in programs.

Using Built-in Functions

BASIC contains a number of built-in functions that can be used to perform specific tasks. You will learn to use some of the simpler of these functions involving numerical computations.

Working with Program Examples

You will continue with activities designed to draw you into programming. Remember that the overall objective of the book is to teach you how to write BASIC language programs.

7–2 DISCOVERY EXERCISES

1. Turn on the computer, bring up ATARI BASIC, and enter the following program:

```
100 LET Y=10
110 PRINT Y,
120 LET Y=Y+5
130 IF Y<=50 THEN 110
140 END
```

Study the program and then execute it. Record what happened.

Which statement in the program determines the difference in the numbers that were typed out?

2. Clear out the program in memory and enter the following program:

```
100 FOR Y=10 TO 50 STEP 5
110 PRINT Y,
120 NEXT Y
130 END
```

Run the program and record what happened.

Compare the output to that obtained from the program in step 1.

3. Since the two programs produce the same output, it is reasonable to assume that the statements must be related in some way. Type

```
100 FOR Y=10 TO 50 STEP 10
```

Display the program in memory and study it. What do you think will happen if you run this program?

See if you were right. Run the program and record the results.

4. Now try out a few different ideas. Type

```
100 FOR Y=0 TO 5 STEP 1
```

Display the program. What do you think this program will do?

Run the program and record the output.

5. Now type

```
100 FOR Y=0 TO 5
```

Display the program. What do you think this program will do?

Run the program and record the output.

Now compare line 100 in the program just executed with line 100 in the program in step 4. If the difference between the numbers to be printed out is 1, is the STEP part of the statement necessary?

6. Let's try a different tactic. Type

```
100 FOR Y=20 TO 10 STEP -2
```

Display the program and study it. What do you think this program will do?

Run the program and record the output.

7. Now type

```
100 FOR Y=10 TO 20 STEP -2
```

Display the program. What do you think will happen now if you run the program?

Run the program and record the output.

We have led you into a potential trap in BASIC. What seems to be the problem?

8. So far the step sizes in the FOR NEXT statements have worked out even. Let's try a new step size that might not come out even given the limits in the FOR NEXT statement. Type

```
100 FOR Y=2 TO 9 STEP 3
```

Display the program. What do you think will be printed out?

Run the program and record the output.

9. Clear the program in memory and enter the following program:

```
100 FOR X=1 TO 3
110 FOR Y=1 TO 4
120 PRINT X,Y
130 NEXT Y
140 NEXT X
150 END
```

Run the program and record the output.

10. Now type

```
100 FOR X=1 TO 2
```

Run this new program and record the output.

Compare the two number patterns you have obtained. Can you see the connection between the patterns and the limits in the FOR NEXT statements?

11. Let's modify the program a bit more. Type

```
100 FOR X=1 TO 3
110 FOR Y=1 TO 2
```

Display this program and study it. What do you think will be the output if you run it?

Try it and see if you were right.

12. Type

```
100 FOR X=1 TO 2
110 FOR Y=1 TO 2
```

Display the program and write down what you think will be typed out when you run the program.

Run the program and record the results.

Obtain a listing of the program just executed and draw a line from the line number of the FOR Y statement to the line number of the NEXT Y statement. Do the same thing for the FOR X and the NEXT X statements. Do the lines cross?

13. Now type

```
100 FOR Y=1 TO 2
110 FOR X=1 TO 2
```

Display the program. What do you think will be output of this program?

Run the program and record the output.

Obtain a listing of this program. Connect the FOR Y and the NEXT Y line numbers with a line just as you did in step 12. Do the same thing for the FOR X and the NEXT X statements. Do the lines cross? Compare with the situation in step 12.

Does this suggest a way to avoid getting into trouble using more than one FOR NEXT combination in a single program?

14. Clear the program in memory. Enter the program below.

```
100 GRAPHICS 8
110 FOR I=1 TO 150 STEP 5
120 FOR J=1 TO 0 STEP -1
130 COLOR J
140 PLOT I,I:DRAWTO I+20,I
150 DRAWTO I+20,I+20
160 DRAWTO I,I+20:DRAWTO I,I
170 NEXT J
180 NEXT I
190 END
```

Study the program. What shape do you think will be drawn by lines 140 to 160?

Run the program. Were you right?

What two colors are used in this program?

15. Exit the graphics mode by typing GRAPHICS 0 Clear the program in memory and enter the program below.

```
100 INPUT A
110 LET B=SQR(A)
120 PRINT B
130 GOTO 100
140 END
```

Run the program and at the INPUT prompt, type in 4. What happened?

Type in 9 at the INPUT prompt and record the results.

Type in 25. What happened?

Finally, type in 10. What happened?

What happens to A in the expression SQR(A) in line 110 of the program? In other words, what does SQR do?

16. Jump the computer out of the input loop. Type

```
110 LET B=INT(A)
```

Run the program for the following values of A. In each case, record the output of the program.

A	Output
1	_____
3.4	_____
256.78	_____
0	_____
−1	_____
−2.3	_____

Examine the output you have recorded above and compare each number with the corresponding value of A that you typed in. What does the INT(A) function do?

If you had trouble understanding what was happening to the negative values of A, don't worry at this point. We will review this completely later.

17. Jump the computer out of the input loop. Type

```
110 LET B=SGN(A)
```

Display the program and review the program structure to refresh your memory about how the program works. Run the program for each of the following values of A. In each case, record the output.

A	Output
1.5	_____
43	_____
128.3	_____
0	_____
−1	_____
−1.2	_____
−345.7	_____
4.7	_____
−5.8	_____

Examine the output carefully. What does the SGN function do?

_____

18. Jump the computer out of the input loop. Type

```
110 LET B=ABS(A)
```

Execute the program for each of the values of A given below. Again, record the output in each case.

A	Output
3.4	_____
0	_____
−3.4	_____
−2	_____
−8.45	_____
8.45	_____

Examine the output. What does the ABS function do?

19. This concludes the computer work for now. Jump the computer out of the input loop and turn off the computer.

7–3 DISCUSSION

Using Built-in Looping Statements

In the previous chapters you learned how to loop programs under the control of transfer statements. The unconditional (GOTO) statement was useful but could sometimes result in a loop with no way out. The conditional (IF THEN) statement provided a way to loop the program and also a way to get out of the loop. All of these are good techniques. However, BASIC gives programmers a simple and elegant way to take care of looping. We will now go over this new method, which uses the FOR NEXT statements.

All FOR statements have the same format. This format and a typical statement are shown below.

Line# **FOR**<variable> = <relation>**TO**<relation> **STEP**<relation>

```
120 FOR X=1 TO 9 STEP 2
```

The things that can change in FOR statements are the variable and the three relations. If the STEP is left out of the statement, the

computer will use a step size of 1. There are many different forms of the FOR statement. A few of the possibilities are:

```
130 FOR J=2 TO 8
130 FOR T=25 TO 10 STEP -2
130 FOR W=-20 TO 10 STEP 2
130 FOR X=3*Z TO A*B STEP D
```

In general, you can use any legal BASIC statement for the relations if the variables are properly defined in the program.

■ **Use FOR NEXT statements for looping.**

The FOR statement opens a loop. You close the loop with the NEXT statement. The following example shows how this is done.

```
200 FOR X=2 TO 18 STEP 2 (Opens loop)
.
.
.
Program lines inside loop
.
.
.
340 NEXT X (Closes loop)
```

In the NEXT statement, the variable must be the same as that in the FOR statement that opened the loop.

It is important to acquire a complete understanding of how these loops work. In the example above, when the program reaches line 200 the first time, X is set equal to 2. Then the computer works through the lines until it reaches line 340, which closes the loop and directs the computer back to line 200 and the next value of X (in this case, 4). The computer stays in the loop until the value of X exceeds the limit of 18. Then, instead of going through the statements inside the loop, the computer jumps to the line following line 340.

Let's look at another example of the FOR NEXT statements in action.

```
100 LET A=1
110 FOR X=1 TO 6 STEP 2
120 LET A=2*A
130 PRINT A, X
140 NEXT X
150 END
```

The table below shows the line numbers in the order the computer encounters them and gives the corresponding values of the variables at each stage.

Line Number	A	X
100	1	
110	1	1
120	2	1
130	2	1
140	2	1
110	2	3
120	4	3
130	4	3
140	4	3
110	4	5
120	8	5
130	8	5
140	8	5
110	8	7*
150	**	

* Jumps out of loop

** Program stops

Study the sequence of line numbers and the corresponding values of A and X until you are certain that you understand how the FOR NEXT statements control the loop.

Quite often a program requires more complicated loop structures. The structure can be as involved as desired provided that the loops do not cross. The example below illustrates a segment of a program with crossed loops.

```
100 FOR A=2 TO 20
110 FOR B=4 TO 8
    Loops cross!
240 NEXT A
250 NEXT B
```

Another example of crossed loops is

```
100 FOR I=0 TO 20 STEP 2
110 FOR A=10 TO 2 STEP -1
120 FOR B=1 TO 4
    Outer loop OK; inner loops cross!
170 NEXT A
180 NEXT B
190 NEXT I
```

With large complex programs containing many FOR NEXT loops, it is easy to cross one or more of the loops. However, when this does happen, the error message will point you quickly to it.

The following example illustrates a complicated structure in which the loops are organized correctly:

```
100 FOR X=1 TO 10
110 FOR Y=2 TO 4
.
.
140 NEXT Y
.
.
170 FOR Z=1 TO 5
.
.
210 FOR K=20 TO 10 STEP -2
.
.
270 NEXT K
.
.
310 NEXT Z
.
.
410 NEXT X
```

In this example we have double loops and loops within loops. Remember, any combination of loops may be used in a program provided that lines connecting the FOR statements and their corresponding NEXT statements do not cross. If they do, the computer will signal an error and stop.

■ **Don't cross your FOR NEXT loops!**

Using Built–in Functions

Since many computing tasks are needed routinely, ATARI BASIC has some tasks preprogrammed in the form of functions. With these built-in functions, the programmer can perform very complicated mathematical operations without difficulty. The functions include the following:

Function	Action
SQR(X)	Square root of X
INT(X)	Integer part of X
SGN(X)	Sign of X
ABS(X)	Absolute value of X

Let's examine the first function, SQR(X), to see how all the functions operate in general. First, X is called the argument of the function and can be thought of as "what the function works on." If you use SQR(X) in a program, you are instructing the computer to look up the value of X and take the square root of that number. For example,

```
SQR(36) = 6
SQR(64) = 8
SQR(81) = 9
SQR(2)  = 1.41421356
```

and so on. The only limitation is that you can't take the square root of a negative number. If you asked the computer to evaluate SQR(−6), for example, it would signal an error and stop.

The argument of the function can be as complicated as needed in the program. If the computer runs across an expression such as

SQR(X+4*Y)

it will look up the values of the variables, carry out the calculation indicated, and take the square root of the result. This characteristic is true for all the functions.

INT(X) takes the integer part of X. The term integer means "whole number." Thus, 2 is an integer while 23.472 is not. To take the integer part of a number, you simply forget about everything following the decimal point. Thus

INT(3.1593) = 3

INT(54.76) = 54

INT(0.362) = 0

However, negative numbers require special attention. What really happens when you take the integer of a number is that you go to the first integer less than or equal to the number. Using this rule,

INT(−2) = −2

INT(−.93) = −1

and so on. Note carefully that the INT function does not round off a number. Often beginners are somewhat confused about this.

- **The integer part of a number is the first integer less than the number.**

SGN(X) is a very interesting function. If X is positive, SGN(X) is +1. If X is negative, SGN(X) is −1. If X is 0, SGN(X) is 0. In effect, SGN(X) returns the sign of X, either +1, −1, or 0. Therefore,

SGN(4.568) = +1

SGN(375) = +1

SGN(0) = 0

SGN(−5.93) = −1

SGN(−4) = −1

At this point it may not be clear how such a function could be useful. The SGN function is very useful, however, and has many applications. For the time being, it is enough simply to learn what the function does.

ABS(X) tells the computer to ignore the sign of X. In effect, it converts all values of X, other than 0, to positive numbers. For example:

ABS(4.5) = 4.5
ABS(−4.5) = 4.5
ABS(95.34) = 95.34
ABS(−95.34) = 95.34
ABS(0) = 0

There are many other built-in functions in BASIC. However, most of them involve more mathematical knowledge than many students have. If you know the mathematics necessary to understand what the functions are doing, you will have no difficulty learning how to use them. If you are interested, consult Chapter 6 of the *ATARI BASIC Reference Manual*. We will take up some functions that involve strings of characters in Chapter 9.

The built-in functions we have been discussing are used in BASIC statements. Lines that use such functions might include

```
100 LET X=SQR(Y)
100 LET Z=3*INT(C)+ABS(D)
```

The built-in functions can also be used within the argument of functions such as

```
100 LET Y=INT(SQR(X)+3*ABS(Z))
```

In this example, the computer would add the square root of X to the absolute value of Z multiplied by 3 and express the sum as an integer.

- **Any BASIC expression can be the argument of BASIC function**

7—4 PROGRAM EXAMPLES

Example 1 - Finding the Average of a Group of Numbers

In the previous chapter, you found an average in one of the program examples. Let's return to the same problem but use a different method. We want the program to produce the following printout:

```
HOW MANY NUMBERS (You type in)
ENTER NUMBERS, ONE AT A TIME
?  (You type in the numbers)
THE AVERAGE IS (Computer types out average)
```

The first few lines should be easy for you to write by now.

```
100 PRINT "HOW MANY NUMBERS";
110 INPUT N
120 PRINT "ENTER NUMBERS, ONE AT A TIME"
```

Now you must arrange for the input of N numbers but must also keep in mind that we are supposed to compute the average of the numbers. So initially set S (which will be used to sum the numbers) equal to 0.

```
130 LET S=0
```

FOR NEXT statements are ideal for inputting numbers and summing them.

```
140 FOR I=1 TO N
150 INPUT X
160 LET S=S+X
170 NEXT I
```

You don't use I, the loop variable, except to count the numbers as they are input. When all the numbers are in, the computer will jump out of the loop to the next line after 170. When this happens, S will contain the sum of all the values of X that were typed in. Since you know that N is the number of numbers typed in, you can immediately get the program to compute the average.

```
180 LET A=S/N
```

The rest of the program follows without difficulty.

```
190 PRINT "THE AVERAGE IS ";A
200 END
```

The complete program is

```
100 PRINT "HOW MANY NUMBERS ";
110 INPUT N
120 PRINT "ENTER NUMBERS, ONE AT A TIME"
130 LET S=0
140 FOR I=1 TO N
150 INPUT X
160 LET S=S+X
170 NEXT I
180 LET A=S/N
190 PRINT "THE AVERAGE IS ";A
200 END
```

Example 2 - Temperature Conversion Table

In one of the earlier programs you used the relation

$$C = 5/9*(F-32)$$

to convert from degrees Fahrenheit to degrees Celsius. Now let's generate a conversion table as follows:

Degrees F	Degrees C
0	−17.77777777
5	−15
10	−12.22222222
etc.	
100	37.77777777

First you'll want the column headings and a space before the table begins.

```
100 PRINT "DEGREES F","DEGREES C"
110 PRINT
```

You can use a FOR NEXT loop to generate the values of F, which can then be converted to C and printed out.

```
120 FOR F=0 TO 100 STEP 5
130 LET C=5*(F-32)/9
140 PRINT F,C
150 NEXT F
```

Finally you need the END statement.

```
160 END
```

The whole program is

```
100 PRINT "DEGREES F","DEGREES C"
110 PRINT
120 FOR F=0 TO 100 STEP 5
130 LET C=5*(F-32)/9
140 PRINT F,C
150 NEXT F
160 END
```

Example 3 - Exact Division

Now let's write a program that will compute all the integers (whole numbers) that divide exactly into another integer. To illustrate, suppose we take N as the test integer. The problem is to find all the integers (X) that will divide exactly into N with no remainder. The rule to use is

If N/X = INT(N/X) then there is no remainder
If N/X <> INT(N/X) then there is a remainder

Now write a program to produce the following output when executed:

```
INPUT A POSITIVE WHOLE NUMBER? (You type in)
THE EXACT DIVISORS ARE
```
(Computer types out first number, second number, etc.)

The program begins easily.

```
100 PRINT "INPUT A POSITIVE WHOLE NUMBER";
110 INPUT N
120 PRINT "THE EXACT DIVISORS ARE"
```

Now you want the program to look at each of the whole numbers between 1 and N. Of course, this is an ideal use of the FOR NEXT loop. The rule given above tests whether each number divides exactly.

```
130 FOR X=1 TO N
140 IF N/X<>INT(N/X) THEN 160
150 PRINT X,
160 NEXT X
```

Finally you need the END statement.

```
170 END
```

The complete program is

```
100 PRINT "INPUT A POSITIVE WHOLE NUMBER "
110 INPUT N
120 PRINT "THE EXACT DIVISORS ARE"
130 FOR X=1 TO N
140 IF N/X<>INT(N/X) THEN 160
150 PRINT X,
160 NEXT X
170 END
```

Try the program using fairly large values of N. How could you make the program run in half the time?

Example 4 - Depreciation Schedule

When a company invests in equipment, the investment is depreciated over a number of years for tax purposes. This means that the value of the equipment decreases each year and the amount of decrease is a tax-deductible item. One of the methods used to compute depreciation is the "sum-of-the-years'-digits" schedule.

To illustrate, suppose a piece of equipment has a lifetime of 5 years. The sum of the years' digits would be

$$1 + 2 + 3 + 4 + 5 = 15$$

The depreciation the first year will be 5/15 of the initial value. The depreciation fraction the second year will be 4/15, and so on. If the equipment had an initial value of $3000, the depreciation schedule would be

End of Year	Depreciation Fraction	Depreciation	Current Value
1	5/15	1000	2000
2	4/15	800	1200
3	3/15	600	600
4	2/15	400	200
5	1/15	200	0

The problem is to write a BASIC program that will generate depreciation schedules by the "sum-of-the-years'-digits" method. The output should be as follows:

```
THE INITIAL ASSET VALUE IS (You type in)
THE ASSET LIFE IN YEARS IS (You type in)
END OF      DEPREC DEPREC     CURRENT
 YEAR     FRACTION                ASSET VALUE
(Computer prints out table)
```

The first few lines of the program can be written without any explanation:

```
100 PRINT "THE INITIAL ASSET VALUE IS ";
110 INPUT P
120 PRINT "THE ASSET LIFE IN YEARS IS ";
130 INPUT N
140 PRINT
150 PRINT "END OF";"    ";"DEPREC";"
";"DEPREC";"    ";"CURRENT"
160 PRINT " YEAR";" ";"FRACTION";"
          ";"ASSET VALUE"
170 PRINT
```

Next, compute the sum-of-the-years' digits.

```
180 LET S=0
190 FOR I=1 TO N
200 LET S=S+I
210 NEXT I
```

Now compute the schedule and print it out. Use the variable P1 to keep track of the current asset value.

```
220 LET P1=P
230 FOR I=1 TO N
240 LET F=(N+1-I)/S
250 LET D=P*F
260 LET P1=P1-D
270 PRINT I,F,D,P1
280 NEXT I
```

In line 240, F is the depreciation fraction for the Ith year. You can check this out for various values of I to ensure that the expression does generate the correct value of F. In line 250, D is the depreciation. The only thing missing now is the END statement.

```
290 END
```

The complete program is

```
100 PRINT "THE INITIAL ASSET VALUE IS ";
110 INPUT P
120 PRINT "THE ASSET LIFE IN YEARS IS ";
130 INPUT N
140 PRINT
150 PRINT "END OF";"   ";"DEPREC";"
";"DEPREC";"   ";"CURRENT"
160 PRINT " YEAR";" ";"FRACTION";"
        ";"ASSET VALUE"
170 PRINT
180 LET S=0
190 FOR I=1 TO N
200 LET S=S+I
210 NEXT I
220 LET P1=P
```

```
230 FOR I=1 TO N
240 LET F=(N+1-I)/S
250 LET D=P*F
260 LET P1=P1-D
270 PRINT I,F,D,P1
280 NEXT I
290 END
```

Try out the program with different inputs. Use $1000 and 4 years first. Other choices will generally give long fractional values that will wrap around on the screen and be difficult to read. You can make the table more legible by reducing the number of digits displayed. See example 2, Chapter 10. You may be able to come up with a solution yourself using the INT function covered in this chapter.

Use this program to impress the Internal Revenue Service with computer-generated depreciation schedules!

7—5 PROBLEMS

1. Write a program to generate a table of numbers and their square roots. The table should look like the following:

```
N            SQR(N)
2           1.41421356
2.1         1.44913767
2.2         1.48323969
etc.
3.9         1.97484176
4           2
```

2. Write a program to count from 0 to 500 by tens and print out the results.
3. Write a program to accept the input of a number N, then print out the even numbers greater than 0 but less than or equal to N.
4. Write a program to print out a conversion table from inches to centimeters. Include the appropriate headings. Start the table at 0 and continue to 10 inches in steps of 0.5 inch. There are 2.54 centimeters in one inch.

5. What will be printed out if you run the following program?

```
100 FOR X=5 TO 1 STEP -1
110 PRINT "ABCD";
120 NEXT X
130 END
```

6. Study the following program. What will be output?

```
100 FOR I=1 TO 5
110 READ A
120 LET B=INT(A)-SGN(A)*2
130 PRINT B
140 NEXT I
150 DATA 2.2,-3,10,0,-1.5
160 END
```

7. What will be printed out if you run the following program?

```
100 FOR X=1 TO 10
120 LET Y=2*X
130 FOR Z=1 TO 5
140 LET U=Z+Y
150 FOR V=1 TO 3
160 PRINT V+U
170 NEXT Z
180 NEXT V
190 NEXT X
200 END
```

8. The following program won't work. What's wrong?

```
100 FOR X=-10 TO +10 STEP 2
110 PRINT X, SQR(X)
120 NEXT X
130 END
```

9. What does the following program do?

```
100 FOR X=1 TO 5
110 READ Y
120 LET Z=INT(100*Y+.5)/100
130 PRINT Z
140 NEXT X
150 DATA 1.06142,27.5292,138.021
160 DATA .423715,51.9132
170 END
```

10. Write a program to print out the following pattern of asterisks without using more than three PRINT statements.

```
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

11. Write a graphics program that draws a grid on the screen (a set of evenly spaced horizontal and vertical lines).

12. N! is read "N factorial" and means the product of all the integers from 1 to N inclusive. For example,

    3! = (1)(2)(3) = 6
    5! = (1)(2)(3)(4)(5) = 120

    and so on. Write a program calls for the input of N and then computes and prints out N!

13. Write a BASIC program that calls for the input of N grades and computes and prints out (1) the highest grade, (2) the lowest grade, and (3) the average of the grades.

14. What, if anything, is wrong with the following program?

```
100 FOR X=1 TO 2
110 FOR Y=2 TO 6
120 PRINT X+Y
130 NEXT Y
140 FOR Z=1 TO 3
150 PRINT X+Z
```

```
160 NEXT X
170 NEXT Z
180 END
```

15. What will be output if you run the following program?

```
100 FOR X=1 TO 4
110 FOR Y=1 TO 3
120 LET Z=X*Y
130 PRINT Z,
140 NEXT Y
150 PRINT
160 NEXT X
170 END
```

16. Suppose you decide to invest $1000 on the first of each year for ten years at an annual interest rate of 6 percent. At the end of the tenth year, the value of the investment will be $13,971.64. To see how this could be computed, use the following formula:

$$\$NEW = (\$OLD + I)(1 + R/100).$$

In this formula, R is the annual interest rate in percentage. I is the annual investment, $OLD is the value of the investment at the beginning of each year, and $NEW is the value of the investment at the end of the year. Thus, $NEW becomes $OLD for the next year. Write a BASIC program that will produce the following output.

```
WHAT IS THE ANNUAL INVESTMENT? (You type in)
THE ANNUAL INTEREST RATE (%)?  (You type in)
HOW MANY YEARS? (You type in)
AT THE END OF THE LAST YEAR THE VALUE OF
THE INVESTMENT WILL BE (Computer types answer)
```

17. The DATA statements below contain the time worked by a number of employees during a one–week period.

```
190 DATA 5
200 DATA 2, 4.8, 8, 10, 8, 7, 10
201 DATA 5, 3.75, 7, 8, 8, 6, 10
202 DATA 1, 3.25, 8, 10, 6, 8, 8
203 DATA 4, 5, 8, 10, 6, 10, 6
204 DATA 3, 4.25, 6, 6, 8, 10, 7
```

The number in line 190 gives the number of employees to follow. Each of the DATA lines after line 190 contains a weekly record for one employee. The data are the employee number, the hourly rate, and the hours worked Monday through Friday. The employee receives time and a half for everything over 40 hours per week. Write a BASIC program using these DATA statements to compute and print out the employee number and the gross pay for the week for each of the employees.

18. Assume that the following DATA statements give the performance of the students in an English class on three examinations:

```
190 DATA 6
200 DATA 3, 90, 85, 92
201 DATA 1, 75, 80, 71
202 DATA 6, 100, 82, 81
203 DATA 5, 40, 55, 43
204 DATA 2, 60, 71, 68
205 DATA 4, 38, 47, 42
```

The number in line 190 is the number of students in the class. Each of the DATA statements that follow gives the performance for a single student. The information is the student ID number, grade 1, grade 2, and grade 3. Thus, as shown in line 202, student 6 got examination grades of 100, 82, and 81. Write a program using these DATA statements to compute and print out each student's ID number and his or her course grade. Assume that the first two examination grades are weighted 25 percent each toward the overall grade and the last grade is weighted 50 percent.

7—6 PRACTICE TEST

1. What will be printed if you run the following program?

```
100 FOR Y=20 TO 1 STEP -2
110 PRINT Y,
120 NEXT Y
130 END
```

2. What will be printed out if you run the following program?

```
100 FOR A=1 TO 4
110 FOR B=1 TO 3
120 PRINT A*B,
130 NEXT B
140 NEXT A
150 END
```

3. Fill in the blanks.

a. SQR(36) = _____
b. INT(7.13) = _____
c. ABS(−22.8) = _____
d. SGN(−1.3) = _____

4. What, if anything, is wrong with the following program?

```
100 FOR I=1 TO 5
110 FOR J=2 TO 5
120 PRINT I, J
130 NEXT I
140 NEXT J
150 END
```

5. Miles can be converted to kilometers by multiplying the number of miles by 1.609. Write a program to produce the following output:

```
MILES       KILOMETERS
-----       ----------
10          16.09
15          24.135
20          32.18

etc.

100         160.9
```

6. Numerical information is loaded into DATA statements as follows:

```
100 DATA 10
110 DATA 25,21,24,21,26,27,25,24,23,24
```

The number in line 100 gives the number of numbers to be processed in the rest of the DATA statements. Write a program using these statements to compute the average of the numbers excluding the one in line 100.

7. Study the following program.

```
100 GRAPHICS 8
110 COLOR 1
120 FOR VERT=0 TO 120 STEP 40
130 FOR HRZ=0 TO 240 STEP 40
140 PLOT HRZ,VERT:DRAWTO HRZ,VERT+15
150 DRAWTO HRZ+10,VERT+15
160 DRAWTO HRZ,VERT
170 NEXT HRZ
180 NEXT VERT
190 END
```

a. What shape will be drawn when you run the program?
b. How many copies of the shape will be drawn?
c. Are the shapes drawn across first or down first?

CHAPTER 8

WORKING WITH COLLECTIONS OF NUMBERS

8—1 OBJECTIVES

In this chapter you will apply some of the ideas you learned earlier to collections of numbers. You will be introduced to new concepts that will expand the capability of BASIC.

Learning to Use Single— and Double— Subscripted Variables

You will learn what subscripted variables are and how to use them to create more useful programs.

Saving Space for Arrays

When you want to store a collection of numbers in the computer, you must indicate how much space the numbers will occupy in the memory. You will learn how to use the DIM statement to save space in the computer's memory.

Using FOR NEXT Loops to Handle Subscripted Variables

You will learn to apply FOR NEXT loops to the repetitive process of naming numbers in a collection.

Working with Program Examples

You will study BASIC programs that take advantage of subscripted variables.

8–2 DISCOVERY EXERCISES

Subscripts

When working with groups of numbers you must be able to distinguish members of the group from one another. This is the reason for subscripts. Before learning about subscripts, however, add two important words to your computer vocabulary. You could use the word *collection* to describe a group of numbers, but two other words are more commonly used: *matrix* and *array*. For our purposes they both mean the same thing: a "collection of numbers." Remember, then, the terms *matrix* and *array* mean a collection of numbers.

■ **MATRIX and ARRAY mean collections of numbers.**

Let's look at the array below.

$$Y_1 = 9$$

$$Y_2 = 10$$

$$Y_3 = 7$$

$$Y_4 = 14$$

$$Y_5 = 12$$

$$Y_6 = 15$$

The name of the array is Y. Its size is six, since there are six elements (or numbers) in it. The numbers 9, 10, 7, 14, 12, and 15 are the elements in the array. The numbers printed to the right and slightly below the Ys are called subscripts. In BASIC, subscripts are printed in parentheses, e.g., Y(3) rather than Y_3. Each subscript merely points to one element in the array. Thus, Y(4) means the fourth number in the array, which in this case is 14. We read Y(4) as "Y sub four." The third number in the array would be called "Y sub three," and so on. This array is one-dimensional, since it takes only a single number (or subscript) to locate a given element in the array.

Now let's look at a more complicated example.

$$Z_{1,1} = 4 \quad Z_{1,2} = 9 \quad Z_{1,3} = 5$$

$$Z_{2,1} = 3 \quad Z_{2,2} = 8 \quad Z_{2,3} = 7$$

In this example, there are six elements in the array Z. However, this is a two-dimensional array, since we must specify which row and column we want. The first subscript gives the row number; the second specifies the column. $Z_{2,1}$ is read as "Z sub two one" and means the element of Z at the second row and first column. Likewise, the element at row 1, column 3 would be identified as $Z_{1,3}$ and would be read "Z sub one three."

In summary, you will work with two kinds of matrices or arrays. Elements in a one-dimensional array are located with a single number. Elements in a two-dimensional array are located with two numbers, designating a row and a column. A location in a one-dimensional array is designated by a single-subscripted variable. Likewise, the double-subscripted variable is used in the two-dimensional array. You are now ready for the computer work.

1. Turn on the computer, bring up ATARI BASIC, and enter the following program:

```
100 LET X(1)=21
110 LET X(2)=13
120 LET X(3)=16
130 LET X(4)=8
140 LET X(5)=11
150 PRINT X(1)
160 END
```

What do you think will be printed out if you run the program?

Run the program and record what happens.

Error 9 is a dimension error that will be considered in greater detail later. Type in

```
90 DIM X(5)
```

Run the program again and record what happens.

2. Now modify the program to print out the fourth value of X. Run the program. Did it work?

3. Now type

```
150 PRINT X(3)+X(4)
```

Display the program and study it briefly. What do you think will happen if you run the program?

Run the program and see if you were right. Record the output.

4. Type

```
150 FOR I=1 TO 5
152 PRINT X(I)
154 NEXT I
```

Display the program. What do you think will be printed out?

See if you were right. Record what happens when you run the program.

5. Modify this program to print out only the first three values of the array X. Record what happens.

6. Again modify the program, but this time so that the first value of the array, then every other one, will be printed out. Record what happens.

7. Clear the program in memory. Enter the following program:

```
90 DIM Y(2,3)
100 LET Y(1,1)=2
110 LET Y(1,2)=5
120 LET Y(1,3)=1
130 LET Y(2,1)=2
140 LET Y(2,2)=4
150 LET Y(2,3)=3
160 PRINT Y(1,3)
170 END
```

Display the program and make sure you have entered it correctly. What do you think this program does?

Run the program and record the output.

8. Type

```
160 PRINT Y(2,2)+Y(1,3)+Y(1,1)
```

Display the program. What will this program do?

Run the program and see if you were right.

9. Type

```
160 LET S=0
162 FOR J=1 TO 3
164 LET S=S+Y(1,J)
166 NEXT J
168 PRINT S
```

Display the program and study it carefully. What will happen if you run this program?

Run the program and record the output.

Explain in your own words what is taking place.

10. Type

```
162 FOR I=1 TO 2
164 LET S=S+Y(I,2)
166 NEXT I
```

Display the program. What is the program doing now?

Run the program and record the output.

Again try to explain in your own words what is happening.

11. Now type

```
164 FOR J=1 TO 3
166 LET S=S+Y(I,J)
168 NEXT J
170 NEXT I
172 PRINT S
180 END
```

Display the program and think a minute about it. In particular, compare what you see now to what happened in steps 9 and 10. What does this program do?

Run the program and record the output.

12. Clear the program in memory. Type the following program:

```
100 DIM X(4), Y(4)
110 FOR I=1 TO 4
120 READ A,B
130 LET X(I)=A
140 LET Y(I)=B
150 NEXT I
160 PRINT X(1)+Y(4)
170 DATA 2,1
171 DATA -1,3
172 DATA 5,6
173 DATA 2,4
180 END
```

Display the program and check to see that you have entered it correctly. Study the program carefully. If you run the program, what will be typed out?

Run the program and see whether you were right. Record the output.

13. Clear the program in memory. Type the following program:

```
100 DIM A(4,3)
110 FOR I=1 TO 4
120 FOR J=1 TO 3
130 READ T
140 LET A(I,J)=T
150 NEXT J
160 NEXT I
170 FOR I=1 TO 4
180 FOR J=1 TO 3
190 PRINT A(I,J);"   ";
200 NEXT J
210 PRINT
220 PRINT
230 NEXT I
240 DATA 1,3,1
250 DATA 4,2,5
260 DATA 1,4,2
270 DATA 3,2,5
280 END
```

Make sure that you have entered the program correctly, then take a few minutes to study it. Notice the two spaces between the quotes in line 190. Can you see what will be printed out if you run the program?

Run the program and record the output.

Compare what was printed out to the numbers in the DATA statements in the program.

14. Clear the program in memory and then enter the following program:

```
100 DIM A(2,2)
110 FOR I=1 TO 2
120 INPUT X,Y
130 LET A(I,1)=X
140 LET A(I,2)=Y
150 NEXT I
160 PRINT
170 PRINT
180 FOR I=1 TO 2
190 FOR J=1 TO 2
200 PRINT A(I,J);"   ";
210 NEXT J
220 PRINT
230 PRINT
240 NEXT I
250 END
```

Run the program and when the INPUT prompt appears, type

```
2,5
3,8
```

What happens?

Compare the output to the numbers you typed in.

15. Clear the program in memory. Then enter the following program:

```
100 DIM X(3,3)
110 FOR I=1 TO 3
120 FOR J=1 TO 3
130 READ A
140 LET X(I,J)=A
150 NEXT J
160 NEXT I
```

```
170 PRINT
180 PRINT
190 FOR I=1 TO 3
200 FOR J=1 TO 3
210 PRINT X(I,J);"   ";
220 NEXT J
230 PRINT
240 NEXT I
250 DATA 2,1,3
260 DATA 4,7,5
270 DATA 1,2,6
280 END
```

Run the program. What happens?

Compare the output to the numbers in the DATA statements.

16. This concludes the computer work for this chapter. Turn off the computer.

8–3 DISCUSSION

Most students are confused by arrays. Pay particular attention to the discussion material to clear up any questions that might have arisen in the computer work.

Learning to Use Single- and Double-Subscripted Variables

The need for subscripted numbers becomes obvious when you handle large collections of numbers. If, for example, you were writing a program that involved only four numbers, you would have no difficulty naming them. You might call the numbers X, Y, U, and V. But suppose you needed to work with 100 numbers. For this reason, it is often very useful to have subscripted numbers. Fortunately, BASIC makes provisions for subscripts.

Consider the following set of numbers:

i	Y_i
1	14
2	8
3	9
4	11
5	16
6	20
7	5
8	3

We can refer to the entire set of numbers by the single name Y. Thus, Y is a collection of numbers, a matrix, or an array— all of which mean roughly the same thing for our purposes. To locate a number in an array, we must have the array name (in this case Y) and the number's position in the array. The i column gives the number's position. Thus Y(3) (pronounced "Y sub three") locates the third number in the array Y. In this case, Y(3) has the value 9. Likewise, Y(7), is 5, Y(1) is 14, and so on. Y(i) (pronounced "Y sub i"), is a general way to denote any element of the array, depending on the value of i. In the example above, If i were 8, then Y(i) would be 3. This collection of numbers is one-dimensional, since only one number (subscript) is needed to locate any element in the array.

Next let's look at a two-dimensional array.

$Y_{i,j}$	1	2	3	4
1	3	−1	10	8
2	2	4	5	6
3	1	−2	9	3

Now you need two numbers to locate an element in the array. Given a row number and a column number, you can find any element of the array. For example, Y(1,3) means the element of Y located at row 1, column 3. In the example above, the element has the value 10. A general way to denote an element in the two-dimensional array is Y(i,j). The first subscript (i) is the row number, and the second subscript (j) is the column number.

To make sure you understand how the double subscripts are used, refer to the two-dimensional array in the table above and verify that the following statements are correct:

$$Y_{3,2} = -2$$

$$Y_{1,4} = 8$$

$$Y_{3,3} = 9$$

$$Y_{2,1} = 2$$

- **Double subscripts define row and column numbers.**

An interesting question comes up. Does X(M−N+3,S*T) mean anything? The answer is yes provided that the computer can convert M−N+3 and S*T into numbers. Even Y(Y(1,1),Y(2,3)) is all right as long as the computer can locate the numbers in Y(1,1) and Y(2,3). However, suppose you want to locate X(A+B) where A = 2.6 and B = 1.1. Thus, A+B = 3.7. But it doesn't make any sense to try to look up the 3.7th number in the array X. In this case, the computer will take only the integer in 3.7, and compute X(A+B) as X(3), the third element in the array X.

Saving Space for Arrays

The computer must know how big an array is for two reasons. First, it must allow sufficient space in memory to hold the array. Next, the computer must know the size of the array in order to carry out arithmetic operations properly.

- **Save space with a DIM statement.**

An example of a DIM (for "dimension") statement is

```
100 DIM B(5,20),Y(8),Z(34),X(3,6)
```

Four arrays are "dimensioned" in line 100. B is a two-dimensional array with five rows and twenty columns. Y is one-dimensional, with eight elements. Likewise, Z is one-dimensional, with thirty-four

elements. Finally, X is a two-dimensional array with three rows and six columns. It's a good practice to make the DIM statement the first one of the program because you can see the sizes of the arrays that will be used by glancing at the beginning of the program. However, the DIM statement must appear before any other statement that refers to arrays.

Using FOR NEXT Loops to Handle Subscripted Variables

Subscripts involve collections of numbers. Because operations with collections of numbers almost always involve repetition, it is reasonable to employ FOR NEXT statements to handle arrays. You will use FOR NEXT loops to define the subscripts used in the arrays. For example, the following program segment sets up a six by four array, then load 5s into all the elements.

```
100 DIM A(6,4)
110 FOR R=1 TO 6
120 FOR C=1 TO 4
130 LET A(R,C)=5
140 NEXT C
150 NEXT R
```

If you study this program segment, the details of the process become clear. When the computer reaches line 130 in the program for the first time, R = 1 and C = 1. Then R is held constant while C goes to 2, 3, and 4. At each step in this process, the corresponding element of the array is set equal to 5. Then R is set equal to 2, and C takes on the values 1, 2, 3, and 4. The process goes on until all the elements of the array have been set equal to 5.

Either one- or two-dimensional arrays can be handled in this fashion using subscripts. In many applications it is preferable to use FOR NEXT loops to carry out the desired operations on arrays.

An important fact about subscripted variables is that they cannot be used directly in INPUT or READ statements. Therefore, it is necessary to use non subscripted variables first and then use the LET assignment statement. For example, the program segment

```
100 DIM A(3)
110 FOR I=1 TO 3
120 INPUT X
130 LET A(I)=X
140 NEXT I
```

shows how the variable X is used to contain the value that is typed in at the keyboard. Then the assignment statement

```
100 LET A(I)=X
```

transfers the value to the subscripted variable A(I). This is done for all three values. The LET statement allows you to use the INPUT and READ statements indirectly with subscripted variables.

8–4 PROGRAM EXAMPLES

Example 1 - Examination Grades

Suppose the distribution of examination grades in a class of fifteen students is as follows:

Student Number

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Grade	67	82	94	75	48	64	89	91	74	71	65	83	72	69	72

The problem is to write a BASIC program that allows you to type in the grades. The format should be:

```
HOW MANY STUDENTS? (You type in)
STUDENT GRADE
1 (You type in grade, etc.)
2
3
(etc.)
```

The program should instruct the computer to find the class average, the highest grade, and the lowest grade, then print this information out as follows:

```
CLASS AVERAGE IS (Computer prints out average)
HIGHEST GRADE IS (Computer prints out highest grade)
LOWEST GRADE IS (Computer prints out lowest grade)
```

Let's approach the problem by steps. First, since you are going to store the student grades in subscripted form, you must include a DIM statement to save space for the array.

```
100 DIM G(50)
```

You are using the variable G to store grades and can insert up to fifty grades. Next you have a message, an input, and a space.

```
110 PRINT "HOW MANY STUDENTS ";
120 INPUT N
130 PRINT
```

Now you are ready to input the grades. First you must generate the heading for the table.

```
140 PRINT "STUDENT", "GRADE"
150 PRINT
```

A loop using FOR NEXT statements is ideal to control the input of grades.

```
160 FOR I=1 TO N
170 PRINT I,
180 INPUT A
190 LET G(I)=A
200 NEXT I
```

The student number is printed out in line 170. In line 190, the student number (I) is used as a subscript for the grade. This line generates grades in the computer in the form G(1), G(2),...,G(N). The next task is to find the average of the grades. This can be done by summing up all the grades and dividing by the number of grades.

```
210 LET S=0
220 FOR I=1 TO N
230 LET S=S+G(I)
240 NEXT I
250 PRINT
```

Now you can program the computer to find the average and print out the results.

```
260 LET M=S/N
270 PRINT "CLASS AVERAGE IS ";M
```

The final part of the program is to locate and print out the highest and lowest grades in the class. H and L will stand for the highest and lowest grades, respectively. Initially, set both H and L equal to G(1), the first grade in the list. You know that the same grade can't be the highest and lowest at the same time. Thus, you will program the computer to go through the rest of the grades, compare H and L with each grade, and make adjustments to H and L as required.

```
280 LET H=G(1)
290 LET L=G(1)
300 FOR I=2 TO N
310 IF L<G(I) THEN 330
320 LET L=G(I)
330 IF H>G(I) THEN 350
340 LET H=G(I)
350 NEXT I
```

The required printout can be obtained with two lines.

```
360 PRINT "HIGHEST GRADE IS ";H
370 PRINT "LOWEST GRADE IS ";L
```

Finally the END statement completes the program.

```
380 END
```

The complete program follows:

```
100 DIM G(50)
110 PRINT "HOW MANY STUDENTS ";
120 INPUT N
130 PRINT
140 PRINT "STUDENT", "GRADE"
150 PRINT
160 FOR I=1 TO N
170 PRINT I,
180 INPUT A
190 LET G(I)=A
200 NEXT I
210 LET S=0
220 FOR I=1 TO N
```

```
230 LET S=S+G(I)
240 NEXT I
250 PRINT
260 LET M=S/N
270 PRINT "CLASS AVERAGE IS ";M
280 LET H=G(1)
290 LET L= G(1)
300 FOR I=2 TO N
310 IF L<G(I) THEN 330
320 LET L=G(I)
330 IF H>G(I) THEN 350
340 LET H=G(I)
350 NEXT I
360 PRINT "HIGHEST GRADE IS ";H
370 PRINT "LOWEST GRADE IS ";L
380 END
```

Turn on the computer, bring up ATARI BASIC, and run this program using the data at the beginning of the example. If you have any difficulty with the highest and lowest search in lines 280 through 350, trace the program in detail.

Example 2 - Course Grades

You can easily extend the ideas in example 1 to a two-dimensional array. Suppose the class has ten students, and the course grade is based upon five examinations. Typical results for such a class might be

Student Number

		1	2	3	4	5	6	7	8	9	10
	1	92	71	81	52	75	97	100	63	41	75
	2	85	73	79	49	71	91	93	58	52	71
Exam	3	89	74	80	61	79	88	97	55	51	73
	4	96	68	84	58	80	93	95	61	47	70
	5	82	72	82	63	73	92	93	68	56	74

You can use an array with a FOR NEXT statement to READ the data from DATA statements. The computer is to find and print out the following information:

```
STUDENT          COURSE AVE
1              (Computer prints average, etc.)
2
3
etc.
TEST             CLASS AVE
1              (Computer prints average, etc.)
2
3
etc.
```

The program should start with a DIM statement although the DATA statements can go anywhere in the program.

```
100 DIM G(5,10)
```

This statement reserves memory space for an array with five rows and ten columns. The row number (R) will be the examination number, and the column number (C) will correspond to the student number.

```
110 DATA 92,71,81,52,75,97,100,63,41,75
120 DATA 85,73,79,49,71,91,93,58,52,71
130 DATA 89,74,80,61,79,88,97,55,51,73
140 DATA 96,68,84,58,80,93,95,61,47,70
150 DATA 82,72,82,63,73,92,93,68,56,74
```

All the numbers can be read into the program with the following FOR NEXT statement.

```
160 FOR R=1 TO 5
170 FOR C=1 TO 10
180 READ A
190 LET G(R,C)=A
200 NEXT C
210 NEXT R
```

You have just programmed the numbers to be read into the matrix G by rows. Thus, the data in line 110 become row 1 of the matrix G, and so forth. Before doing anything else, print out the required headings.

```
220 PRINT "STUDENT", "COURSE AVE"
230 PRINT
```

Now you can compute the course average for each student.

```
240 FOR C=1 TO 10
```

Line 230 opens a loop that will look at each column in the matrix. For each value of C, the computer must find the column average and print it out.

```
250 LET S=0
260 FOR R=1 TO 5
270 LET S=S+G(R,C)
280 NEXT R
290 PRINT C,S/5
```

Then close the C loop.

```
300 NEXT C
```

Now the process is repeated except that the averages are computed on rows rather than columns.

```
310 PRINT
320 PRINT "TEST", "CLASS AVE"
330 PRINT
340 FOR R=1 TO 5
350 LET S=0
360 FOR C=1 TO 10
370 LET S=S+G(R,C)
380 NEXT C
390 PRINT R,S/10
400 NEXT R
```

Finally the END statement,

```
410 END
```

The complete program follows:

```
100 DIM G(5,10)
110 DATA 92,71,81,52,75,97,100,63,41,75
120 DATA 85,73,79,49,71,91,93,58,52,71
130 DATA 89,74,80,61,79,88,97,55,51,73
140 DATA 96,68,84,58,80,93,95,61,47,70
150 DATA 82,72,82,63,73,92,93,68,56,74
160 FOR R=1 TO 5
170 FOR C=1 TO 10
180 READ A
190 LET G(R,C)=A
200 NEXT C
210 NEXT R
220 PRINT "STUDENT", "COURSE AVE"
230 PRINT
240 FOR C=1 TO 10
250 LET S=0
260 FOR R=1 TO 5
270 LET S=S+G(R,C)
280 NEXT R
290 PRINT C,S/5
300 NEXT C
310 PRINT
320 PRINT "TEST", "CLASS AVE"
330 PRINT
340 FOR R=1 TO 5
350 LET S=0
360 FOR C=1 TO 10
370 LET S=S+G(R,C)
380 NEXT C
390 PRINT R,S/10
400 NEXT R
410 END
```

This program illustrates valuable programming techniques involving arrays. It is worth studying and executing on your computer.

Example 3 - Array Operations

The final example is a series of short programs that will be given without explanation. Study each program until you are sure you understand what is taking place.

a. Write a program using FOR NEXT loops to load a three by four array with 1s.

```
100 DIM X(3,4)
110 FOR R=1 TO 3
120 FOR C=1 TO 4
130 LET X(R,C)=1
140 NEXT C
150 NEXT R
160 END
```

b. Write a program to generate and load the numbers

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048

into a one-dimensional array.

```
100 DIM Z(11)
110 LET Z(1)=2
120 FOR I=2 TO 11
130 LET Z(I)=2*Z(I-1)
140 NEXT I
150 END
```

c. Write a program to read in the array

$$\begin{bmatrix} 2 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix}$$

from DATA statements and then print out the array.

```
100 DIM A(2,3)
110 FOR R=1 TO 2
120 FOR C=1 TO 3
130 READ X
140 LET A(R,C)=X
150 NEXT C
160 NEXT R
170 FOR R=1 TO 2
180 FOR C=1 TO 3
190 PRINT A(R,C),
200 NEXT C
```

```
210 PRINT
220 PRINT
230 NEXT R
240 DATA 2,3,5
250 DATA 1,4,2
260 END
```

8–5 PROBLEMS

1. Write a program using the DATA statements

```
200 DATA 12
210 DATA 2,1,4,3,2,4,5,6,3,5,4,1
```

   The program will read the size of an array from the first DATA statement, read the elements of the array from the second DATA statement, load them into an array X, and print out the array.

2. Write a program to fill a four-by-three array with 2s.

3. Write a program to call for the input of a square N by N matrix where N is a whole number no larger than 10. Program the computer to find and print out the sum of the entries on the main diagonal of the array.

4. Write a BASIC program using the READ command to read twenty-five numbers from the DATA statements into a one-dimensional array named A. Program the computer to search the array and print out the number of elements in the array that are greater than fifty. Fill in the required DATA statements with any numbers you choose.

5. Write a program to call for the input of an M by N matrix. Assume that both M and N are no larger than 15. Then have the computer find and print out the sum of all the elements in the matrix.

6. The program below is supposed to compute and print out the sum of the elements in a one-dimensional array that are positive but not greater than 10. As it stands the program is incorrect. What's wrong?

```
100 DIM A(6)
110 FOR I=1 TO 6
120 INPUT T
130 LET A(I)=T
```

```
140 NEXT I
150 LET S=0
160 FOR I=6 TO 1 STEP -1
170 IF A(I)>10 THEN 200
180 IF A(I)<0 THEN 200
190 LET S=S+A(I)
200 NEXT I
210 PRINT S
220 END
```

7. What will be output if you run the following program?

```
100 DIM Y(6)
110 FOR I=1 TO 6
120 READ A
130 LET Y(I)=A
140 NEXT I
150 DATA 2,1,3,1,2,1
160 LET S1=0
170 LET S2=0
180 FOR I=1 TO 6
190 LET S1=S1+Y(I)
200 LET S2=S2+Y(I)^2
210 NEXT I
220 LET X=S2-S1
230 PRINT X
240 END
```

8. What will be output if you run the following program?

```
100 DIM A(10)
110 FOR I=1 TO 10
120 READ X
130 LET A(I)=X
140 NEXT I
150 LET S=A(1)
160 FOR I=1 TO 9
170 LET A(I)=A(I+1)
180 NEXT I
190 LET A(10)=S
200 FOR I=1 TO 10
210 PRINT A(I)
220 NEXT I
230 DATA 10,9,8,7,6,5,4,3,2,1
240 END
```

9. What will be printed out if you run the following program?

```
100 DIM X(4,4)
110 FOR I=1 TO 4
120 FOR J=1 TO 4
130 READ A
140 LET X(I,J)=A
150 NEXT J
160 NEXT I
170 DATA 1,2,3,4,2,3,4,5
180 DATA 3,4,5,6,4,5,6,7
190 LET S=0
200 FOR I=1 TO 4
210 LET S=S+X(I,5-I)
220 NEXT I
230 PRINT S
240 END
```

10. What will be printed out if you run the following program?

```
100 DIM Y(4,4)
110 FOR R=1 TO 4
120 FOR C=1 TO 4
130 LET Y(R,C)=0
140 NEXT C
150 NEXT R
160 FOR R=1 TO 4
170 FOR C=1 TO 4
180 LET Y(R,C)=R*C
190 NEXT C
200 NEXT R
210 FOR R=1 TO 4
220 FOR C=1 TO 4
230 PRINT Y(R,C);" ";
240 NEXT C
250 PRINT
260 NEXT R
270 END
```

11. Write a BASIC program to call for the input of N (assumed to be a whole number between 1 and 100), then input a one-dimensional array with N elements. The program should sort the array into descending order, and finally print out the sorted array.

12. Let's assume that the first number in the DATA statements gives the number of pieces of data to follow. Assume that the pieces of data are all whole numbers between 1 and 10 inclusive. Write a program that will compute the number of 1s, 2s, etc., in the data and then print the information. (Hint: Use the data as they are read in as a subscript to increment an element of an array used to count the numbers.)

13. What will be printed out if you run the following program?

```
100 DIM Z(6,6)
110 FOR R=1 TO 6
120 FOR C=1 TO 6
130 LET Z(R,C)=0
140 NEXT C
150 NEXT R
160 FOR R=1 TO 5 STEP 2
170 FOR C=R TO 6
180 LET Z(R,C)=1
190 NEXT C
200 NEXT R
210 FOR R=1 TO 6
220 FOR C=1 TO 6
230 PRINT Z(R,C);" ";
240 NEXT C
250 PRINT
260 PRINT
270 NEXT R
280 END
```

14. If you run the program below, what will the computer print out?

```
100 DIM A(5,5)
110 FOR R=1 TO 5
120 FOR C=1 TO 5
130 READ X
140 LET A(R,C)=X
150 NEXT C
```

```
160 NEXT R
170 DATA 2,2,2,2,2,2,2,2,2,2
180 DATA 2,2,2,2,2,2,2,2,2,2
190 DATA 2,2,2,2,2
200 FOR C=5 TO 1 STEP -1
210 FOR R=1 TO C
220 LET A(R,C)=3
230 NEXT R
240 NEXT C
250 FOR R=1 TO 5
260 FOR C=1 TO 5
270 PRINT A(R,C);" ";
280 NEXT C
290 PRINT
300 NEXT R
310 END
```

15. Write a program to read the following array from DATA statements, then print out the array.

$$\begin{bmatrix} 2 & 1 & 0 & 5 & 1 \\ 3 & 2 & 1 & 3 & 1 \end{bmatrix}$$

16. Write a program to read the following array from DATA statements, then print out the array.

$$\begin{bmatrix} 5 & 3 \\ 2 & 0 \\ -1 & 1 \\ 4 & 2 \\ 2 & 6 \end{bmatrix}$$

17. Write a BASIC program that will call for the input of an M by N array. The program should then compute and print out the sum of the elements in each row and the product of the elements in each column.

18. Write a BASIC program that will read two arrays from DATA statements. Both the arrays are two by three. Then have the program compute a third two-by-three array such that each element is the sum of the corresponding elements in the first two arrays. Print out the third array.

19. The data below represent sales totals made by salespeople over a one-week period.

		Mon	Tue	Wed	Thu	Fri	Sat
Salesperson	1	48	40	73	120	100	90
	2	75	130	90	40	110	85
	3	50	72	140	125	106	92
	4	108	75	92	152	91	87

Write a program that will compute and print out
a. The daily sales totals
b. The weekly sales totals for each salesperson
c. The total weekly sales

20. Write a program to call for the input of a four-by-four matrix. The program should compute a new matrix from the first with the rows and columns interchanged. That is, row 1 of the input matrix becomes column 1 of the new matrix. Row 2 of the input matrix becomes column 2 of the new matrix, and so on. Finally, have the program print out the new matrix.

21. Consider the two arrays below:

```
P          X

1          28
5          2
3          14
6          3
4          17
2          9
```

Each element of P "points" to an element of X. P(1) = 1 and X(1) = 28. P(2) = 5 and X(5) = 17. If you keep this process up, the values of X are listed in descending order. Write a program to set up two arrays X and P to some convenient length. Then call for the input of arbitrary values of X from the keyboard. Construct the array P so that its elements point to X in descending order as illustrated above. Then print out the two arrays as shown.

8–6 PRACTICE TEST

1. What is the purpose of the DIM statement?

2. We have an array named X. What variable name does BASIC use to locate the element in row 3, column 4?

3. Use an array in a progam to input a list of numbers, then find and print out the sum of the positive numbers in the list. The printout should look as follows:

   ```
   HOW MANY NUMBERS? (You type in the number)
   WHAT ARE THE NUMBERS? (You type them in)
   THE SUM OF POSITIVE ELEMENTS IS (Computer types out answer)
   ```

4. Write a program using FOR NEXT statements to load a four by six array with 4s.

5. What will be printed out if you run the following program?

```
100 DIM A(5,5)
110 FOR I=1 TO 5
120 FOR J=1 TO 5
130 LET A(R,C)=0
140 NEXT J
150 NEXT I
160 FOR I=1 TO 5
170 LET A(I,I)=2
180 NEXT I
190 FOR I=1 TO 5
200 FOR J=1 TO 5
210 PRINT A(I,J);" ";
220 NEXT J
230 PRINT
240 PRINT
```

```
250 NEXT I
260 END
```

6. The following array is named A:

$$\begin{bmatrix} 1 & 3 & 5 \\ 6 & 2 & 4 \end{bmatrix}$$

a. Write a DIM statement for A.

b. What is the value of A(2,3)?

c. If X = 1 and Y = 2, what is A(X,Y)?

d. What is A(A(1,1),A(2,2))?

CHAPTER 9

STRING VARIABLES

9—1 OBJECTIVES

Some of the most important applications of computers deal with characters rather than numbers. In this chapter you will learn to handle strings of characters as "string variables."

Handling String Input and Output

Before meaningful operations can be carried out on strings you must learn how to input and interpret output of string variables.

Using String Functions

You have already studied BASIC functions that operated on numbers. Now you will turn to functions that work on strings of characters.

Working with Program Examples

Your final goal is to write programs that work with string variables.

9–2 DISCOVERY EXERCISES

1. Turn on the computer, bring up ATARI BASIC, and enter the following program:

```
90 DIM A$(50)
100 INPUT A$
110 PRINT
120 PRINT A$
130 PRINT A$
140 END
```

The A$ in the program identifies the variable as a string variable. Run the program and type in your full name at the input prompt (the question mark). What happened?

2. Now edit line 90 to read

```
90 DIM A$(4)
```

Run the program and again type in your full name at the input prompt. What happened this time?

Clearly, the dimension of the variable must be large enough to cover anticipated inputs.

3. Now modify the program as follows:

```
90 DIM A$(50),B$(50)
100 INPUT A$
110 INPUT B$
120 PRINT
130 PRINT B$
140 PRINT A$
150 END
```

If you run the program and type the words INTELLIGENT and CONVERSATION at the input prompts, what do you think will be printed out?

Try it and record the output.

4. Let's try a different variation. Clear the program and enter the following:

```
90 DIM X$(50),Y$(50)
100 READ X,X$,Y,Y$
110 DATA 10,HERB,20,CHARLIE
120 PRINT X,Y
130 PRINT X$,Y$
140 END
```

This program contains several new ideas. What do you think will happen if you run the program?

Run the program and record the output.

5. Now that you have seen that strings can be included in DATA statements, you should go a bit further. Change line 100 to read

```
100 READ X$,X,Y,Y$
```

What will happen if you try to run the program in this form?

See if you were right. Record below what happened when you tried to run the program.

Error 8 is an attempt to input a string into a numeric variable. The type of information in the DATA statements must match the type of variable in READ statements.

6. Now on to a different topic. Clear the program and enter the following:

```
90 DIM C$(50)
100 INPUT C$
110 LET N=LEN(C$)
120 PRINT N
130 END
```

The new feature in this program is the function LEN(C$). It works on a string (in this case C$) rather than a number. Can you guess what the function does?

Notice that the result of the function LEN operating on a string must be a number since the result is assigned to a numeric variable N.

7. Run the program and at the input prompt type in ABCDE. What was printed out?

Try it again, but this time type in AARDVARK. Record the output.

By now you should have a pretty good idea what the LEN function does.

8. If you run the program and at the input prompt merely press the RETURN key, what do you think will be printed out by the computer?

Try it and see if you were correct. Now run the program and at the input prompt type in R O B E R T. Note the spaces between the characters. Record the output.

In the LEN function, do spaces count as characters?

9. Now you want to be able to specify a substring in a given string. That is, given a string A$, how can you specify the Mth through Nth characters of that string? The function that gives this substring is A$(M,N).

10. Now clear the program in memory and enter the one below:

```
90 DIM A$(50)
100 INPUT A$
110 PRINT "M = ";
120 INPUT M
130 PRINT "N = ";
140 INPUT N
150 PRINT A$(M,N)
160 GOTO 110
170 END
```

Run the program and at the input prompt for the string, type MISSISSIPPI. Note that the length of the string is 11. Enter 4 for M and 8 for N. Record the output.

Now enter 7 for M and 9 for N. Again, record the output.

This time enter 1 for M and 4 for N and record the output.

11. Jump the computer out of the input loop, clear the program from memory, and enter the one below:

```
90 DIM A$(50)
100 INPUT A$
110 INPUT I
120 PRINT A$(I,I)
130 PRINT
140 GOTO 110
150 END
```

Run the program and at the first input prompt type in ABCDEFGHIJKLMNOPQRSTUVWXYZ. At the second input prompt type in 20. What happened?

Run the program again and type in 10 at the input prompt for I. What happened?

Experiment with various values of I between 1 and 26. Describe in your own words what happens when the computer is directed to print out A$(I,I).

12. Now type 30 at the input prompt for I. What happened?

By now you should understand fairly clearly what happens when the computer prints out A$(I,I). Of course, in this instance, the value of I is greater than the length of the string. We will return to this topic in the discussion section.

13. Experiment on your own with this program. Try various strings and different values of M and N until you understand exactly how the A$(M,N) function works.

14. Now on to a different topic. Clear the program in memory and enter the following:

```
90 DIM A$(50),B$(50)
100 INPUT A$
110 INPUT B$
120 IF A$<B$ THEN 150
130 PRINT B$
140 GOTO 100
150 PRINT A$
160 GOTO 100
170 END
```

Take a few moments to study the program. Clearly, the interesting part is in line 120 where the strings A$ and B$ are involved in an IF THEN statement. In particular, what do you suppose A$ < B$ means with regard to strings?

The way to find out if you are right or not is to run a few test cases. For example, run the program and at the first input prompt, type DUCK; at the second prompt, type CHICKEN. Record the output.

15. This time type HOUSE followed by TELEVISION. Record the output.

Keep experimenting with words or letters of your choice until you understand exactly what the expression A$ < B$ means. Once you understand this, it should be easy for you to see what A$ = B$, A$ > = B$, and A$ <> B$ mean.

16. Jump the computer out of the input loop. Clear the program in memory. Go on to the next string variable statement. Enter the following program:

```
100 INPUT N
110 PRINT CHR$(N)
120 GOTO 100
130 END
```

Now run the program and at the input prompt, type in 65. What happens?

The program will keep looping through as long as you desire. This time try 66. What happens?

You may also wish to try 43, 49, and 50.

17. By now, you have probably realized that CHR$ converts a position number to its corresponding character in the set of characters the computer uses. Experiment with this program trying out various numerical inputs in the range 33 to 122. You should see that you can refer to a character either by the character itself or by its position number in the set of characters.

18. To see a portion of the total available character set, jump the computer out of the input loop, clear the memory and type in the following program.

```
100 FOR N=33 TO 122
110 PRINT CHR$(N);" ";
120 NEXT N
130 END
```

You can see the entire ATASCII character set by referring to Appendix C of the *ATARI BASIC Reference Manual.*

19. Now on to a different topic. Clear the program in memory and type the following program:

```
90 DIM A$(50)
100 INPUT A$
110 LET X=ASC(A$)
120 PRINT X
130 GOTO 100
140 END
```

Run the program and at the input prompt, type Z. What happens?

This new function, ASC(A$), is just the reverse of CHR$(N). It converts a character to its equivalent position number in the character set used by the computer. Use various letters and numbers and compare your results to those you obtained in step 16.

20. There is one more detail to be seen to. Your computer should still be at the input prompt waiting for A$ to be typed in. This time type in POTATO. Record the output.

Now try PEA and note the results. The only similarity between the two words is that they both have the same first letter. We will return to this concept later.

21. Jump the computer out of the loop, clear the memory, and type in the following program.

```
90 DIM A$(50),B$(50)
100 PRINT
110 INPUT A$
120 INPUT B$
130 LET A$(LEN(A$)+1)=B$
140 PRINT A$
150 END
```

Run the program. At the input prompts, type HONEY and BEE respectively. Record the output.

Now edit lines 90 and 130 and add lines 125 and 135 as follows:

```
90 DIM A$(50),B$(50),C$(1)
125 LET C$=" "
130 LET A$(LEN(A$)+1)=C$
135 LET A$(LEN(A$)+1)=B$
```

Run the program and again type in HONEY and BEE at the input prompts. How many words are displayed?

22. Now let's carry this idea one step further. Either modify the program in memory or clear the memory and type in the following program.

```
90 DIM A$(50),B$(50),C$(1),D$(50)
100 PRINT
110 INPUT A$
120 INPUT B$
122 LET D$="HELLO MY"
125 LET C$=" "
130 LET D$(LEN(D$)+1)=C$
132 LET D$(LEN(D$)+1)=A$
135 LET D$(LEN(D$)+1)=B$
140 PRINT D$
150 END
```

Display the program and check its accuracy. Run the program and again input HONEY and BEE at the input prompts. Record the output.

23. This concludes the computer work for this chapter. Turn off the computer and go on to the next section.

9—3 DISCUSSION

Handling String Input and Output

As you have already seen, a set of characters surrounded by quotation marks is called a string. (The quotation marks are not part of the string.) The new idea in this chapter is that the string can be treated as a variable — the string variable.

The string variable is identified by appending a dollar sign ($) to a name. Thus, BARN$, BOX$, and B$ are string variable names.

Input and output of string variables are handled the same way as for numeric variables except in one case. You can mix numeric and string variables in the same BASIC statements. For example:

```
100 PRINT A$,X,Y,Z$
110 INPUT N,M$
120 READ A$,B$,Z
```

However, you cannot follow a string variable with another variable in an INPUT statement.

You must be careful that the input in either INPUT or READ statements matches the type of variable given. In line 110 above, the computer would be looking for a number and a string of characters Note that the string variable follows the numeric variable. It would not work the other way around. In addition, you must be aware that you can type in 123456789 and if the computer is looking for a string it will identify this quantity as a string, not as a number. The reason is that the string, as has been pointed out, consists of characters, and symbols 0 through 9 are part of the standard character set that will be discussed later. If, however, the computer is looking for a number and you type in ABCDEFGHI, you will get an error statement.

Using String Functions

The LEN function is used to determine the length of a string. If, for example, A$ = "HOW NOW BROWN COW" then LEN(A$) = 17. Note that the spaces are counted as characters. You can also have a "null" string. If A$ = "" (there is nothing inside the quotation marks), then LEN(A$) = 0.

- **LEN(A$) gives the number of characters in A$.**

A substring is a piece or segment of a string. Consider the following program:

```
90 DIM A$(20)
100 LET A$="ROBERT E. LEE"
110 PRINT A$(8,13)
120 END
```

The expression A$(8,13) identifies the substring of A$ consisting of six characters starting at the eighth character. If you run the program, the output will be E. LEE.

■ A$(M,N) gives Mth through Nth characters of A$

String variables can be compared in IF THEN statements. The comparison is done by alphabetical ordering. Thus A < B since A comes before B in the alphabet. CAT < DOG, HOUSE > CAR, PEA < PEARL, and so on.

The last two functions discussed here, CHR$(N) and ASC(A$), are used to handle the ATASCII standard character set. This set consists of two-hundred and fifty-six characters numbered 0 through 255. Refer to Appendix C of the *ATARI BASIC Reference Manual* for a complete listing of the ATASCII character set.

Characters 0 through 31 have special purposes and are irrelevant here. The numerals 0 through 9 are numbered 48 through 57. The upper-case letters A through Z are numbered 65 through 90. The lower-case letters a through z are numbered 97 through 122. The other numbered characters include punctuation marks, arithmetic operators (+, −, *, etc.), and other special characters.

Two string functions work with the ATASCII character set. First, CHR$(N) returns the Nth character from the ASCII character set. For example, CHR$(65) = "A", CHR$(90) = "Z", and so forth. You can also turn things around and convert from a character to its ATASCII number. This is done with the ASC(A$) function. For example, ASC("A") = 65 and ASC("Z") = 90.

Suppose that A$ = "AIRPLANE". What is ASC(A$)? Since the length of the string is greater than one, only the first character is considered. In this case the first character is A and ASC(A$) = 65.

Strings can be joined together (catenated) using the LEN function. Thus, for example, if you run the following program

```
90 DIM A$(50),B$(50)
100 LET A$="HAPPY"
110 LET B$=" GO LUCKY"
120 LET A$(LEN(A$)+1)=B$
130 PRINT A$
140 END
```

it will display

```
HAPPY GO LUCKY
```

Each time a string variable is used, it must be dimensioned in a DIM statement. The DIM should be large enough for the anticipated number of characters in the string variable.

9—4 PROGRAM EXAMPLES

Example 1 - String Reversal

The task is to write a program to call for the input of a string and then print it back in reverse order. Begin with the DIM statement and the string input.

```
90 DIM A$(50)
100 INPUT A$
```

The next few lines print the string back in reverse order.

```
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT A$(X,X);
130 NEXT X
```

The loop steps backwards from the length of the string to 1. The function A$(X,X) identifies the substring in A$ consisting of 1 character starting at character number X. This isolates a single character.

With an END statement added the complete program is

```
90 DIM A$(50)
100 INPUT A$
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT A$(X,X);
130 NEXT X
140 END
```

Try running this program using your own name as input.

Example 2 - Word Count

The number of words in a sentence can be determined from the number of spaces (assuming that the only purpose of a space is to separate words). The following program prints out the number of words in the input string.

```
90 DIM A$(50)
100 INPUT A$
110 LET S=0
120 FOR I=1 TO LEN(A$)
130 IF A$(I,I)<>" " THEN 150
140 LET S=S+1
150 NEXT I
160 PRINT "WORD COUNT = ";S+1
170 END
```

Study the program until you see exactly how it works. Try out the program by typing in a sentence. Verify that it works correctly.

Example 3 - Replacement Code

Suppose you want a program to encode a sentence. A simple way to construct a code (which incidentally could be broken very rapidly with computers) is to replace each character in the message with another. This is done most easily by reference to the ATASCII character set. However, let's do this one "from scratch."

The first part of the program calls for the DIM statement and the input of the string to be coded and sets up the conversion scheme.

```
90 DIM A$(500),B$(30),C$(30)
100 LET B$="ABCDEFGHIJKLMNOPQRSTUVWXYZ ,."
110 LET C$="ETAVZBHCW KPSYDF,GXIMJLONQU.R"
120 INPUT A$
```

B$ contains the characters that can be used in the input string that is to be coded. C$ is the replacement key. An A in the input string is to be replaced by an E, an F by a B, a J by a space, and so on.

Now we can examine each character and do the replacement.

```
130 FOR I=1 TO LEN(A$)
140 FOR J=1 TO 29
150 IF A$(I,I)<>B$(J,J) THEN 180
160 PRINT C$(J,J);
170 GOTO 190
180 NEXT J
190 NEXT I
```

The outer I loop steps through each character in A$. The inner J loop compares the Ith character of A$ to the character in B$ until a match is found at the Jth character. When this happens, the coded Jth character of C$ is printed out, and the program goes on to the next character in A$.

Finish the program with

```
200 PRINT
210 END
```

The complete program is

```
90 DIM A$(500),B$(30),C$(30)
100 LET B$="ABCDEFGHIJKLMNOPQRSTUVWXYZ ,."
110 LET C$="ETAVZBHCW KPSYDF,GXIMJLONQU.R"
120 INPUT A$
130 FOR I=1 TO LEN(A$)
140 FOR J=1 TO 29
```

```
150 IF A$(I,I)<>B$(J,J) THEN 180
160 PRINT C$(J,J);
170 GOTO 190
180 NEXT J
190 NEXT I
200 PRINT
210 END
```

The code can be changed by rearranging the characters in C$. It might be interesting for you to try out the program and see how a coded message looks.

9–5 PROBLEMS

1. Write a program that calls for the input of a string and then prints the string out in a vertical column of characters.

2. If at the input prompt for the program below you type in the string ABCDEFGH, what will be output?

```
90 DIM A$(50)
100 INPUT A$
110 FOR J=1 TO LEN(A$) STEP 2
120 PRINT A$(J,J);
130 NEXT J
140 END
```

3. Write a program to count the number of vowels in an input string.

4. Write a program that calls for the input of a string and then prints the words in the string in a vertical column.

5. Ask for a sentence to be input. Generate a new string from this sentence that has all the spaces removed. Then print out the new string.

6. You want to know how many times each of the twenty-six letters in the alphabet (you may assume that they are all upper-case) occurs in ten sentences to be typed in at the keyboard. Do not count spaces or punctuation marks. Write a program to compute and print out a table consisting of each of the letters and the number of times it occurrs in the sentences. Do you think you could identify an author with the use of such a table?

7. Assume that five sentences are to be typed in one at a time. Write a program to count the number of times the word THE appears in the five sentences.
8. Write a program to count the number of spaces in an input string.
9. Write a program that calls for the input of a string and counts the number of times the character I is followed by the character N.

9—6 PRACTICE TEST

1. How are string variables identified in BASIC?

2. If A$ = "KITTY" and B$ = "KITTYCAT" then A$ > B$. True or false?

3. If A$ = "HOW NOW BROWN COW", write a function that will extract NOW BROWN.

4. Write a program that calls for the input of a string and then keep printing back the string with one character deleted each time until nothing is left. If, for example, you typed in PIECE OF CAKE, the computer should print out

```
PIECE OF CAKE
PIECE OF CAK
PIECE OF CA
PIECE OF C
PIECE OF
PIECE OF
PIECE O
PIECE
PIECE
PIEC
PIE
PI
P
```

5. What will be printed out if you run the following program?

```
100 FOR N=65 TO 90
110 FOR M=65 TO N
120 PRINT CHR$(M);
130 NEXT M
140 PRINT
150 NEXT N
160 END
```

CHAPTER 10

SOUND AND SUBROUTINES

10—1 OBJECTIVES

In this chapter you will learn how the computer can be programmed to produce sound and perform suboperations. This can be done through either program segments or special on-line instructions.

Creating Music

In order to create music, you must be able to generate notes.You will learn to make notes of different intensity and octaves.

Exploring Subroutines

When complicated operations are to be repeated, subroutines are often very useful. You will learn how subroutines can be set up and used in BASIC programs.

Working with Program Examples

Sometimes it is difficult for the beginner to see the value of subroutines. You will learn how useful subroutines are as a programming tool.

10—2 DISCOVERY EXERCISES

1. Turn on the computer and bring up ATARI BASIC. Then type in the following direct mode statement.

```
SOUND 0,60,10,8
```

Did you hear anything? If not, turn up the volume on the television set. If you still do not hear anything, retype the line: You probably made a typing error when you entered the statement.

2. Now type

```
SOUND 1,81,10,8
SOUND 2,96,10,8
SOUND 3,121,10,8
```

How many "voices" do you hear?

3. Type

```
END
```

What happened?

4. Now type the following.

```
100 INPUT PITCH
110 IF PITCH=0 THEN 140
120 SOUND 0,PITCH,10,8
130 GOTO 100
140 END
```

Before running the program, think about what will occur. Now run the program and each time the input prompt is displayed, type in one of the numbers in the following sequence: 60, 81, 96, 121, 240

Did the notes get successively higher or lower?

An input of 0 will stop the sound.

5. Clear the program in memory and type in the following.

```
100 INPUT QUALITY
110 IF QUALITY=0 THEN 140
120 SOUND 0,121,QUALITY,8
130 GOTO 100
140 END
```

Run the program and type in successive numbers from 6 to 19 each time the input prompt is displayed. What numbers produced sound?

6. Now clear the program in memory and type in the following.

```
100 INPUT LOUDNESS
110 IF LOUDNESS=0 THEN 140
120 SOUND 0,121,10,LOUDNESS
130 GOTO 100
140 END
```

What effect do you think the last of the four numbers in the SOUND statement has?

Run the program and enter successive numbers from 1 to 16 at each input prompt. Did the note get louder?

What happened when you entered 16?

Enter 0 to exit the program.

7. Now add the following lines to the program.

```
121 SOUND 1,60,10,LOUDNESS
122 SOUND 2,81,10,LOUDNESS
123 SOUND 3,96,10,LOUDNESS
```

and run the program entering the same values as above. What happened?

8. Now on to something new. Clear the program in memory and enter the following program:

```
100 PRINT "A",
110 GOSUB 200
120 PRINT "B",
130 GOSUB 300
140 PRINT "C",
150 END
200 PRINT 1,
210 RETURN
300 PRINT 2,
310 RETURN
```

This program has two statements you haven't seen so far. They are GOSUB and RETURN. The program itself is intended only to provide practice in understanding these new statements. Run the program and record the output.

Compare what was printed out with the program lines that caused the printout.

9. To which statement does the GOSUB statement in line 110 transfer the program? (Hint: Look at the table in step 11.)

10. To which statement does the RETURN statement in line 210 transfer the program? (Hint: Again, examine the table in step 11.)

11. The table below indicates the flow of the program as it is executed.

Line Number	What Happens
100	Print out A
110	Transfer to line 200
200	Print out 1
210	Transfer to line 120
120	Print out B
130	Transfer to line 300
300	Print out 2
310	Transfer to line 140
140	Print out C
150	End of program

Study this flow carefully and follow through with the program. Can you see the purpose of the GOSUB and RETURN statements yet? What about the placement of the END statement?

12. Clear the program in memory. Enter the following program:

```
100 REM PROGRAM TO DEMONSTRATE SUBROUTINES
110 DIM X(4)
120 FOR I=1 TO 4
130 READ A
140 LET X(I)=A
150 NEXT I
160 REM SORT
170 GOSUB 400
180 REM PRINT OUT SORTED ARRAY
190 FOR I=1 TO 4
```

```
200 PRINT X(I);" ";
210 NEXT I
220 PRINT
230 LET X(3)=7
240 REM SORT AGAIN
250 GOSUB 400
260 REM PRINT OUT SORTED ARRAY
270 FOR I=1 TO 4
280 PRINT X(I);" ";
290 NEXT I
300 END
310 DATA 2,1,5,6
400 REM SUBROUTINE TO SORT
410 FOR I=1 TO 3
420 IF X(I+1)>X(I) THEN 470
430 LET C=X(I+1)
440 LET X(I+1)=X(I)
450 LET X(I)=C
460 GOTO 410
470 NEXT I
480 RETURN
```

Display lines 100 through 200 by typing

```
LIST 100,200
```

Check that these lines are correct. Display and check the remaining lines. This program furnishes an example of how a subroutine might be used. The subroutine in lines 400 through 480 sorts the array X into ascending order. Run the program and record the output.

Note that the original array is

```
2 1 5 6
```

You can see this by checking the DATA statement in line 310. In line 170, the program jumps to the subroutine, which sorts the numbers. After the program returns to line 180, the sorted array is now

```
1 2 5 6
```

In line 230, the third element of the array is changed. Line 150 branches the program to the subroutine again for another sorting. After the return to line 260, the sorted array

```
1 2 6 7
```

is printed out. Finally, the END command in line 300 stops the program. Clearly, we could sort the array X as often as desired merely by inserting a statement GOSUB 400. This is certainly more efficient that writing out sorting instructions each time they are needed.

13. Now let us look at another statement. Clear the memory and type the following program.

```
100 PRINT "ENTER A NUMBER BETWEEN 1 AND 5"
110 PRINT "ENTER 5 TO QUIT."
120 COLOR 1
130 INPUT N
140 ON N GOSUB 1000,2000,3000,4000
150 IF N=5 THEN END
160 GOTO 100
1000 GRAPHICS 8
1010 PLOT 5,5:DRAWTO 20,5
1020 DRAWTO 20,20:DRAWTO 5,5
1030 RETURN
2000 GRAPHICS 8
2010 PLOT 5,5:DRAWTO 50,5
2020 DRAWTO 50,20:DRAWTO 5,20
2030 DRAWTO 5,5
2040 RETURN
3000 GRAPHICS 8
3010 PLOT 5,50:DRAWTO 5,5
3020 DRAWTO 30,5:DRAWTO 30,20
3030 DRAWTO 5,20
3040 RETURN
4000 GRAPHICS 8
4010 PLOT 50,0:DRAWTO 0,0
4020 DRAWTO 30,10:DRAWTO 0,20
4030 RETURN
```

What figure will be drawn if you enter 3?

Run the program. Were you right?

Input in the other choices to see what is drawn. Input 5 to quit.

14. This completes the computer work for this chapter. Turn off the computer.

10—3 DISCUSSION

Creating Music

The SOUND statement instructs the computer to play notes. The four numbers that follow the SOUND statement allow for the variation in the notes played. The first number indicates which "voice" is to be used. The four voices available are numbered 0 through 3. As you saw in the discovery exercises, the effect of harmonizing can be attained by using all four voices at the same time.

The second number indicates what note will be played. A value of 121 will play the musical note middle C. A value of 60 will also play a C note except that it will be one octave higher, while a value of 243 will play a C note that is one octave lower than middle C. A list of numbers and their corresponding notes can be found in Table 10.1 of the *ATARI BASIC Reference Manual.*

The third number refers to the quality of the note being played. Essentially, there are two qualities of notes —"pure" tone and distortion. Distortion allows you to produce sound effects. Values of 10 and 14 produce "pure" tones, while other even values produce a variety of distortions. Odd values produce silence.

The fourth number determines how loud the note will be; the higher the number, the louder the note. Thus the direct mode statement

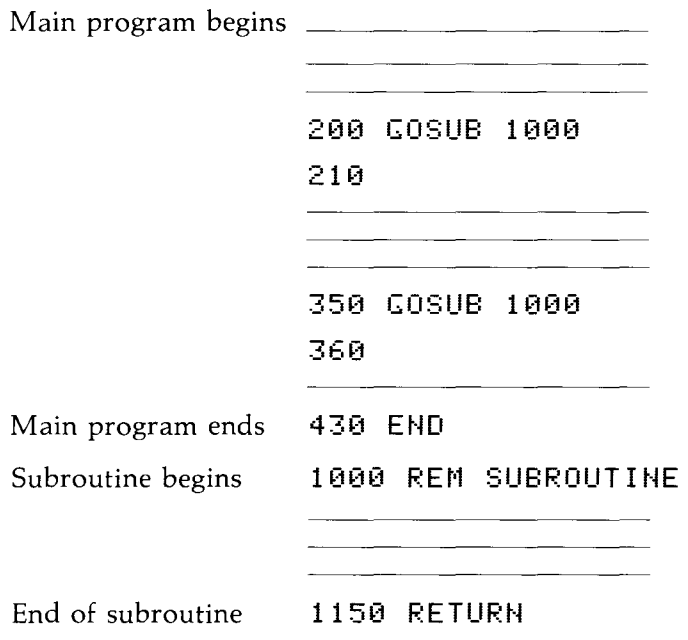
```
SOUND 2,121,10,8
```

produces a middle C note in voice 2 with a "normal" loudness.

■ **Use the SOUND statement to play music on the computer.**

Exploring Subroutines

You will almost certainly encounter complicated situations in which you want to carry out the same process many times in a program. Subroutines are very useful for this purpose. The diagram below indicates how a subroutine might be used in a program.



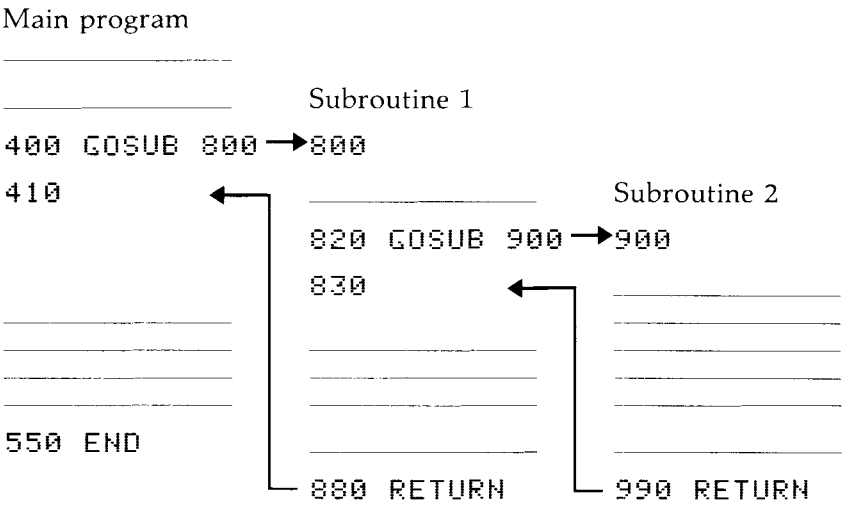
When the computer reaches the GOSUB in line 200 of the program above, it jumps to line 1000 and executes the subroutine beginning on that line. When the computer encounters the RETURN in line 1150, it goes back to line 210, the next higher line number after the GOSUB that put it in the subroutine. Then the computer proceeds through the main program until it reaches the GOSUB in line 350, where it again branchs to the subroutine in line 1000. This time the RETURN statement directs the computer back to line 360.

Of course, you can use GOSUB 1000 as many times as you want in this program or can have as many subroutines as you need.

Generally, the main program is at the beginning and the subroutines are grouped together at the end. There is a good reason for this. The subroutines are performed only when they are called by a GOSUB. After the main program is finished, put an END statement in the program.

■ Transfer to subroutines with GOSUB.

It is possible, and sometimes desirable, to jump to a subroutine from a subroutine. The diagram below indicates how the computer treats such an event.



Note that the computer jumps from 400 to 800, then from 820 to 900, and finally arrives at the RETURN in line 990. But where does the RETURN direct the computer—to line 410 or to line 830? The rule is that the RETURN directs the computer back to the next statement after the GOSUB that put it in the subroutine containing the RETURN. The computer is in subroutine 2 because of the GOSUB in line 820; hence the RETURN in line 990 directs it back to line 830. The same rule applies to the RETURN in line 880. At that point the computer is in subroutine 1 and arrived there from the GOSUB in line 400. Thus, the RETURN in line 880 directs the computer back to line 410.

- **Get back from subroutines with RETURN**

The ON...GOSUB statement allows you to direct the computer to one of numerous subroutines, depending on the value of a numeric variable. For example, the statement

```
140 ON N GOSUB 300,400,500
```

will go to the subroutine at 300 if N is 1, to the subroutine at 400 if N is 2, or to the subroutine at 500 if N is 3.

At this point it may not be clear to you why subroutines are valuable. The need for subroutines becomes more evident as you acquire more skill as a programmer. For the moment, remember that subroutines are one of the most valuable tools available to the programmer.

10—4 PROGRAM EXAMPLES

Example 1 - Writing a Song

To compose music on the computer, you need to know what numbers play which notes and how to insert pauses. A simple FOR NEXT loop can accomplish this. The READ DATA statements are the most useful for producing songs. Start with the loop to generate the notes and pauses as follows:

```
100 FOR K=1 TO 54
110 READ PITCH,LENGTH
120 FOR L=1 TO LENGTH
130 SOUND 0,PITCH,10,8
140 NEXT L
150 NEXT K
```

This is the heart of the program because the inner FOR NEXT loop plays the desired note for the length of time given by the variable LENGTH. You can insert pauses easily by using a value that produces silence. You compose the song by choosing appropiate notes and pauses.

The complete program for the song **Row, Row, Row Your Boat** follows.

```
100 FOR K=1 TO 54
110 READ PITCH,LENGTH
120 FOR L=1 TO LENGTH
130 SOUND 0,PITCH,10,8
140 NEXT L
150 NEXT K
160 DATA 121,50,0,5,121,50,0,5
170 DATA 121,30,0,2,108,15,0,2
180 DATA 96,60,0,5,96,50,0,2
190 DATA 108,20,0,2,96,20,0,10
200 DATA 91,20,0,5,81,100,0,10
210 DATA 60,15,0,2,60,15,0,2
220 DATA 60,15,0,2,81,15,0,2
230 DATA 81,15,0,2,81,15,0,2
240 DATA 96,15,0,2,96,15,0,2
250 DATA 96,15,0,2,121,15,0,2
260 DATA 121,15,0,2,121,15,0,2
270 DATA 81,25,0,5,91,15,0,2
280 DATA 96,25,0,5,108,20,0,3
290 DATA 121,25,0,5
300 END
```

Note that you won't know the value of 54 in line 100 until you know how many READ statements you will need to program the song.

Example 2 - Rounding off Dollar Values to Cents

Business applications generally involve printing out the results of calculations in dollars and cents. Since the computer normally handles nine significant figures, it is not unusual for the computer to type 23.1579384 and similar amounts. Ordinarily, you would want to round off the figure to the nearest cent, or 23.16.

You can use a subroutine to round off the numbers. Let's write a program that will produce the following output when executed:

```
LABEL PRICE? (You type in price)
10% DISCOUNT IS (Computer prints discount price)
15% DISCOUNT IS (Computer prints discount price)
20% DISCOUNT IS (Computer prints discount price)
```

All dollar values typed out should be rounded off to the nearest cent.

First set up a method to do the rounding off. A subroutine to accomplish this is

```
300 Y=INT(X*100+.5)/100
310 RETURN
```

To see how this expression works, suppose X = 23.1597. Follow this value through the expression to see what happens.

X*100	=	2315.97
X*100+0.5	=	2316.47
INT(X*100+0.5)	=	2316
INT(X*100+0.5)/100	=	23.16

Therefore, 23.1597 was correctly rounded "up" to 23.16.

But suppose that X = 23.1547.

X*100	=	2315.47
X*100+0.5	=	2315.97
INT(X*100+0.5)	=	2315
INT(X*100+0.5)/100	=	23.15

the computer rounds 23.1547 "down" to 23.15.

The first few lines of the program are self-explanatory.

```
100 PRINT "LABEL PRICE"
110 INPUT Z
```

To obtain the first discount rounded off to the nearest cent, use the following subroutine:

```
120 LET X=0.9*Z
130 GOSUB 300
140 PRINT "10% DISCOUNT IS ";Y
```

The remaining discount prices are determined in the same way. The complete program is

```
100 PRINT "LABEL PRICE";
110 INPUT Z
120 LET X=0.9*Z
130 GOSUB 300
140 PRINT "10% DISCOUNT IS ";Y
150 LET X=0.85*Z
160 GOSUB 300
170 PRINT "15% DISCOUNT IS ";Y
180 LET X=0.8*Z
190 GOSUB 300
200 PRINT "20% DISCOUNT IS ";Y
210 END
300 Y=INT(X*100+.5)/100
310 RETURN
```

In lines 130, 160, and 190 the rounding-off subroutine is called and used. If the discount is 10 percent, the selling price is 90 percent of the original price, Z. Thus, the computer calculates 0.9*Z, which is rounded off to the nearest cent in the subroutine.

Example 3 - Carpet Estimating

Suppose you want to write a program that uses a subroutine to compute the price of installed carpet. There are four grades of carpet, and each is discounted as the quantity of carpet ordered increases. Assume that the price structure is as follows:

		1	2	3
Grade	A	$10.00	$ 8.50	$ 7.25
	B	13.25	12.00	9.75
	C	16.00	14.00	11.25
	D	20.00	17.20	15.25

The rows represent the four grades of carpet. The numbers at the top of the columns represent quantities installed, as follows:

1. First 15 square yards

2. Any part of the order exceeding 15 but not more than 25 square yards

3. Anything over 25 square yards

When executed, the program should produce the following output:

```
HOW MANY ROOMS? (You type in)
FOR EACH ROOM, TYPE IN LENGTH
AND WIDTH IN FEET SEPARATED
BY A COMMA

ROOM            DIMENSIONS

1               (You type in)
2               (You type in)
        (Loop until all rooms are entered)
   (Computer types out number) SQUARE YARDS REQUIRED
   CARPET GRADE                COST OF ORDER
   A                 (Computer types out,etc.)
   B
   C
   D
```

Before getting involved in the program, think a bit about the output. Since the output is in dollars and cents, you can use the subroutine in example 2 to round off the answers. The first few lines follow without difficulty.

```
100 PRINT "HOW MANY ROOMS";
110 INPUT N
120 PRINT "FOR EACH ROOM, TYPE IN LENGTH"
130 PRINT "AND WIDTH IN FEET SEPARATED"
140 PRINT "BY A COMMA"
150 PRINT
160 PRINT "ROOM","DIMENSIONS"
170 PRINT
```

You are ready to call for the input of the room dimensions. Use the variable A to keep track of the area of the rooms. Remember that the area of a room is its length times its width.

```
180 LET A=0
190 FOR I=1 TO N
200 PRINT I,
210 INPUT L,W
220 LET A=A+L*W
230 NEXT I
```

Since the total room area is now in square feet, include programming instructions to divide the area by 9 to convert to square yards, and then to print out the quantity of carpet required.

```
240 LET A=A/9
250 PRINT A;" SQUARE YARDS REQUIRED"
```

At this point include the price table in the program in the form of DATA statements.

```
260 DATA 10,8.5,7.25
270 DATA 13.25,12,9.75
280 DATA 16,14,11.25
290 DATA 20,17.2,15.25
```

Next print out the heading required for the price printout.

```
300 PRINT
310 PRINT "CARPET GRADE","COST OF ORDER"
320 PRINT
```

Now you come to the point in the program where the subroutine will be useful. Since you don't know precisely where the subroutine should begin, simply use a large line number and correct it later if needed.

```
330 REM COMPUTE PRICE FOR GRADE A
340 GOSUB 800
```

Let's write the subroutine now. First, for each of the grades of carpet you need the three prices. Use a READ statement to get the prices from the DATA statements.

```
800 REM SUBROUTINE TO COMPUTE CARPET PRICE
810 READ C1,C2,C3
```

Next the program should determine if the area of the carpet is less than 15, between 15 and 25, or more than 25 square yards and then compute the price accordingly.

```
820 IF A>25 THEN 860
830 IF A>15 THEN 880
840 LET P=C1*A
850 GOTO 890
860 LET P=15*C1+10*C2+(A-25)*C3
870 GOTO 890
880 LET P=15*C1+(A-15)*C2
890 GOSUB 1000
900 RETURN
```

Study this program segment to convince yourself that the price is being computed correctly. Note how the rounding off subroutine is called within this subroutine. Now return to the main program and add a statement to print out the first price.

```
350 PRINT "A",Y
```

Once you establish this pattern, the rest of the main program follows easily.

```
360 REM COMPUTE PRICE FOR GRADE B
370 GOSUB 800
380 PRINT "B",Y
390 REM COMPUTE PRICE FOR GRADE C
400 GOSUB 800
410 PRINT "C",Y
420 REM COMPUTE PRICE FOR GRADE D
430 GOSUB 800
440 PRINT "D",Y
450 END
```

You need the END statement in line 450 to prevent the program from falling into the subroutine.

The subroutine, as you see, saves programming time and makes programs shorter. If you rewrote this program without GOSUBs, you would need to add many statements at each point the GOSUB is used.

The complete program is

```
100 PRINT "HOW MANY ROOMS"
110 INPUT N
120 PRINT "FOR EACH ROOM, TYPE IN LENGTH"
130 PRINT "AND WIDTH IN FEET SEPARATED"
140 PRINT "BY A COMMA"
150 PRINT
160 PRINT "ROOM","DIMENSIONS"
170 PRINT
180 LET A=0
190 FOR I=1 TO N
200 PRINT I,
210 INPUT L,W
220 LET A=A+L*W
230 NEXT I
240 LET A=A/9
250 PRINT A;" SQUARE YARDS REQUIRED"
260 DATA 10,8.5,7.25
270 DATA 13.25,12,9.75
280 DATA 16,14,11.25
290 DATA 20,17.2,15.25
300 PRINT
310 PRINT "CARPET GRADE","COST OF ORDER"
320 PRINT
330 REM COMPUTE PRICE FOR GRADE A
340 GOSUB 800
350 PRINT "A",Y
360 REM COMPUTE PRICE FOR GRADE B
370 GOSUB 800
380 PRINT "B",Y
390 REM COMPUTE PRICE FOR GRADE C
400 GOSUB 800
410 PRINT "C",Y
420 REM COMPUTE PRICE FOR GRADE D
430 GOSUB 800
440 PRINT "D",Y
450 END
800 REM SUBROUTINE TO COMPUTE CARPET PRICE
810 READ C1,C2,C3
820 IF A>25 THEN 860
830 IF A>15 THEN 880
```

```
840 LET P=C1*A
850 GOTO 890
860 LET P=15*C1+10*C2+(A-25)*C3
870 GOTO 890
880 LET P=15*C1+(A-15)*C2
890 GOSUB 1000
900 RETURN
1000 LET Y=INT(P*100+.5)/100
1010 RETURN
```

Example 4 - Designing a House

This program draws a house. The subroutines in the program use the graphics statements we introduced in Chapter 3. As you read this program example, pay close attention to the REM statements. The instructions, set up, and GOSUBs to draw the parts of the house are at the beginning of the program. The subroutines begin in line 1000.

You draw the roof first. You select the roof position using the POINT MOVING program you studied in Chapter 6, step 15. You position the roof by moving the point to the location you wish using the L, R, U, and D keys.

You draw the frame next and make decisions about the windows and the door. Notice that the roof drawing subroutine (line 1000), window drawing subroutine (line 3000), and door drawing subroutine (line 4000) use the POINT MOVING program (line 5000) as a subroutine. Note that, except for the roof, each object is drawn using the subroutine in line 7000. Recall that after you position the point on the screen with the POINT MOVING program, you exit by typing a Q.

If you wish to enter and run this program, you may save yourself some typing by loading the POINT MOVING program you saved in Chapter 6 step 15 and and changing the numbering so it begins at line 5000. Note the small modification to this program in the listing of the complete program which follows. Again, do not become concerned with the statement in line 110 as it will be discussed fully in the last chapter.

```
100 REM     DESIGNING A HOUSE
110 OPEN #1,12,0,"K:"
120 DIM ANS$(20),Y$(1),ARRAY(10,2),S(10,2)
130 PRINT "YOU CAN MOVE THE POINT ON THE"
140 PRINT "SCREEN TO LOCATE THE BOTTOM"
150 PRINT "CENTER OF THE ROOF, WINDOWS,"
160 PRINT "AND DOOR ON THE HOUSE. PRESS"
170 PRINT "U,D,L,R TO MOVE THE POINT"
180 PRINT "ON THE SCREEN. WHEN YOU HAVE"
190 PRINT "THE POINT WHERE YOU WANT IT,"
200 PRINT "PRESS Q."
210 PRINT
220 PRINT "ENTER SCALE FACTOR (1 TO 6)."
230 INPUT SCALE
240 GRAPHICS 8
250 COLOR 1
260 REM DRAW ROOF
270 GOSUB 1000
280 REM DRAW HOUSE FRAME
290 GOSUB 2000
300 REM DRAW WINDOW
310 GOSUB 3000
320 REM DRAW DOOR
330 GOSUB 4000
340 END
1000 REM ROOF DRAWING SUBROUTINE
1010 REM GOSUB POINT MOVING SUBROUTINE
1020 GOSUB 5000
1030 REM GOSUB ARRAY READ AND
1040 REM SCALE SUBROUTINE
1050 LET NOOFPTS=5
1060 GOSUB 6000
1070 PLOT A,B:DRAWTO A+S(1,1),B+S(1,2)
1080 FOR R=2 TO 5
1090 DRAWTO A+S(R,1),B+S(R,2)
1100 NEXT R
1110 PLOT A,B
1120 DATA 0,0,15,0,0,-5,-15,0,0,0
1130 RETURN
2000 REM FRAME DRAWING SUBROUTINE
2010 REM GOSUB ARRAY READ AND
```

```
2020 REM SCALE SUBROUTINE
2030 LET NOOFPTS=6
2040 GOSUB 6000
2050 REM GOSUB DRAW THE OBJECT SUBROUTINE
2060 GOSUB 7000
2070 DATA 0,0,-15,0,-15,10,15,10,15,0,0,0
2080 RETURN
3000 REM WINDOW DRAWING SUBROUTINE
3010 LET NOOFPTS=6
3020 REM GOSUB ARRAY READ AND
3030 REM     SCALE SUBROUTINE
3040 GOSUB 6000
3050 PRINT "DO YOU WANT WINDOWS? (Y/N)"
3060 INPUT ANS$
3070 IF ANS$<>"Y" THEN 3170
3080 REM GOSUB POINT MOVING SUBROUTINE
3090 GOSUB 5000
3100 REM GOSUB DRAW THE OBJECT SUBROUTINE
3110 GOSUB 7000
3120 PLOT A,B
3130 DATA 0,0,1,0,1,-3,-1,-3,-1,0,0,0
3140 PRINT "ANOTHER WINDOW? (Y/N)"
3150 INPUT ANS$
3160 IF ANS$="Y" THEN 3090
3170 RETURN
4000 REM DOOR DRAWING SUBROUTINE
4010 LET NOOFPTS=6
4020 REM GOSUB ARRAY READ AND
4030 REM SCALE SUBROUTINE
4040 GOSUB 6000
4050 PRINT "DO YOU WANT A DOOR? (Y/N)"
4060 INPUT ANS$
4070 IF ANS$<>"Y" THEN 4150
4080 REM GOSUB POINT MOVING SUBROUTINE
4090 GOSUB 5000
4100 REM GOSUB DRAW THE OBJECT SUBROUTINE
4110 GOSUB 7000
4120 PLOT A,B
4130 DATA 0,0,1.5,0,1.5,-5
4140 DATA -1.5,-5,-1.5,0,0,0
4150 RETURN
```

```
5000 REM POINT MOVING SUBROUTINE
5010 LET A=160
5020 LET B=96
5030 REM PLOTS A WHITE POINT AT (A,B)
5040 COLOR 1
5050 PLOT A,B
5060 GET #1,X
5070 LET Y$=CHR$(X)
5080 REM PLOTS A BLACK POINT AT (A,B)
5090 COLOR 0
5100 PLOT A,B
5110 IF Y$="D" THEN B=B+5
5120 IF Y$="U" THEN B=B-5
5130 IF Y$="L" THEN A=A-5
5140 IF Y$="R" THEN A=A+5
5150 IF Y$="Q" THEN 5170
5160 GOTO 5040
5170 COLOR 1
5180 RETURN
6000 REM ARRAY READ AND
6010 REM SCALE SUBROUTINE
6020 FOR R=1 TO NOOFPTS
6030 FOR C=1 TO 2
6040 READ W
6050 LET ARRAY(R,C)=W
6060 LET S(R,C)=SCALE*ARRAY(R,C)
6070 NEXT C
6080 NEXT R
6090 RETURN
7000 REM DRAWING THE OBJECT SUBROUTINE
7010 PLOT A,B:DRAWTO A+S(1,1),B+S(1,2)
7020 FOR R=2 TO 6
7030 DRAWTO A+S(R,1),B+S(R,2)
7040 NEXT R
7050 RETURN
```

Note: You must be careful where you move the starting point for the house so that lines drawn do not go off the screen. If the program does attempt to draw lines that go off the screen, you will get an error message and you will need to rerun the program.

10–5 PROBLEMS

1. The following sequence of pitch values are for the song **Frere Jacques**. Use them and appropriate pauses to play the song. The grouping is for convenience.

```
121,108,96,121,    121,108,96,121,
96,91,81,    96,91,81,
81,72,81,91,96,121,    81,72,81,91,96,121,
121,162,121,    121,162,121
```

2. What will be output by the following program?

```
100 DIM A(5)
110 FOR I=1 TO 5
120 READ X
130 LET A(I)=X
140 NEXT I
150 DATA 6,2,7,1,3
160 GOSUB 500
170 LET A(3)=10
180 GOSUB 500
190 LET A(5)=8
200 GOSUB 500
210 END
500 FOR I=1 TO 4
510 LET A(I)=A(I+1)
520 NEXT I
530 GOSUB 600
540 RETURN
600 FOR J=1 TO 5
610 PRINT A(J),
620 NEXT J
630 RETURN
```

3. What will be printed if you run the program below?

```
100 LET X=10
110 GOSUB 500
120 PRINT S
130 LET X=X/2
```

```
140 GOSUB 500
150 PRINT S
160 LET X=X+3
170 GOSUB 500
180 PRINT S
190 END
500 LET S=0
510 FOR Y=1 TO X
520 LET S=S+Y
530 NEXT Y
540 RETURN
```

4. Assume that a one-dimensional array Z contains numbers to be added together. The first element of the array Z(1) gives the number of elements that follow in the array and are to be summed. Write a subroutine beginning in line 800 to compute the sum of the elements after Z(1). Assign the sum to the variable T. Terminate the subroutine with a RETURN statement. Assume that the array Z has been properly dimensioned and that the values in the array have been loaded in the main program.

5. X is a one-dimensional array. The first element of the array X(1) gives the number of pieces of data that follow in the array. Write a subroutine beginning in line 500 to search through that array for the largest value. Assign this value to the variable L. Terminate the subroutine with a RETURN statement. Assume that the array X has been properly dimensioned and loaded with numbers elsewhere.

6. Suppose that as part of a printout you need a series of seventy-two * characters in a straight line across the page. Write a subroutine beginning in line 1000 to do this. Terminate the subroutine with a RETURN statement.

7. Assume that a one-dimensional array Y is loaded with numbers. The first element Y(1) gives the number of elements to follow. We want a subroutine to calculate the mean (M) and standard deviation (S) of the numbers in the array that follow the first element. Begin the subroutine in line 900 and terminate with a RETURN statement. The formulas for calculation of the mean and standard deviation are given below.

   Mean = (Sum of values)/N

$$\text{Standard deviation} = \sqrt{\frac{N\times(\text{sum of squares of values}) - (\text{sum of values})^2}{N\times(N-1)}}$$

10—6 PRACTICE TEST

1. Which note will be lower when the following direct mode statements are entered?

```
SOUND 0,96,10,8
SOUND 0,162,10,8
```

2. How do you pass control from the main program to the subroutine?

3. How do you pass control from the subroutine back to the main program?

4. What will be printed out if you run the following program?

```
100 LET A=1
110 GOSUB 200
120 LET A=A+4
130 GOSUB 200
140 LET A=A-2
150 GOSUB 200
160 END
200 REM SUBROUTINE
210 IF A<2 THEN 250
220 IF A=3 THEN 270
230 PRINT "RED"
240 GOTO 280
250 PRINT "WHITE"
260 GOTO 280
270 PRINT "BLUE"
280 RETURN
```

CHAPTER 11

RANDOM NUMBER AND SIMULATIONS

11—1 OBJECTIVES

One of the most interesting applications of computers concerns simulation of events or processes that involve an element of chance. For instance, the computer can simulate gambling games or investigate the number of bank tellers required to ensure that arriving customers do not have to wait more than a few minutes to be served. In this chapter we will see how the computer can be used to handle problems of this type.

Generating Random Numbers

Your ATARI computer has a random-number generator function that is the heart of all programs involving the element of chance, or randomness. You will learn how these random-number generators can be employed in BASIC programs.

Designing Sets of Random Numbers

Sometimes you will need a set of random numbers with characteristics of your own choosing. You will learn how to get the computer to generate numbers in a set you define.

Working with Program Examples

Your programming work in this chapter will involve simulations and applications that involve the element of chance.

11-2 DISCOVERY EXERCISES

Setting up the Random-Number Generator

By their very nature, random-number generators produce sequences of numbers that appear to have no pattern or relationship. If a random-number generator is to be useful, it must produce a a different sequence of numbers each time it is used in a program.

1. Turn on the computer and bring up ATARI BASIC.
2. Enter the following program:

```
100 FOR I=1 TO 9
110 PRINT RND(0)
120 NEXT I
130 END
```

Run the program. Record the largest and smallest numbers that were printed out.

3. Run the program again. Did the same numbers appear?

What was the largest number typed out?

What was the smallest number?

4. Change line 110 as follows.

```
110 PRINT RND(-1)
```

Run the program several times. Were all the numbers different?

Change line 110 as follows.

```
110 PRINT RND(52)
```

Run the program. Were all the numbers different?

5. Clear the program in memory. Enter the following program:

```
100 LET L=.5
110 LET S=.5
120 FOR I=1 TO 100
130 LET X=RND(0)
140 IF X>L THEN 170
150 IF X<S THEN 190
160 GOTO 200
170 LET L=X
180 GOTO 200
190 LET S=X
200 NEXT I
210 PRINT "LARGEST = ";L
220 PRINT "SMALLEST = ";S
230 END
```

This program examines all the numbers generated by the RND function and keeps track of the largest and smallest numbers generated. As the program stands, it will generate 100 random numbers. Run the program and record what was typed out.

6. Type

```
120 FOR I=1 TO 1000
```

Now the program will generate 1000 random numbers. Run the program and record what was printed out. Be prepared to wait about 25 seconds for the output.

Based on what you have seen thus far, what do you think is the largest number that the RND function will generate?

What about the smallest?

7. Now let's go on to some other ideas associated with random numbers. Clear the program in memory and enter the following program:

```
100 FOR I=1 TO 10
110 PRINT INT(2*RND(0))
120 NEXT I
130 END
```

Run the program and record the output.

What were the only two numbers typed out?

8. Type

```
110 PRINT INT(3*RND(0))
```

Display the program. If you run this program, what numbers do you think will be typed out?

Can you predict anything about the sequence or pattern in which the numbers will be typed out?

9. Type

```
110 PRINT INT(2*RND(0)+1)
```

What do you think the program will do now?

Run the program and record the output.

10. Type

```
110 PRINT INT(4*RND(0)+4)
```

If you run the program, what do you think will be printed out?

Run the program and describe the output.

Is there a pattern in the output?

11. Type

```
110 PRINT INT(30*RND(0))/10
```

Display the program and study it carefully. What do you think this program will print out?

Run the program and describe the printout.

12. Type

```
110 PRINT INT(200*RND(0))/100
```

Display the program in memory. What do you think will happen if you run this program?

See if you were right. Run the program and record the output.

13. Turn off the computer. This terminates the computer work for now.

11–3 DISCUSSION

Generating Random Numbers

The details of how random numbers are generated are not important here; it is enough to say that several mathematical methods can be used to produce these numbers. Remember, however, that the RND function calls on the random-number generator. This function is used like the other built-in functions of BASIC that you studied previously, but differs in one important respect.

Recall that the argument of a function (what the function works on) determines the result. Thus, SQR(4) is 2, INT(3.456) is 3, and so forth. The RND function also requires an argument but seems to use no logical rules to generate numbers. Of course, this randomness is precisely the point of the function.

The function generates numbers between 0 and 1 at random. All the numbers in the interval have an equal chance of showing up. Actually, the range of numbers generated is from 0.0000000000 to 0.9999999999. Zero can show up occasionally, but the number 1 never occurs.

- **RND function generates numbers in the range 0 to .9999999999.**

Designing Sets of Random Numbers

Most often the random numbers in the range produced by the RND function are not the most useful. More typically, programs call for random integers (whole numbers) over a certain range or a set of random numbers with a particular set of characteristics. Therefore, it is useful to learn how to generate sets of random numbers with characteristics you can specify.

Let's begin with the characteristics of the RND function. RND delivers numbers in the range 0 to 1, that is, from 0 to slightly less than 1. If you multiply RND by N, you multiply the range of the function by N. Thus, N*RND(0) produces random numbers in the range 0 to N. If you wish, you can shift the numbers (keeping the same range) by adding a number. N*RND(0)+A produces random numbers over the range A to (A+N) or from A to slightly less than (A+N). Also, you can take the integer part of an expression, using the INT function, to produce random integers. The table below shows how the RND function might be used.

BASIC Expression	Result
5*RND(0) + 10	Random numbers in the range 10 to 15
INT(5*RND(0) + 10)	Random integers 10, 11, 12, 13, 14
INT(2*RND(0) + 1)	Random integers 1, 2
100*RND(0)	Random numbers in the range 0 to 100

You may have encountered the notion of mean and standard deviation (see problem 7 in Chapter 10). You can use the RND function to generate numbers that appear to be drawn from a collection of numbers having a given mean and standard deviation. The rule for generating these numbers is

$$X = M+S((\text{sum of 12 numbers from RND function}) - 6)$$

where M and S are the desired mean and standard deviation, respectively. The values of X will appear to come from a collection of numbers with mean M and standard deviation S. The values of X would fall along the bell curve that you have probably seen in textbooks. A subroutine would be very useful in this application.

11–4 PROGRAM EXAMPLES

The following examples illustrate how random numbers can be used. Study these examples carefully and make sure you understand exactly what is taking place.

Example 1 - Flipping Coins

One of the easiest applications of random numbers is a coin-tossing simulation. The goal is to write a program that will produce the following printout:

```
TOSS        OUTCOME
1           H
2           T
3           T
4           H
     etc.
```

The outcome is to be determined randomly for each toss of the coin, with both heads and tails having equal probability. The program should print out the results of ten coin tosses.

The first part of the program generates the heading and the space indicated in the printout above.

```
100 PRINT "TOSS","OUTCOME"
110 PRINT
```

Now open the loop to generate the ten tosses of the coin.

```
120 FOR I=1 TO 10
```

The next step is to generate 0s and 1s randomly. Assume that a 0 signifies heads and a 1 signifies tails. You should be able to convince yourself that the following statement will produce 0s and 1s randomly.

```
130 LET X=INT(2*RND(0))
```

Now analyze X to see whether heads (0) or tails (1) has come up.

```
140 IF X=0 THEN 170
150 PRINT I,"T"
160 GOTO 180
170 PRINT I,"H"
180 NEXT I
```

All that remains now is the END statement.

```
190 END
```

The complete program is listed below.

```
100 PRINT "TOSS","OUTCOME"
110 PRINT
120 FOR I=1 TO 10
130 LET X=INT(2*RND(0))
140 IF X=0 THEN 170
150 PRINT I,"T"
160 GOTO 180
170 PRINT I,"H"
180 NEXT I
190 END
```

Run this program several times and count the number of tails and heads that show up.

Example 2 - Random Integers

The next problem is to write a BASIC program to generate and print out 50 random integers (whole numbers) over the range 10 to 15. The only part of the program that requires much thought is the statement to generate the random integers, so concentrate on this one statement.

Remember that RND generates numbers over the range 0 to 1. Thus, 6*RND(0) will generate numbers in the range 0 to 6. Actually the upper limit is 5.99999999. By using the integer function, you can convert from random numbers to random integers. INT(6*RND) will produce the integers 0, 1, 2, 3, 4, 5 randomly. To get integers in the range 10 to 15, you must add 10. Thus, the expression INT(6*RND(0)) + 10 will produce the desired numbers.

Once this line is figured out, the program follows easily.

```
100 FOR I=1 TO 50
110 LET Y=INT(6*RND(0))+10
120 PRINT Y;" ";
130 NEXT I
140 END
```

Example 3 - Distribution of Random Numbers

Suppose you generate a great number of integers at random over the range 1 to 10. If the random-number generator on the computer is working properly, you would expect to get the same number of each of the integers. If you generated 1000 integers, you would expect to get 100 1s, 100 2s, and so on. The problem is to write a BASIC program that tallies the random integers as they are generated and then prints out the totals. These totals will tell you how well the computer's random-number generator is working.

First, think about how you are going to do the tally. A good way to do this is to use a one-dimensional subscripted array. X(1) will contain the number of 1s generated, X(2) the number of 2s, and so forth up to X(10). Thus, the first task is to dimension the array and set all the values in the array equal to 0.

```
100 DIM X(10)
110 FOR I=1 TO 10
120 LET X(I)=0
130 NEXT I
```

Next open a loop to generate 1000 numbers, generate the random integers, and use the integers as subscripts to increment the appropriate counters in the array.

```
140 FOR I=1 TO 1000
150 LET Y=INT(10*RND(0))+1
160 LET X(Y)=X(Y)+1
170 NEXT I
```

Now all that remains is to print out the contents of the array X.

```
180 FOR J=1 TO 10
190 PRINT J,X(J)
200 NEXT J
210 END
```

The complete program follows:

```
100 DIM X(10)
110 FOR I=1 TO 10
120 X(I)=0
130 NEXT I
140 FOR I=1 TO 1000
150 LET Y=INT(10*RND(0))+1
160 LET X(Y)=X(Y)+1
170 NEXT I
180 FOR J=1 TO 10
190 PRINT J,X(J)
200 NEXT J
210 END
```

Run this program to see how well the random-number generator works. Be prepared to wait about 30 seconds for the results. If you decrease the number of integers generated, expect the distribution of occurrence (the number of 1s, 2s, 3s, etc.) to be more skewed. If you generate more random numbers, however, the distribution will be less skewed.

Example 4 - Random Walk

Use the graphics mode in this program to simulate a random walk on the display screen.

The program below simulates a random walk in a city where all the blocks are of the same size (5). Line 80 places the computer in graphics mode. Line 140 randomly assigns the values 1, 2, 3, and 4 to the variable called WHICHWAY. Lines 150 through 520 use WHICHWAY to determine which way to go on the next part of the random walk.

```
70 REM RANDOM WALK
80 GRAPHICS 8
90 COLOR 1
100 LET A=160
110 LET B=80
120 PLOT A,B
130 FOR I=1 TO 500
140 LET WHICHWAY=INT(4*RND(0))+1
150 ON WHICHWAY GOSUB 200,300,400,500
```

```
160 NEXT I
170 END
200 LET A=A-5
210 DRAWTO A,B
220 RETURN
300 LET A=A+5
310 DRAWTO A,B
320 RETURN
400 LET B=B-5
410 DRAWTO A,B
420 RETURN
500 LET B=B+5
510 DRAWTO A,B
520 RETURN
```

Example 5 - Birthday Pairs in a Crowd

What is the probability that two people in a crowd of fifty have the same birthday? (Consider only the day of the year, not the year of birth.) This famous problem in probability theory has surprising results. You can attack the problem with the following strategy. By generating random integers over the range 1 to 365, you can simulate a birthday for each of the strangers. If you use a one-dimensional array for the birthdays as they are generated, it is easy to check for identical birthdays. Beginning with the first birthday B(1), the program checks to see if that birthday matches any remaining birthdays. Then it does the same thing for B(2), and so on.

First look at the complete program below, then go back and see what is taking place in each line.

```
100 DIM B(50)
110 FOR I=1 TO 50
120 LET B(I)=INT(365*RND(0))+1
130 NEXT I
140 LET F=0
150 FOR I=1 TO 49
160 FOR J=I+1 TO 50
170 IF B(I)<>B(J) THEN 190
180 LET F=F+1
190 NEXT J
200 NEXT I
210 PRINT "NUMBER OF BIRTHDAY PAIRS IS ";F
220 END
```

Of course, line 100 merely dimensions an array for 50 elements. Lines 110 through 130 load the array with random integers selected over the range 1 to 365 inclusive. In line 140, the variable F is set equal to 0. This variable is used to keep track of the number of birthdays to be compared with the rest of the birthdays in the list. The value of I stops at 49 because there must be at least one birthday left in the list against which to compare.

In line 160, the second half of the comparison is set up. J begins at the next value past the current value of I and runs through the rest of the list. The test for a birthday pair is made in line 170. If no match if found, the program jumps to the next value of J. If a match is found, the pair counter is increased by 1 in line 180. The results are printed out in line 210. One problem with the program is that it would record three people having the same birthday as two birthday pairs. Can you figure out a way to fix this?

This is a very interesting program to experiment with. The number of people in the crowd can be modified with simple changes in the program. You can run the program many times to see how many birthday pairs on the average will be found in a crowd of a specified size.

11—5 PROBLEMS

1. Write a program to generate and print out 25 random numbers of the form X.Y where X and Y are digits selected randomly from the set 0, 1 , 2 ,3 9.

2. Write a program to generate and print out 50 integers selected at random from the range 13 to 25.

3. What will be printed out if you run the following program?

```
100 FOR N=1 TO 15
110 PRINT INT(20*RND(0)+1)/100,
120 NEXT N
130 END
```

4. What will be printed out if you run the following program?

```
100 FOR I=1 TO 10
110 PRINT INT(100*RND(0))/10
120 NEXT I
130 END
```

5. Write a program that will simulate tossing a coin, 10, 50, 500, and 1000 times. In each case, print out the total number of heads and tails that occur.

6. Construct a dice-throwing simulation in BASIC. The dice are to be thrown twenty times. At each toss, the program should print out the dice faces that are uppermost.

7. Write a program to generate and print out the average of 100 random numbers selected from the range 0 to 1.

8. Modify the program of example 5 and run it as many times as you need to find how large a crowd must be before there is a 50 percent chance that at least two people in the crowd have the same birthday.

9. John and Bill want to meet at the library. Each agrees to arrive at the library sometime between 1:00 and 2:00 p.m. They further agree that each will wait ten minutes after arriving (but not after 2:00 p.m.). If after ten minutes the other person has not arrived, the person who is waiting will leave. Write a BASIC program to compute the probability that John and Bill will meet. Do a simulation of the problem using the random-number generator.

10. Suppose a basket contains colored golf balls. There are ten red balls, five blue, two green, and eleven yellow. Write a BASIC program to simulate drawing five balls at random from the bucket if they are not replaced after being drawn in sequence.

11. Use the rule given in the discussion section in this chapter to generate and print out 25 numbers selected at random from a bell curve distribution of numbers with mean 10 and standard deviation 2. Round off the numbers to two places past the decimal point.

12. Suppose a soap manufacturer decides to select a five-character brand name. The first, third, and fifth characters are selected at random from the letters BCDFGHJKLMNPQRSTVWXYZ. The second and fourth letters are selected at random from the vowels AEIOU. Use the rules above to write a program that generates and prints out one hundred trial soap names.

11–6 PRACTICE TEST

1. Write a BASIC program to generate and print out 100 random integers selected from the set 1, 2, 3, and 4.
2. Write a BASIC program to generate and print out 100 random numbers over the range 25 to 50.
3. What will be printed out if you run the following program?

```
100 FOR I=1 TO 10
110 LET N=INT(2*RND(0)+1)
120 IF N=1 THEN 150
130 PRINT "WHITE"
140 GOTO 160
150 PRINT "RED"
160 NEXT I
170 END
```

4. What will be printed out if you run the following program?

```
100 FOR J=1 TO 5
110 PRINT INT(1000*RND(0))/100
120 NEXT J
130 END
```

CHAPTER 12

FILES

12–1 OBJECTIVES

Files saved on diskette can be used to store and retrieve collections of information. In this chapter you will learn to use files to manipulate collections of information.

Storing Information to a File

You will learn to create files to store information.

Retrieving Information from a File

Once you store information, you must be able to retrieve it. You will learn methods of retrieving information stored on files.

Modifying Information Stored on Files

It is frequently necessary to change information stored on files. You will learn how to do this.

Working with Examples

You will learn to extract and format information stored on files.

12–2 DISCOVERY ACTIVITIES

In the past, files have been difficult to work with on computers. Fortunately, advances in computer languages have significantly eased the difficulties. You will find that writing programs that use files requires many techniques and concepts discussed in previous chapters.

1. You will need a formatted diskette for this activity. Turn on the disk drive and when the busy light goes out, place the formatted diskette in the drive and close the door. Turn on your computer and bring up ATARI BASIC.

2. Type in the following program.

```
100 DIM NAME$(30)
110 OPEN #1,8,0,"D:GIFT.DAT"
120 PRINT "NAME: ";
130 INPUT NAME$
140 PRINT #1,NAME$
150 PRINT "GIFT AMOUNT ";
160 INPUT AMT
170 PRINT #1,AMT
180 CLOSE #1
190 END
```

   Display the program and check it, especially the punctuation.

3. Run the program. Did the disk whirr?

   _____

   Enter your name and press RETURN . Then enter an appropriate gift amount. Did the disk whirr again?

   _____

   Did the print statements in lines 140 and 170 cause anything to be displayed on the screen?

   _____

   What happened to the information represented by NAME$ and AMT?

   _____

4. Save this program as WRITE. Recall from Chapter 4, step 20 that you can use the SAVE command and the program name:

```
SAVE "D:WRITE"
```

5. Type

```
DOS
```

then type A and press RETURN twice to obtain a list of files and programs on your diskette. Is the program you saved as WRITE and the data file named GIFT.DAT displayed on the screen?

6. Press B RETURN and type in the following program.

```
100 DIM NAME$(30)
110 OPEN #1,4,0,"D:GIFT.DAT"
120 INPUT #1,NAME$,AMT
130 CLOSE #1
140 END
```

Display the program and check it. Run the program. Did the disk whirr?

Was a question mark placed on the screen by line 120?

Was anything displayed on the screen by the program?

7. Add line 125 as follows

```
125 PRINT NAME$,AMT
```

Run the program. The information is read from the file in line 120. Is that enough to make the information useful?

8. Save this program as READ (See step 4.)

9. You must open and close a file each time you used it. To write to a file, use a PRINT statement. To read from a file, use an INPUT statement.

10. Now let's move on to using files with more than one record. Load the program WRITE. Display the program. Add the following lines.

```
115 PRINT "TYPE QUIT FOR NAME WHEN DONE"
135 IF NAME$="QUIT" THEN 180
175 GOTO 120
```

Display the program once again to check its accuracy. Run the program. At the input prompt, enter several names and amounts as you choose. Enter at least three records and amounts above and below 50. (Note: The disk will whirr after four names and amounts have been entered even if you are not finished.) What do you type to end the information entry?

Try it and see if you were correct.

11. Save this new program with the same name: WRITE.

12. Load READ. Display the program. Add the following line.

```
128 GOTO 120
```

What do you think will happen if you run the program?

Run the program and see if you were right.

13. The ERROR 136 error message signifies end of file and can be avoided by adding the following line.

```
122 TRAP 130
```

Run the program. Did the error message occur this time?

14. Now extract some information from your file. Display the program. Add the following line.

```
121 IF AMT<50 THEN 128
```

Run the program. Were all the names printed out?

15. You can also compile information from the file. Display the program. Change lines 121 and 140 and add lines 90 and 150 as follows:

```
121 IF AMT>50 THEN C=C+1
140 PRINT "THERE ARE ";C;" GIFTS OVER 50"
90 LET C=0
150 END
```

Now delete line 125. Run the program. Does the number displayed agree with the information you wrote out to file?

You may wish to display the program and study it.

16. Now let's add some things up. Load READ once again. Display the program. Add the following lines.

```
90 LET SUM=0
123 LET SUM=SUM+AMT
135 PRINT "TOTAL GIFT SUM IS ";SUM
```

Delete lines 125 and 140. Display the program. Run the program. What was the total of the gifts in the information you entered?

17. That ends the discovery exercises. Turn off the computer and go on to the discussion.

12-3 DISCUSSION

Several BASIC statements can be used with files. The ones you have learned are flexible enough to cover almost every use of files. As you saw in the discovery exercises, the OPEN statement prepares a file so that information can be written to or read from it. The PRINT statement causes information to be written to a file that has been opened for output. The INPUT statement allows information to be read from a file opened for input. All files that are opened must be closed before you run a program or before you change the status of the file. For instance, you must use the CLOSE statement to change file status from open to accept information to open to be read.

Storing Information to a File

To access a file, you must use the OPEN statement. For example,

```
110 OPEN #1,8,0,"D:FILEONE"
```

prepares FILEONE to have information written to it. The statement will create the file, if necessary.

The #1 in line 110 is called an input/output control block (IOCB). For the purposes of this book, you need to know only that an IOCB is necessary in every OPEN statement and that you can freely use IOCB's #1 through #5.

The second number in the OPEN statement, 8 in the example above, determines the operation to be performed with the file. The number 4 indicates an input operation and prepares the file to have information read from it. The number 8 indicates an ouput operation and prepares the file to have information written to it. The number 9 also signifies an output operation but prepares the file to have information added to it.

In this book, the third number is always 0. The only other value allowed, 83, causes the printer (if one is attached) to print sideways.

The last portion of the OPEN statement indicates which file is to be used.

Once a file is opened for output, you can write to the file. To write to a file, a PRINT statement is required. For example,

```
110 OPEN #1,8,0,"D:FILEONE"
120 PRINT #1;NAME$
```

will set up FILEONE to accept data from subsequent PRINT statement. When you write to a file, the PRINT statements you use must have an IOCB. The data contained in NAME$ will be written to a record in the file. The information items will be written out to the file in the order that they appear in the program.

In the following example, a file is opened and some information is written to it.

```
100 DIM NAME$(25),GRADE$(1)
110 OPEN #1,8,0,"D:QUIZ.DAT"
120 PRINT "NAME: "
130 INPUT NAME$
140 IF NAME$ = "QUIT" THEN END
150 PRINT #1;NAME$
160 PRINT "LETTER GRADE: "
170 INPUT GRADE$
180 PRINT #1;GRADE$
190 GOTO 120
```

In line 110 the file QUIZ.DAT is opened (created if necessary). Lines 150 and 180 write the information, in this case the student's name and grade, to the file QUIZ.DAT. The .DAT appended to the file name QUIZ is an extension allowed in file names. You can extend the file name with one to three letters following a period. This addition to the name can be very helpful when you display the file directory; for instance the extension DAT to the right of the file name helps you see at a glance which files contain data.

Also note that there is no CLOSE statement in this example. The END statement closes all files that are open. However, files are often opened and closed several times in a program. The CLOSE statement is necessary because the END statement does not close files until the program stops running.

When you use the number 8 to open a file for output, any information in that file is lost. Therefore, use number 8 only when you don't need the information in the file or when the file is empty. However, if you wish to add information to a file, then use a statement such as

```
150 OPEN #1,9,0,"D:FILEONE"
```

This statement prepares FILEONE for output, and all information written to the file will be added to the end of information that is already on it. You cannot use this option unless FILEONE already exists.

Retrieving Information from a File

To read information from a file, you must use an OPEN statement and INPUT statements, as in the following example.

```
100 DIM NAME$(25),GRADE$(1)
110 OPEN #1,4,0,"D:QUIZ.DAT"
120 TRAP 160
130 INPUT #1;NAME$,GRADE$
140 PRINT NAME$,GRADE$
150 GOTO 120
160 END
```

Recall that without line 140, nothing will be displayed on the screen. The INPUT statement in line 130 only reads information into the variables. What you instruct the program to do with the variables depends on the outcome you desire.

The TRAP statement in line 120 traps the end-of-file error that will occur in line 130. Instead of displaying the error message, and stopping the program, the computer will branch to line 160. Other types of errors can be treated in this way as well because the TRAP statement takes control of the program when an INPUT error occurs and directs execution to a specified line number.

You have been using sequential files throughout this chapter. Sequential files are files on which records (information) are placed one after another. When reading from a sequential file, you must start at the beginning of the file and read each record until you reach the one you wish to process. Another kind of file called a random access file is also available on your ATARI computer. Random access files allow you to read from and write to any record by indicating which record you want to use. They are, however, more difficult to learn than are sequential files. Sufficient versitality of sequential files will be seen in the examples given below. Certain tasks are better handled with sequential files than with random access files. Information about random access files can be found in your *Disk Operating System II Reference Manual.*

12—4 PROGRAM EXAMPLES

The following set of programs can be used to create and maintain a mailing list for advertising and billing purposes.

Example 1 - Mail List Data Entry Program

A mailing list contains names, addresses, and other information about individuals. Thus, the program might request the following items.

```
FIRST NAME: (You type in)
LAST NAME: (You type in)
STREET: (You type in)
CITY: (You type in)
ZIP CODE: (You type in)
BALANCE: (You type in)
```

You would request this information about every person to be placed on the mailing list. Termination of input could be accomplished by entering QUIT for the first name. The information would then be stored to a file called MAIL.DAT.

The complete program follows.

```
80 REM MAIL LIST DATA ENTRY
90 REM REPLY WITH N TO CREATE A NEW FILE
100 DIM FIRST$(10),LAST$(15)
110 DIM STREET$(25),CITY$(15),ANS$(3)
120 PRINT "DO YOU WISH TO ADD DATA"
130 PRINT "TO THE CURRENT FILE? (Y/N) ";
140 INPUT ANS$
150 IF ANS$="Y" THEN 180
160 OPEN #1,8,0,"D:MAIL.DAT"
170 GOTO 190
180 OPEN #1,9,0,"D:MAIL.DAT"
190 PRINT "USE QUIT FOR FIRST NAME TO END
200 PRINT "FIRST NAME: ";
210 INPUT FIRST$
220 IF FIRST$="QUIT" THEN 400
230 PRINT #1;FIRST$
240 PRINT "LAST NAME: ";
250 INPUT LAST$
```

```
260 PRINT #1;LAST$
270 PRINT "STREET: ";
280 INPUT STREET$
290 PRINT #1;STREET$
300 PRINT "CITY: ";
310 INPUT CITY$
320 PRINT #1;CITY$
330 PRINT "ZIP CODE: ";
340 INPUT ZIP
350 PRINT #1;ZIP
360 PRINT "PAYMENT BALANCE: ";
370 INPUT BAL
380 PRINT #1,BAL
390 GOTO 190
400 CLOSE #1
410 END
```

Save this program under the name DATAENTR. You will be using it again.

Example 2 - Mailing Label Program

This program will use the MAIL.DAT file to create mailing labels The labels should have three lines as shown below.

```
GEORGE JONES
1234 DATAFILE DRIVE
SAN JOSE, CA 95009
```

The complete program follows.

```
90 REM MAILING LABEL PROGRAM
100 DIM FIRST$(10),LAST$(15)
110 DIM STREET$(25),CITY$(15)
120 OPEN #1,4,0,"D:MAIL.DAT"
130 TRAP 230
140 INPUT #1;FIRST$,LAST$
150 INPUT #1;STREET$,CITY$
160 INPUT #1;ZIP,BAL
170 PRINT FIRST$;" ";LAST$
180 PRINT STREET$
190 PRINT CITY$;", CA ";ZIP$
```

```
200 PRINT
210 PRINT
220 GOTO 140
230 END
```

Save this program under the name LABELS.

Example 3 - Selected Labels Program

It is fairly easy to modify the previous program to select records in a file based on a condition. In this example, the program generates labels for bills by selecting those customers with outstanding balances. You modify the program in example 2 simply by changing line 90 and adding line 165 so that the complete program reads as follows.

```
90 REM SELECTED LABELS PROGRAM
100 DIM FIRST$(10),LAST$(15)
110 DIM STREET$,CITY$
120 OPEN #1,4,0,"D:MAIL.DAT"
130 TRAP 230
140 INPUT #1;FIRST$,LAST$
150 INPUT #1;STREET$,CITY$
160 INPUT #1;ZIP,BAL
165 IF BAL=0 THEN 140
170 PRINT FIRST$;" ";LAST$
180 PRINT STREET$
190 PRINT CITY$;", CA ";ZIP$
200 PRINT
210 PRINT
220 GOTO 140
230 END
```

Save this program under the name LABSEL.

Example 4 - Modifying the MAIL.DAT file

This program will allow you to modify any of the items in a record in the MAIL.DAT file. In order to do this, the program must know which record and item you wish to change. The program should ask for a last name and then display all the items for a record where the LAST$ matches the last name you enter. then it should ask you

to modify one or more items in the record if the record you wish to modify is being displayed. If not, the program should continue to display the other records you wish to modify.

The complete program follows.

```
90 REM MODIFY A RECORD IN THE MAIL LIST
100 DIM FIRST$(10),LAST$(15)
110 DIM STREET$(25),CITY$(15),ANS$(3)
120 OPEN #1,4,0,"D:MAIL.DAT"
130 OPEN #2,8,0,"D:TEMP.DAT"
140 PRINT "PLEASE ENTER THE LAST"
150 PRINT "NAME OF THE RECORD"
160 PRINT "YOU WISH TO MODIFY"
170 INPUT NAME$
180 INPUT #1;FIRST$,LAST$,STREET$
190 INPUT #1;CITY$,ZIP,BAL
200 IF NAME$=LAST$THEN 230
210 GOSUB 820
220 GOTO 180
230 PRINT "1.  ";FIRST$
240 PRINT "2.  ";LAST$
250 PRINT "3.  ";STREET$
260 PRINT "4.  ";CITY$
270 PRINT "5.  ";ZIP
280 PRINT "6.  ";BAL
290 PRINT "7.  EXIT"
300 PRINT "ENTER THE NUMBER OF THE"
310 PRINT "ITEM YOU WISH TO MODIFY ";
320 INPUT N
330 IF N=7 THEN 350
340 ON N GOSUB 700,720,740,760,780,800
350 PRINT "DO YOU WISH TO MODIFY ANOTHER"
360 PRINT "ENTRY IN THIS RECORD? (Y/N) ";
370 INPUT ANS$
380 IF ANS$="Y" THEN 230
385 TRAP 530
390 PRINT #2;FIRST$:PRINT #2;LAST$
400 PRINT #2;STREET$:PRINT #2;CITY$
410 PRINT #2;ZIP:PRINT #2;BAL
420 PRINT "MODIFY ANOTHER RECORD? ";
430 INPUT ANS$
440 IF ANS$="Y" THEN 140
450 TRAP 530
```

```
460 INPUT #1;FIRST$:PRINT #2;FIRST$
470 INPUT #1;LAST$:PRINT #2;LAST$
480 INPUT #1;STREET$:PRINT #2;STREET$
490 INPUT #1;CITY$:PRINT #2;CITY$
500 INPUT #1;ZIP:PRINT #2;ZIP
510 INPUT #1;BAL:PRINT #2;BAL
520 GOTO 460
530 CLOSE #1:CLOSE #2
540 OPEN #1,8,0,"D:MAIL.DAT"
550 OPEN #2,4,0,"D:TEMP.DAT"
560 TRAP 640
570 INPUT #2;FIRST$:PRINT #1;FIRST$
580 INPUT $2;LAST$:PRINT #1;LAST$
590 INPUT #2;STREET$:PRINT #1;STREET$
600 INPUT #2;CITY$:PRINT #1;CITY$
610 INPUT #2;ZIP:PRINT #1;ZIP
620 INPUT #2;BAL:PRINT #1;BAL
630 GOTO 560
640 END
700 PRINT "FIRST NAME: ";
705 INPUT FIRST$
710 RETURN
720 PRINT "LAST NAME: ";
725 INPUT LAST$
730 RETURN
740 PRINT "STREET: ";
745 INPUT STREET$
750 RETURN
760 PRINT "CITY: ";
765 INPUT CITY$
770 RETURN
780 PRINT "ZIP CODE: ";
785 INPUT ZIP
790 RETURN
800 PRINT "BALANCE: ";
805 INPUT BAL
810 RETURN
820 PRINT #2;FIRST$:PRINT #2;LAST$
830 PRINT #2;STREET$:PRINT #2;CITY$
840 PRINT #2;ZIP:PRINT #2;BAL
850 RETURN
```

Save this program under the name MODIFY.

Example 5 - Menu-Driven Mailing Program

This program will give you a menu to select any of the previous activities. It will allow you to use this set of programs to maintain a mailing list for use in billing individuals. This is a very effective way to incorporate several program modules into one short program that will accomplish a desired task.

The program should begin by printing the possible activities that you can select. The program might print

```
1.  ADD A NEW RECORD
2.  COMPLETE SET OF LABELS
3.  SELECTED SET OF LABELS
4.  MODIFY A RECORD
5.  EXIT THE PROGRAM
```

The complete program follows.

```
100 PRINT "1.  ADD A NEW RECORD"
110 PRINT "2.  COMPLETE SET OF LABELS"
120 PRINT "3.  SELECTED SET OF LABELS"
130 PRINT "4.  MODIFY A RECORD"
140 PRINT "5.  EXIT THE PROGRAM"
150 PRINT "CHOOSE A NUMBER ";
160 INPUT N
170 IF N = 5 THEN END
180 ON N GOTO 300,400,500,600
300 RUN "D:DATAENTR"
400 RUN "D:LABELS"
500 RUN "D:LABSEL"
600 RUN "D:MODIFY"
```

Save this program under the name MENU.

If you want the program to return to the main menu-driven program above after it has called for the execution of another program, modify the END statement in each of the programs DATAENTR, LABELS, LABSEL, and MODIFY. For instance, you would add the following two lines

```
410 RUN "D:MENU"
420 END
```

to DATAENTR program. These same lines, but with different line numbers, should replace the END statements in the remaining programs.

12–5 PROBLEMS

1. Design an appropriate record structure for a file that will be used to index your cassette tape collection. Determine valid ATARI BASIC variable names for each item in the record structure.
2. Design an appropriate record structure for a file that will be used to keep track of your checkbook entries.
3. Design an appropriate record structure for a file that will be used to inventory your household possessions.
4. Write out an appropriate record structure for a file that will be used to keep your personal mailing list. Remember that birthdays and anniversaries are important to your friends.
5. Write a program that will use a file called CHARGE to manage your charge cards. Each record should have the following structure.

Variable	Description	Approximate Length
CARD$	Name of Card	20
NAME$	Name of Store	30
DATE$	Date of Purchase	10
DES$	Description of Purchase	50
AMT	Amount of Purchase	8

   The program should allow you to total the amount of money you have charged to each card in the entire file.
6. Write a program that uses the record structure from problem 4 to manage your personal mailing list. The program should allow you to print out labels for your Christmas cards and for messages to your friends at work.

12–6 PRACTICE TEST

1. If you run the following program:

```
100 DIM NAME$(20),MESSAGE$(50)
110 OPEN #2,8,0,"D:FILETWO"
120 INPUT NAME$
130 PRINT #2;NAME$
140 INPUT MESSAGE$
150 PRINT $2;MESSAGE$
160 CLOSE #2
170 END
```

a. What file will be used?

b. How many characters are allowed in each variable name?

c. How many items are placed in the file?

2. Write a program line that opens a file named TRIAL.DAT. The line should allow information to be written to the file.

3. Write a program that will read and print the three information items from variables A$, B$, and C$. Each variable has a length of 10. The variables are in a file named TEST1.DAT. Be sure to close the file.

4. What is wrong with the following program line?

```
200 OPEN #1,4,0,"D,FILEONE"
```

5. Will the following program line allow information to be added to the file FILEONE?

```
200 OPEN #2,9,0,"D:FILEONE"
```

APPENDIX A

GLOSSARY

ABS(X) A BASIC function that takes the absolute value of X. Positive values of X remain positive. Negative values of X become positive.

Arithmetic Operators Addition +, subtraction , multiplication *, division /, and exponentiation ^.

ASC(A$) A BASIC function that converts the first character in A$ to its equivalent position number in the ASCII character set.

BASIC An acronym for "Beginners All-Purpose Instruction Code". More people know how to program computers in BASIC than any other language.

CHR$(N) A BASIC function that returns the Nth character from the ASCII character set.

CLOSE A file statement that terminates access to a text file on the diskette in the disk drive.

COLOR A statement that chooses the color register to be used in color graphics.

Control Characters These are characters typed on the keyboard while holding down the CTRL or CONTROL key. They are used to send special signals to the computer.

Cursor A square displayed on the TV screen that shows where the next typed character will be displayed.

DATA A statement used to hold information within a program. This information is called for with the READ statement.

Deleting BASIC Statements Type the line number of the statement to be deleted and then press RETURN .

DIM A statement used to specify the size and reserve space for arrays.

Double Subscripts Indicated within parentheses following a variable name, and separated by a comma. Used to specify a row and column number in an array. A(3,5), for example, means the element in the two dimensional array A at row 3 and column 5.

DOS Diskette A diskette that has been formatted using the DOS disk operating system. These diskettes can be used to store programs and files.

DRAWTO A statement that draws a line from the current position of the grtaphics pen to the point specified.

Editing Making corrections or changes in a program or data.

END Marks the end of a BASIC program or the end of the main program.

E Notation A notation used in BASIC to express either very large or very small numbers.

FILE A collection of information that is created and used by the ATARI BASIC file statements.

FOR NEXT Statements used in BASIC to set up loops.

GET A statement that calls for information from the keyboard without placing a question mark on the screen.

GOSUB A statement used to transfer program control to a subroutine.

GOTO An unconditional branch statement.

INPUT A statement that calls for input of information from the keyboard.

Inserting BASIC Statements Type in the statement using a line number not already in use.

IF THEN A conditional branch statement.

INT(X) A BASIC function that takes the integer part of X. The integer part of X is defined as the first integer less than or equal to X.

LEN(A$) A BASIC function used to determine the length of a string in characters. For example, if A$ = "DOG" then LEN(A$) is 3.

LET Identifies an assignment statement. It is always followed by a variable name, an equal sign, and a BASIC expression. The LET in the assignment statement is optional.

LIST A command used to display the program in memory.

LOAD A command that loads a file from a diskette to the memory.

NEW A command that erases the current program in memory.

Numeric Variable Names ATARI BASIC allows variable names up to 110 characters long. The first character must be a letter.

ON N GOSUB A statement that branches to one of several subroutines depending upon the value of N.

ON N GOTO A statement that branches to one of several numbered lines depending upon the value of N.

OPEN A file statement that opens the specified file for input or output operations.

PLOT A statement that plots a point at the position specified

PRINT A statement that sends information from the computer to the screen.

Random Numbers A sequence of numbers generated by the RND function. They appear to have no pattern or relationship to one another.

READ A statement that calls for input of information stored in DATA statements within the program.

RECORD A record is a collection of information stored in a text file created and used by the ATARI BASIC file statements.

Replacing BASIC Statements Retype the statement to be replaced including the line number.

RETURN A statement used to transfer program control back from a subroutine to the main program.

RND A BASIC function used to generate random numbers.

RUN A command used to tell the computer to begin execution of the program in memory.

SAVE A command that saves a program from memory to diskette. For example, SAVE "D:AVERAGE" would save the program currently in memory to the diskette in the disk drive under the name AVERAGE.

SETCOLOR Stores the hue and luminance color data in a particular color register.

SGN(X) A BASIC function that determines the sign of X. SGN(X) is +1, 0, −1 as X is positive, zero, or negative respectively.

Single Subscripts Indicated within parentheses following a variable name. Used to specify a particular element in an array. A(6), for example, means the sixth element of the one dimensional array A.

SOUND Controls the register, sound pitch, distortion, and volume of a tone or note.

SQR(X) A BASIC function that takes the square root of X. X cannot be negative.

String Variable Names BASIC string variable names are allowed to be up to 110 characters long. They must start with a letter and end with a $ sign.

System Master Diskette A DOS diskette that contains a number of useful programs.

APPENDIX B

PRACTICE TEST SOLUTIONS

Chapter 1

1. Press the RETURN key.
2. Press the RESET or SYSTEM RESET key.
3. *
4. Press the CLEAR key.
5. Division.
6. The number 2 will be displayed on the screen.
7. The string 25/5+2 will be displayed on the screen.
8. Move the cursor to the G in PRING with the CTRL or CONTROL key and the left arrow key (←). Then type in a T and press RETURN

Chapter 2

1. Press the RETURN key.
2. Press the BREAK key.
3. Press BREAK

4. The statement `PRINT C` has no line number.

5. The number 2 would be displayed on the screen.

6. Up to 110 characters.

7. Type the line using a line number not already in the program.

8. Retype the line including the line number.

9. Type the line number and press RETURN .

10. Type LIST and press RETURN .

11. Press the CLEAR key.

12. Type RUN and press RETURN .

13. Type NEW and press RETURN .

14. A character string variable always ends with a $.

Chapter 3

1. A square with its center at the center of the screen.

2. A house with the peak of the roof at the center of the screen.

3.
```
GRAPHICS 8:PLOT 160,80
DRAWTO 280,80:DRAWTO 280,20
DRAWTO 220,20:DRAWTO 220,80
```

4. **a.** `PLOT 160,80` **b.** `PLOT 0,0` **c.** `PLOT 160,0`
d. `PLOT 240,150`

5. **a.** By typing `GRAPHICS 8` **b.** `COLOR 1`

6. Moves a line segment down the middle of the screen

Chapter 4

1. **a.** * **b.** ^ **c.** /

2. **a.** Exponentiation **b.** Multiplication and division **c.** Addition and subtraction

3. Left to right.

4. `100 LET A = (4+3*B/D)^2`

5. 4

6. **a.** 5.16E+06 **b.** 3.14E 05

7. **a.** 7258000 **b.** 0.001437

8. / then + then ^

9. **a.** Type `LOAD "D:(name of program)"` and press RETURN .
   b. Type `SAVE "D:(name of program)"` and press RETURN .
   c. Type `DOS`, THEN `D` and follow directions.
   d. Type `NEW` and press RETURN .
   e. Type `LIST` and press RETURN .
   f. Type `RUN "D:(name of program)"` and press /RETURN.
   g. Type `DOS`, then `A` and press RETURN twice.

10. Use the left arrow key to move the cursor to the error and type the correct character and press RETURN . Recall the CTRL key is required with the left arrow.

Chapter 5

1.
```
1      2      3      4
  5      6      7      8
    9      10     11     12
```
etc.

2. **a.** By assignment (e.g., `100 LET A=3`) **b.** INPUT statements
   c. READ and DATA statements

3. A string.

4. To provide information within the program for the benefit of the programmer or user.

5. DATA.

6. Y = 3 will be printed out.

7. Four

8. As many as needed.

9.
```
1          3
13
```

10.
```
22
READY
```

11.
```
100 PRINT "INPUT NO. OF MILES ";
110 INPUT N
120 LET K=1.609*N
130 PRINT N;" MILES EQUALS "
140 PRINT K;" KILOMETERS"
150 END
```

Chapter 6

1.
```
6
10
14
18
```

2.
```
BEST

BETTER
BEST

GOOD
BETTER
BEST

ERROR 6   AT LINE 100
```

3.
```
100 PRINT "HOW MANY WIDGETS";
110 INPUT N
120 IF N<=20 THEN 160
130 IF N<=50 THEN 180
140 LET U=1.5
150 GOTO 190
160 LET U=2
170 GOTO 190
180 LET U=1.8
190 LET P=N*U
200 PRINT "PRICE PER WIDGET IS ";U
210 PRINT "TOTAL COST OF ORDER IS ";P
220 PRINT
230 GOTO 100
240 END
```

4.
```
100 LET X=0
110 PRINT X,X+5,X+10
120 LET X=X+15
130 IF X<=175 THEN 110
140 END
```

5.
```
100 PRINT "WHAT WAS THE SPEED LIMIT? ";
110 INPUT A
120 PRINT "SPEED ARRESTED AT? ";
130 INPUT B
140 LET X=B-A
150 IF X>40 THEN F=80
160 IF X<=40 THEN F=40
170 IF X<=30 THEN F=20
180 IF X<=20 THEN F=10
190 IF X<=10 THEN F=5
200 PRINT "FINE IS ";F;" DOLLARS"
210 END
```

Chapter 7

1.
```
20      18      16      14
  12      10      8       6
    4       2
```

2.
```
1      2      3      2
  4      6      3      6
    9      4      8      12
```

3. **a.** 6 **b.** 7 **c.** 22.8 **d.** −1

4. The loops are crossed.

5.
```
100 PRINT "MILES","KILOMETERS"
110 PRINT "-----","----------"
120 FOR M=10 TO 100 STEP 5
130 PRINT M,1.609*M
140 NEXT M
150 END
```

6.
```
100 DATA 10
110 DATA 25,21,24,21,26,27,25,24,23,24
120 READ N
130 LET S=0
140 FOR I=1 TO N
150 READ X
160 LET S=S+X
170 NEXT I
180 PRINT S/N
190 END
```

7. **a.** A triangle **b.** 28 **c.** across

Chapter 8

1. To save space for an array.

2. X(3,4)

3.
```
100 DIM A(50)
110 PRINT "HOW MANY NUMBERS "
120 INPUT N
130 PRINT "WHAT ARE THE NUMBERS "
140 FOR I=1 TO N
150 INPUT X
160 LET A(I)=X
170 NEXT I
```

```
180 LET S=0
190 FOR I=1 TO N
200 IF A(I)<=0 THEN 220
210 LET S=S+A(I)
220 NEXT I
230 PRINT "SUM OF POSITIVE ELEMENTS IS ";S
240 END
```

4.
```
90 DIM X(4,6)
100 FOR R=1 TO 4
110 FOR C=1 TO 6
120 LET X(R,C)=4
130 NEXT C
140 NEXT R
150 END
```

5.
```
2 0 0 0 0
0 2 0 0 0
0 0 2 0 0
0 0 0 2 0
0 0 0 0 2
```

6. **a.** `100 DIM A(2,3)` **b.** A(2,3) = 4 **c.** A(X,Y) = A(1,2) = 3
d. A(A(1,1), A(2,2)) = A(1,2) = 3

Chapter 9

1. By appending $ to a numeric variable name.

2. False

3. `A$(5,13)`

4.
```
90 DIM A$(50)
100 INPUT A$
110 FOR X=LEN(A$) TO 1 STEP -1
120 PRINT A$(1,X)
130 NEXT X
140 END
```

5.
```
A
AB
ABC
ABCD
ABCDE
```

etc.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Chapter 10

1. The second note.

2. Type in GOSUB (line number at beginning of subroutine.)

3.
```
RETURN
```

4.
```
WHITE
RED
BLUE
```

Chapter 11

1.
```
100 FOR I=1 TO 100
110 LET X=INT(4*RND(0)+1)
120 PRINT X,
130 NEXT I
140 END
```

2.
```
100 FOR I=1 TO 100
110 LET X=25+25*RND(0)
120 PRINT X,
130 NEXT I
140 END
```

3. The output will be randomly selected from WHITE and RED. Three program outputs are shown to indicate the random nature of the process.

```
(1)       (2)       (3)
RED       WHITE     WHITE
RED       WHITE     RED
WHITE     RED       WHITE
WHITE     WHITE     WHITE
RED       WHITE     WHITE
RED       RED       WHITE
RED       RED       RED
WHITE     RED       WHITE
RED       WHITE     WHITE
RED       WHITE     RED
```

4. Five random numbers of the form X.XX over the range 0.00 to 9.99. Three program outputs are shown below to illustrate the random nature of the process.

```
 (1)       (2)       (3)
0.51      6.69      1.15
9.34      4.04      8.87
9.08      9.06      9.26
9.26      6.71      2.59
5.98      8.15      3.05
```

Chapter 12

1. **a.** FILETWO **b.** 20 and 50 **c.** 2

2.
```
110 OPEN #1,8,0,"D:TRIAL.DAT"
```

3.
```
100 DIM A$(10),B$(10),C$(10)
110 OPEN #1,4,0,"D:TEST1.DAT"
120 INPUT #1;A$,B$,C$
130 PRINT A$,B$,C$
140 GOTO 120
150 END
```

Good programming practice would call for the statement TRAP 150 just before line 120 to avoid an error message and to close the file smoothly.

4. The correct program line is

```
200 OPEN #1,4,0,"D:FILEONE"
```

5. Yes, the second number, 9, is exactly the value needed to add to a file.

APPENDIX C

SOLUTIONS TO ODD-NUMBERED PROBLEMS

Chapter 5

1.
```
100 REM CHAP 5, PROB 1
110 READ A,B,C,D
120 DATA 10,9,1,2
130 LET S=A+B
140 LET P=C*D
150 PRINT S,P
160 END
```

3.
```
100 REM CHAP 5, PROB 3
110 READ A,B,C,D
120 DATA 21,18,6,3
130 PRINT A
140 PRINT B
150 PRINT C
160 PRINT D
170 END
```

5. There is no value assigned to C.

7.
```
100 REM CHAP 5, PROB 7
110 PRINT "CASH = ";
120 INPUT C
```

```
130 PRINT "MARKETABLE SECURITIES = ";
140 INPUT M
150 PRINT "RECEIVABLES = ";
160 INPUT R
170 PRINT "LIABILITIES = ";
180 INPUT L
190 LET A=(C+M+R)/L
200 PRINT "ACID-TEST RATIO = ";A
210 END
```

9. The program loops back to line 100 where A is set equal to 1 after each printout. The program can be corrected by changing line 130 as follows.

```
130 GOTO 110
```

11. The problem lies in statements 100, 110, and 120. The values of L, W, and H are supposed to be printed out, but they haven't been defined. The computer will assign the value zero to the three variables and these zeros are printed out by lines 100, 110, 120. The program may be corrected by deleting line 130 and L, W, and H at the ends of lines 100, 110, and 120. Now, the following lines need to be added.

```
105 INPUT L
115 INPUT W
125 INPUT H
```

13.
```
100 REM CHAP 5, PROB 13
110 DATA 21423,21493,5
120 DATA 5270,5504,13
130 DATA 65214,65559,11.5
140 READ R1,R2,G
150 LET M=(R2-R1)/G
160 PRINT M
170 GOTO 140
180 END
```

15.
```
100 REM CHAP 5, PROB 15
110 DATA 92,63,75,82,72,53,100,89,70,81
120 READ A,B,C,D,E,F,G,H,I,J
130 PRINT (A+B+C+D+E+F+G+H+I+J)/10
140 END
```

17.
```
100 REM CHAP 5, PROB 17
110 PRINT "QUOTED INTEREST RATE (PERCENT) "
120 INPUT R
130 PRINT"NUMBER OF TIMES COMPOUNDED PER YEAR'
140 INPUT M
150 LET T=((1+R/(100*M))^M-1)*100
160 PRINT "TRUE ANNUAL INTEREST RATE IS"
170 PRINT T
180 END
```

19.
```
100 REM CHAP 5, PROB 19
110 PRINT "INITIAL INVESTMENT ";
120 INPUT P
130 PRINT "ANNUAL INTEREST RATE(%) ";
140 INPUT I
150 PRINT "YEARS LEFT TO ACCRUE INTEREST ";
160 INPUT N
170 LET T=P*(1+I/100)^N
180 PRINT "TOTAL VALUE IS ";T
190 END
```

Chapter 6

1.
```
100 REM CHAP 6, PROB 1
110 INPUT X,Y
120 IF X>Y THEN 150
130 PRINT Y
140 GOTO 160
150 PRINT X
160 END
```

3.
```
100 REM CHAP 6, PROB 3
110 LET S=0
120 LET X=1
130 LET S=S+X
140 LET X=X+1
150 IF X<=100 THEN 130
160 PRINT S
170 END
```

5.
```
ERROR-    6 AT LINE 120
```

7.
```
100 REM CHAP 6, PROB 7
110 LET S=0
120 READ X
130 IF X=9999 THEN 180
140 IF X<-10 THEN 120
150 IF X>10 THEN 120
160 LET S=S+X
170 GOTO 120
180 PRINT S
190 DATA -1,22,17,-6,4,7,9999
200 END
```

9.
```
100 REM CHAP 6, PROB 9
110 LET C=1
120 LET T=0
130 LET W=1
140 LET T=T+W
150 LET C=C+1
160 LET W=2*W
170 IF C<=22 THEN 140
180 PRINT T/100
190 END
```

11. The number 83 will be printed out. The program finds the largest number contained in the two DATA statements.

13.
```
100 REM CHAP 6, PROB 13
110 PRINT "LIST PRICE ($) ";
120 INPUT L
130 PRINT "DISCOUNT RATE (%) ";
140 INPUT R
150 LET D=L*(1-R/100)
160 PRINT "DISCOUNTED PRICE IS"
170 PRINT D;" DOLLARS"
180 END
```

15.
```
100 REM CHAP 6, PROB 15
110 INPUT A,B
120 IF A>=10 THEN 130
121 GOTO 150
130 IF B>=10 THEN 140
131 GOTO 150
140 PRINT A+B
```

```
141 GOTO 210
150 IF A<10 THEN 160
151 GOTO 180
160 IF B<10 THEN 170
161 GOTO 180
170 PRINT A*B
171 GOTO 210
180 IF A<B THEN 200
190 PRINT A-B
191 GOTO 210
200 PRINT B-A
210 END
```

17.
```
100 REM CHAP 6, PROB 17
110 PRINT "GROWTH RATE (%) ";
120 INPUT R
130 LET N=0
140 LET Q=1
150 LET Q=Q*(1+R/100)
160 LET N=N+1
170 IF Q<=2 THEN 150
180 PRINT "NUMBER OF GROWTH PERIODS TO DOUBLE
IS ";N
190 END
```

19.
```
80 REM CHAP 6, PROB 19
90 DIM Y$(1)
100 OPEN #1,12,0,"K:"
110 GRAPHICS 8
120 LET A=160
130 LET B=96
140 COLOR 1
150 PLOT A,B
160 DRAWTO A,B
170 GET #1,X
180 LET Y$=CHR$(X)
190 IF Y$="D" THEN B=B+10
200 IF Y$="U" THEN B=B-10
210 IF Y$="L" THEN A=A-10
220 IF Y$="R" THEN A=A+10
230 IF Y$="Q" THEN 250
```

```
240 GOTO 160
250 END
```

Chapter 7

1.
```
100 REM CHAP 7, PROB 1
110 PRINT "N","SQR(N)"
120 PRINT
130 FOR N=2 TO 4 STEP .1
140 PRINT N,SQR(N)
150 NEXT N
160 END
```

3.
```
100 REM CHAP 7, PROB 3
110 INPUT N
120 FOR X=2 TO N STEP 2
130 PRINT X
140 NEXT X
150 END
```

5. ABCDABCDABCDABCDABCD

7. Since the Z and V loops are crossed, an error message will appear.

9. It reads and prints out five numbers rounded off to two places past the decimal point.

11.
```
100 REM CHAP 7, PROB 11
110 GRAPHICS 8
120 COLOR 1
130 FOR K=1 TO 150 STEP 10
140 PLOT 0,K:DRAWTO 310,K
150 NEXT K
160 FOR J=1 TO 310 STEP 10
170 PLOT J,0:DRAWTO J,150
180 NEXT J
190 END
```

13.
```
100 REM CHAP 7, PROB 13
110 INPUT N
120 INPUT X
130 LET L=X
140 LET H=X
150 LET S=X
160 FOR I=1 TO N-1
170 INPUT X
180 IF X>L THEN 200
190 LET L=X
200 IF X<H THEN 220
210 LET H=X
220 LET S=S+X
230 NEXT I
240 PRINT "HIGHEST GRADE IS ";H
250 PRINT "LOWEST GRADE IS ";L
260 PRINT "AVERAGE IS ";S/N
270 END
```

15.
```
1 2 3
2 4 6
3 6 9
4 8 12
```

17.
```
100 REM CHAP 7, PROB 17
110 READ N
120 FOR I=1 TO N
130 READ M,R,D1,D2,D3,D4,D5
140 PRINT "EMPLOYEE NUMBER ";M
150 LET H=D1+D2+D3+D4+D5
160 IF H<=40 THEN 180
170 LET P=R*40+1.5*R*(H-40)
175 GOTO 185
180 LET P=R*H
185 PRINT "PAY IS ";P
188 NEXT I
190 DATA 5
200 DATA 2,4.8,8,10,8,7,10
201 DATA 5,3.75,7,8,8,6,10
202 DATA 1,3.25,8,10,6,8,8
203 DATA 4,5,8,10,6,10,6
204 DATA 3,4.25,6,6,8,10,7
210 END
```

Chapter 8

1.
```
100 REM CHAP 8, PROB 1
110 DIM X(20)
120 READ N
130 FOR I=1 TO N
140 READ A
150 LET X(I)=A
160 NEXT I
170 FOR I=1 TO N
180 PRINT X(I)
190 NEXT I
200 DATA 12
210 DATA 2,1,4,3,2,4,5,6,3,5,4,1
220 END
```

3.
```
100 REM CHAP 8, PROB 3
110 DIM A(10,10)
120 INPUT N
130 FOR R=1 TO N
140 FOR C=1 TO N
150 INPUT X
160 LET A(R,C)=X
170 NEXT C
180 NEXT R
190 LET S=0
200 FOR I=1 TO N
210 LET S=S+A(I,I)
220 NEXT I
230 PRINT "SUM OF MAIN DIAGONAL IS ";S
240 END
```

5.
```
100 REM CHAP 8, PROB 5
110 DIM A(15,15)
120 INPUT M,N
130 FOR R=1 TO M
140 FOR C=1 TO N
150 INPUT X
160 LET A(R,C)=X
170 NEXT C
180 NEXT R
190 LET S=0
```

```
200 FOR R=1 TO M
210 FOR C=1 TO N
220 LET S=S+A(R,C)
230 NEXT C
240 NEXT R
250 PRINT "SUM OF ENTRIES IS ";S
260 END
```

7. 9.9999998

9. 16

11.
```
100 REM CHAP 8, PROB 11
110 DIM X(100)
120 INPUT N
130 FOR I=1 TO N
140 INPUT A
150 LET X(I)=A
160 NEXT I
170 FOR I=1 TO N-1
180 IF X(I+1) <= X(I) THEN 230
190 LET T=X(I+1)
200 LET X(I+1)=X(I)
210 LET X(I)=T
220 GOTO 170
230 NEXT I
240 FOR I=1 TO N
250 PRINT X(I);" ";
260 NEXT I
270 END
```

13.
```
1 1 1 1 1 1
0 0 0 0 0 0
0 0 1 1 1 1
0 0 0 0 0 0
0 0 0 0 1 1
0 0 0 0 0 0
```

15.
```
100 REM CHAP 8, PROB 15
110 DIM X(2,5)
120 FOR R=1 TO 2
```

```
130 FOR C=1 TO 5
140 READ A
150 LET X(R,C)=A
160 NEXT C
170 NEXT R
180 DATA 2,1,0,5,1
190 DATA 3,2,1,3,1
200 FOR R=1 TO 2
210 FOR C=1 TO 5
220 PRINT X(R,C),
230 NEXT C
240 PRINT
250 NEXT R
260 END
```

17.
```
100 REM CHAP 8, PROB 17
110 DIM X(20,20)
120 INPUT M,N
130 FOR R=1 TO M
140 FOR C=1 TO N
150 INPUT A
160 LET X(R,C)=A
170 NEXT C
180 NEXT R
190 FOR R=1 TO M
200 LET S=0
210 FOR C=1 TO N
220 LET S=S+X(R,C)
230 NEXT C
240 PRINT "SUM OF ROW ";R;" IS ";S
250 NEXT R
260 FOR C=1 TO N
270 LET P=1
280 FOR R=1 TO M
290 LET P=P*X(R,C)
300 NEXT R
310 PRINT "PRODUCT OF COLUMN ";C;" IS ";P
320 NEXT C
330 END
```

19.
```
100 REM CHAP 8, PROB 19
110 DIM X(4,6)
120 FOR R=1 TO 4
130 FOR C=1 TO 6
140 READ A
150 LET X(R,C)=A
160 NEXT C
170 NEXT R
180 DATA 48,40,73,120,100,90
190 DATA 75,130,90,140,110,85
200 DATA 50,72,140,125,106,92
210 DATA 108,75,92,152,91,87
220 FOR C=1 TO 6
230 LET S=0
240 FOR R=1 TO 4
250 LET S=S+X(R,C)
260 NEXT R
270 PRINT "TOTAL-DAY ";C;" IS ";S
280 NEXT C
285 PRINT
290 FOR R=1 TO 4
300 LET S=0
310 FOR C=1 TO 6
320 LET S=S+X(R,C)
330 NEXT C
340 PRINT "TOTAL-SALESPERSON ";R;" IS ";S
350 NEXT R
360 LET S=0
370 FOR R=1 TO 4
380 FOR C=1 TO 6
390 LET S=S+X(R,C)
400 NEXT C
410 NEXT R
420 PRINT
430 PRINT "TOTAL SALES FOR THE WEEK IS ";S
440 END
```

21.
```
100 REM CHAP 8, PROB 21
110 DIM P(20),X(20)
120 PRINT "HOW LONG ARE THE ARRAYS ";
130 INPUT N
```

```
140 FOR I=1 TO N
150 LET P(I)=I
160 NEXT I
170 FOR R=1 TO N
180 INPUT A
190 LET X(R)=A
200 NEXT R
210 FOR I=1 TO N-1
220 IF X(P(I))>X(P(I+1)) THEN 270
230 LET T=P(I)
240 LET P(I)=P(I+1)
250 LET P(I+1)=T
260 GOTO 210
270 NEXT I
280 PRINT "P","X"
290 PRINT
300 FOR I=1 TO N
310 PRINT P(I),X(I)
320 NEXT I
320 END
```

Chapter 9

1.
```
100 REM CHAP 9, PROB 1
110 DIM A$(50)
120 INPUT A$
130 FOR I=1 TO LEN(A$)
140 PRINT A$(I,I)
150 NEXT I
160 END
```

3.
```
100 REM CHAP 9, PROB 3
110 DIM A$(50)
120 INPUT A$
130 LET A=0
140 LET E=0
150 LET I=0
160 LET O=0
170 LET U=0
180 FOR K=I TO LEN(A$)
190 IF A$(K,K)="A" THEN A=A+1
```

```
200 IF A$(K,K)="E" THEN E=E+1
210 IF A$(K,K)="I" THEN I=I+1
220 IF A$(K,K)="O" THEN O=O+1
230 IF A$(K,K)="U" THEN U=U+1
240 NEXT K
250 PRINT "A = ";A
260 PRINT "E = ";E
270 PRINT "I = ";I
280 PRINT "O = ";O
290 PRINT "U = ";U
300 END
```

5.
```
100 REM CHAP 9, PROB 5
110 DIM A$(50),B$(50)
120 INPUT A$
130 FOR I=1 TO LEN(A$)
140 IF A$(I,I)=CHR$(32) THEN 160
150 LET B$(LEN(B$)+1)=A$(I,I)
160 NEXT I
170 PRINT B$
180 END
```

7.
```
100 REM CHAP 9, PROB 7
110 DIM A$(50)
120 LET C=0
130 FOR K=1 TO 5
140 INPUT A$
150 IF A$(1,4)<>"THE " THEN 170
160 LET C=C+1
170 FOR I=2 TO LEN(A$)-3
180 IF A$(I,I+3)<>"THE " THEN 200
190 LET C=C+1
200 NEXT I
210 NEXT K
220 PRINT C
230 END
```

9.
```
100 REM CHAP 9, PROB 9
110 DIM A$(50)
120 INPUT A$
130 LET C=0
```

```
140 FOR K=1 TO LEN(A$)-1
150 IF A$(K,K+1)<>"IN" THEN 170
160 LET C=C+1
170 NEXT K
180 PRINT C
190 END
```

Chapter 10

1.
```
100 FOR K=1 TO
110 READ PITCH,LENGTH
120 FOR L=1 TO LENGTH
130 SOUND 0,PITCH,10,8
140 NEXT L
150 NEXT K
160 DATA 121,40,0,10,108,40,0,10
170 DATA 96,40,0,10,121,40,0,10
180 DATA 121,40,0,10,108,40,0,10
190 DATA 96,40,0,10,121,40,0,10
200 DATA 96,40,0,10,91,40,0,10
210 DATA 81,40,0,50
220 DATA 96,40,0,10,91,40,0,10
230 DATA 81,40,0,50
240 DATA 81,20,0,5,72,20,0,5
250 DATA 81,20,0,5,91,20,0,5
260 DATA 96,40,0,5,121,20,0,20
270 DATA 81,20,0,5,72,20,0,5
280 DATA 81,20,0,5,91,20,0,5
290 DATA 96,40,0,5,121,20,0,20
300 DATA 121,40,0,10,162,40,0,10
310 DATA 121,40,0,50
320 DATA 121,40,0,10,162,40,0,10
330 DATA 121,40,0,50
340 END
```

3.
```
55
15
36
```

5.
```
500 REM SUBROUTINE
510 LET L=X(2)
```

```
520 FOR I=3 TO X(1)+1
530 IF L>=X(I) THEN 550
540 LET L=X(I)
550 NEXT I
560 RETURN
```

7.
```
900 REM SUBROUTINE
910 LET S1=0
920 LET S2=0
930 FOR I=2 TO Y(1)+1
940 LET S1=S1+Y(I)
950 LET S2=S2+Y(I)^2
960 NEXT I
970 LET M=S1/Y(1)
980 LET S=SQR((Y(1)*S2-S1^2)/
    (Y(1)*(Y(1)-1)))
990 RETURN
```

Chapter 11

1.
```
100 REM CHAP 11, PROB 1
110 FOR I=1 TO 25
120 LET N=INT(100*RND(1))/10
130 PRINT N,
140 NEXT I
150 END
```

3. A typical output is:

```
0.15          0.14          0.12          0.09
  0.18          0.2           0.06          0.13
    0.14          0.02          0.13          0.05
      0.19          0.05          0.01
```

5.
```
90 REM CHAP 11, PROB 5
100 FOR I=1 TO 5
110 READ N
120 LET H=0
130 LET T=0
140 FOR J=1 TO N
150 LET X=INT(2*RND(0)+1)
160 IF X=1 THEN 190
```

```
170 LET T=T+1
180 GOTO 200
190 LET H=H+1
200 NEXT J
210 PRINT
220 PRINT "FOR ";N;" TOSSES THERE WERE"
230 PRINT H;" HEADS AND ";T;" TAILS"
240 NEXT I
250 DATA 10,50,100,500,1000
260 END
```

7.
```
90 REM CHAP 11, PROB 7
100 LET S=0
110 FOR I=1 TO 100
120 LET S=S+RND(0)
130 NEXT I
140 PRINT S/100
150 END
```

9.
```
90 REM CHAP 11, PROB 9
100 LET M=0
110 FOR I=1 TO 1000
120 LET A=60*RND(0)
130 LET B=60*RND(0)
140 IF ABS(A-B)>10 THEN 160
150 LET M=M+1
160 NEXT I
170 PRINT "PROBABILITY OF A MEET IS ";M/1000
180 END
```

11.
```
90 REM CHAP 11, PROB 11
100 FOR I=1 TO 25
110 LET S=0
120 FOR J=1 TO 12
130 LET S=S+RND(0)
140 NEXT J
150 LET R=10+2*(S-6)
160 PRINT INT(100*R+.5)/100,
170 NEXT I
180 END
```

Chapter 12

1. A possible record structure would be as follows.

Variable	Description	Approximate Length
TITLE$	Title of cassette	30
LABEL$	Record company	20
NAME$	Name of musicians	50
KIND$	Category of music	15
PRICE	Price of cassette	5

3. A possible record structure would be as follows.

Variable	Description	Approximate Length
NAME$	Name of item	50
ROOM$	Location of item	15
AMT	Value of item	10

5. A program for entering a month's charges and totalling them for a particular credit card follows.

```
100 REM CHAP 12, PROB 5
110 DIM CARD$(20),NAME$(30),TEMP$(30)
120 DIM DATE$(10),DES$(50),ANS$(3)
130 PRINT "ARE YOU CREATING THE FILE"
140 PRINT "FOR THE FIRST TIME? (Y/N) ";
150 INPUT ANS$
160 IF ANS$<>"Y" THEN 190
170 OPEN #1,8,0,"D:CARD.DAT"
180 GOTO 240
190 PRINT "DO YOU WISH TO OBTAIN CREDIT"
200 PRINT "CARD TOTALS ONLY? ";
210 INPUT ANS$
220 IF ANS$="Y" THEN 420
230 OPEN #1,9,0,"D:CARD.DAT"
240 PRINT "NAME OF CARD: ";
250 INPUT CARD$
```

```
260 IF CARD$="QUIT" THEN 410
270 PRINT #1;CARD$
280 PRINT "NAME OF STORE: ";
290 INPUT NAME$
300 PRINT #1;NAME$
310 PRINT "DATE OF PURCHASE: ";
320 INPUT DATE$
330 PRINT #1;DATE$
340 PRINT "DESCRIPTION OF PURCHASE: ";
350 INPUT DES$
360 PRINT #1;DES$
370 PRINT "COST OF PURCHASE: ";
380 INPUT AMT
390 PRINT #1;AMT
400 GOTO 240
410 CLOSE #1
420 OPEN #1,4,0,"D:CARD.DAT"
430 REM SUM THE CHARGES
440 LET SUM=0
450 PRINT "NAME OF CARD TO TOTAL: ";
460 INPUT TEMP$
470 IF TEMP$="QUIT" THEN END
480 TRAP 540
490 INPUT #1;CARD$,NAME$,DATE$
500 INPUT #1;DES$,AMT
510 IF CARD$<>TEMP$THEN 490
520 LET SUM=SUM+AMT
530 GOTO 490
540 PRINT "TOTAL FOR ";TEMP$;" IS ";SUM
550 CLOSE #1
560 GOTO 420
```

Note: This program does not accumulate the information on the file. You might want to provide for adding information to the file as was done in Program Example 2.

INDEX

COMMANDS

CONT Restarts execution after use of the BREAK key

DOS Places the computer in a mode for certain disk operations

LIST Displays the program in memory

LIST lists entire program

LIST 40,310 lists lines 40 through 310

LOAD filename Brings the specified program from diskette into memory

LOAD "D:MENU"

NEW Erases the contents in memory

RUN Executes the program

SAVE filename Puts the specified program in memory out to diskette

SAVE: "D:LABELS"

STRING FUNCTIONS

ASC(string) Returns ASCII code of first character of specified string

LET A = ASC(B$)

CHR$(number) Takes an ASCII code number and returns the corresponding character

LET P$ = CHR$(65)

LEN(string) Returns the length of a string

FOR X = 1 TO LEN(A$)

STR$(number) Returns the string representation of a number

LET A$ = STR$(27)

VAL(string) Converts a string of digits into a number

LET P = VAL(A$)

v$(n1,n2) Returns a substring of v$ starting with character n1 and running through character n2: (not a proper function but included here because it is an important implied function)

IF N$(1,4) = "NAME" THEN 300

NUMERIC FUNCTIONS

ABS(number) Computes absolute value

LET X = ABS(– 7)

INT(number) Returns the greatest integer less than or equal to the specified number

LET Y = INT(2.7) LET D = INT(RND(0)*100)

RND(0) Returns a random number between (and including) 0 up to (but not including) 1

LET D = 1 + RND(0)*9

SGN(number) Returns the sign of the specified numeric expression

LET P = SGN(R – E)

SQR(number) Returns the square root of a number

LET C = SQR(A*A + B*B)

GRAPHICS AND SOUND STATEMENTS

COLOR n Chooses the color register to be used in color graphics

COLOR 1

DRAWTO x,y Draws a line from the current position of the graphics pen to the point specified

DRAWTO 10,15

GRAPHICS n Specifies which of the 8 graphics modes are to be used

GRAPHICS 8

PLOT x,y Causes one point to be plotted at the point specified

PLOT X,Y PLOT 5,17

SETCOLOR register#,hue,brightness Stores hue and brightness data in specified color register

SETCOLOR 2,6,4

SOUND n1,n2,n3,n4 Controls which of 4 voices, sound pitch, distortion, and volume of a note

SOUND 2,121,10,10.8

QUICK REFERENCE GUIDE

STATEMENTS

CLOSE #n Closes the specified open file
```
CLOSE #1
```

DATA datalist Contains values to be assigned to variables in a READ statement
```
DATA 7.2,BOY,3,4.5
```

DIM v(n) Reserves space for arrays and strings
```
DIM A(10,20),B$(50)
```

END Stops program execution; closes files; turns off sound

FOR..TO/NEXT Creates a loop
```
FOR J=1 TO 10          FOR A=2*X TO Y STEP 2
NEXT J                 NEXT A
```

GOSUB line# Calls a subroutine beginning at specified line number
```
GOSUB 500
```

GOTO line# Jumps to specified line number (unconditional jump)
```
GOTO 412
```

IF condition **THEN** line# Jumps to the line number specified after THEN when the condition is true (conditional jump)
```
IF X > 10 THEN 320
```

IF condition **THEN** statement Performs the statement after THEN when the condition is true
```
IF Y-2=7 THEN END
```

INPUT v Causes program execution to pause for input from keyboard
```
INPUT A,B    INPUT D$
```

INPUT#n;v Causes information to be read from a file on diskette
```
INPUT #1;A,G$
```

LET v=expression Assigns a value to a variable
```
LET X=7    LET C=C+1
```

ON v **GOSUB** n1,n2,.. Causes a jump to one of several subroutines depending on the value of the variable after the ON
```
ON N GOSUB 300,350,400
```

ON v **GOTO** n1,n2,.. Causes a jump to one of several line numbers depending on the value of the variable after ON
```
ON N GOTO 100,200,252
```

OPEN #n1,n2,n3,filename Opens a file for input or output operations; n1, n2, and n3 stand respectively for the IOCB number, I/O mode, and rotated printing control (nearly always set to 0)
```
OPEN# 1,4,0,"D:DATA"
```

PRINT list Causes output to be placed on the specified output device; default is the screen
```
PRINT A,B    PRINT C$,X(J)
PRINT #1;A$
```

READ variable list Reads the next item in the DATA list and assigns it to a specified variable
```
READ B$,NUMBER,M
```

REM Allows insertion of a comment in a program line
```
REM SUBROUTINE SCALE
```

RESTORE Resets data in DATA statements so it can be read again
```
RESTORE     restores all DATA statements
RESTORE 90  restores data in statement 90
```

RETURN Returns program control from a subroutine to the line following the GOSUB that called it

TRAP line# Jumps to specified line number if an INPUT error occurs
```
TRAP 790
```

SPECIAL KEYS

[→] [←] [↑] [↓] Move cursor

[BACKSPACE] Moves cursor one space to left, deleting whatever was there and leaving a space

[BREAK] Interrupts anything in progress and returns to command level

[CLEAR] Clears the screen

[CTRL] [DELETE] Deletes the character under the cursor and contracts the line

[CTRL] [INSERT] Inserts a space

[RETURN] Signifies end of current line; when editing, accepts edit changes that were made in the line.

IMPORTANT ERROR MESSAGES

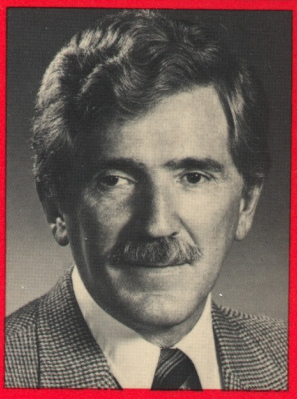
ERROR CODE	PROBLEM
2	**Out of Memory:** Insufficient room for statement or variable
3	**Wrong Value:** A negative value used where a positive one is required or value is not within allowed range
5	**String Too Long:** Attempt to store beyond the DIMensioned string length
6	**Out of Data:** READ statement requires more data than supplied by the DATA statement(s)
7	**Integer out of range:** Value is not the required positive integer or is >32767
8	**INPUT Type Conflict:** Attempt to INPUT a nonnumeric value into a numeric variable
9	**DIM Error:** Subscript out or range; reference to an unDIMensioned variable; attempt to reDIMension a variable; or DIM value is >32767
11	**Overflow/Underflow:** Attempt to divide by zero or reference to a number >1E98 or <1E−99
12	**Line Not Found:** Number after GOTO, GOSUB, or THEN does not match any line number
13	**NEXT Without FOR** or improperly nested FOR/NEXT statements
14	**Line Too Long:** Statement is too long or too complex
16	**RETURN Without GOSUB**
18	**Invalid String Characters** or string in VAL function is not a numeric string
19	**Program Too Long for LOAD**
20	**Bad Device Number:** Device number not in the required range of 1 through 7
128	**BREAK Abort:** User pressed the BREAK key during an I/O operation
129	**IOCB Already Open:** IOCB already in use for another file or device
130	**Nonexistent Device:** Specified a filename without a device (i.e. "MYFILE" instead of "D:MYFILE")
131	**IOCB Write Only:** READ command to a write-only device
133	**Device or File Not Open:** No OPEN specified for the device
134	**Bad IOCB Number:** IOCB number is not in the required range of 1 through 7
135	**IOCB Read Only:** Attempt to write to a read-only device or file
136	**EOF:** End of file has been reached
138	**Device Timeout:** Device does not respond
141	**Cursor Out of Range:** Cursor out of the allowed range for the graphics mode
144	**Protected Diskette:** Attempt to write on a write-protected diskette
147	**RAM Insufficient:** Insufficient RAM for operating selected graphics mode
160	**Drive Number Error:** Drive number is out of the allowed 1 through 8 range or computer was switched on before the drive was switched on
162	**Disk Full**
163	**Unrecoverable System Data I/O Error:** Diskette or DOS may be bad
164	**File Number Mismatch:** Problems with links on the disk (try turning the system off and on again)
165	**Filename Error:** Filename includes illegal characters
169	**Directory Full:** Not enough room on diskette
170	**File Not Found:** File does not exist on diskette

*IOCB = Input/Output Control Block (IOCB number corresponds to the device number)

BYTE BOOKS

HANDS-ON BASIC FOR THE ATARI® 400/800/1200XL

PECKHAM



About the Author

HERBERT PECKHAM is a principal in Computer Literacy, a firm specializing in materials for computer education. He formerly was professor of mathematics and physics at Gavilan College where he also taught courses in computing languages.

The author has served on the Executive Board of the American association of Physics Teachers and was a member of that organization's committee on computers in physics teaching. Professor Peckham is author and co-author of numerous books and monographs on BASIC, Pascal, and computer applications.

When using, fold flap inside back cover.

Hands-On BASIC

FOR THE ATARI® 400/800/1200XL

About the Book

"Hands-On BASIC for the Atari 400/800/1200XL" is one of a series of adaptations of an earlier work by the author. This version includes sections on graphics, sound and files. The book makes use of the hands-on method, providing computer experience through a series of guided activities. Each of these activities is followed by a discussion of the BASIC topic just explored. The hands-on method is a proven, efficient way to learn BASIC programming with a minimum of supervision.

ISBN 0-07-049194-1