TEXAS INSTRUMENTS
INCORPORATED

Programming BASIC
with the TI Home Computer

"Programming BASIC" is a tutorial guide that helps you learn TI BASIC in a friendly, relaxed manner. It goes beyond the "Beginner's BASIC" furnished with the Texas Instruments Home Computer—so you can explore the full range and power of TI BASIC, including color graphics and sound.

Herbert D. Peckham

TEXAS INSTRUMENTS SOFTWARE SERIES
McGRAW-HILL BOOK COMPANY

PROGRAMMING BASIC WITH THE TI HOME COMPUTER

PROGRAMMING BASIC WITH THE TI HOME COMPUTER

Herbert D. Peckham
Professor of Natural Science
Gavilan College

Texas Instruments, Inc.

McGraw-Hill Book Company
New York St. Louis San Francisco Auckland Bogotá Düsseldorf
Johannesburg London Madrid Mexico Montreal New Delhi
Panama Paris São Paulo Singapore Sydney Tokyo Toronto

PROGRAMMING BASIC WITH THE TI HOME COMPUTER

5 6 7 8 9 0 DODO 8 3 2

This book was set in Megaron by Instant Type. The editor was Charles E. Stewart.
Production supervisor was Richard A. Ausburn.
R. R. Donnelley & Sons Company was printer and binder.

Library of Congress Cataloging in Publication Data

Peckham, Herbert D.
Programming BASIC with the TI home computer.

Includes index.
1. Basic (Computer program language)
2. Minicomputers—Programming. I. Title.
II. Title: Texas Instruments personal computers.
QA76.73.B3P43 001.6'424 79-12417
ISBN 0-07-049156-9

TABLE OF CONTENTS

PREFACE

This book is a modification of an earlier work by the author that was also published by McGraw-Hill Book Company. That book, titled "BASIC: A Hands-On Method," introduces students to BASIC on a number of different timesharing computers. This earlier material has been revised and modified to be used specifically on the home computer manufactured by Texas Instruments Incorporated. Since the motivation and ideas that lead to the original work are equally valid with respect to the TI Home Computer, they bear repeating in this book.

Two characteristics of most BASIC programming texts on the market are very objectionable. First, almost all quickly begin to use mathematics at a level that excludes the vast majority of the people we are most interested in, many of whom can rely on introductory algebra (very dimly remembered) but who, for a variety of reasons, want to learn how to program in BASIC. The second objection is that generally nothing in the structure of most BASIC texts requires the beginner to spend much (if any) time on the computer. Beginners typically try to study programming like any other subject and do not feel the need to experiment with and execute programs on the computer. It seems axiomatic that much more effective learning will take place if most of the study of BASIC utilizes the computer. This text's main thesis is that more traditional text material should be preceded by a good deal of time experimenting with the language on the computer. The experience to date validates the idea that students work though the material more rapidly and effectively with this initial exposure to BASIC on the computer.

Most textbooks are used in a classroom environment as part of the formal educational system. Certainly, many students will learn how to program in this traditional setting. However, the sales of the home computers will touch all parts of our society. This means that the usual concept of a "student" must be changed dramatically. This text has been designed to be useful to anyone (whether part of the educational system or not) who wants to learn how to program the TI Home Computer.

The reader will immediately note that the book is structured quite differently compared to most programming texts. Each chapter begins with a statement of the objectives for that chapter. Then the student is guided through a set of exercises that demonstrates BASIC in action and permits experimentation with its characteristics. Once a "feel" for BASIC has been acquired, one can more profitably proceed to the usual text treatment. The mathematics level has intentionally been kept very low. The student with more advanced mathematical skills will have little difficulty learning how to employ these skills on the computer. However, if the mathematics level in the text were set too high, the majority of beginners would become discouraged in the first few chapters. At the level presented, nearly anyone should be able to work through the material without getting "hung up" by the mathematics. The student must have access to a Texas Instruments Home Computer to use this book.

The book is organized into eleven chapters. If used in a classroom setting, each chapter forms a block of instruction that should require about two hours of classroom time and possibly three or four hours of time outside class. Review tests are provided at the end of each chapter, enabling the student to see if the objectives have been mastered. Problem sets have been included to provide practice in programming. Solutions to the odd-numbered problems are at the end of the book.

The book can be used in several different ways. First, and probably most important, it can be used with no supervision as a self-study text. It has also been used in an open-entry, open-exit, self-paced course. If desired, the material can be presented in a traditional lecture format.

Students at any level, from junior high through graduate school, from housewife to senior citizen, from factory worker to professional, should be able to master the material without difficulty. The goal is to provide programming skills in BASIC as rapidly and effectively as possible. Some capabilities of the TI Home Computer are not covered in this book. By and large the topics not covered involve mathematics past the level assumed in the presentation. As already indicated, no mathematics past introductory algebra is required, and the algebra used is mainly formula evaluation. More mathematical ability is nice but unnecessary.

Two documents furnished with the TI Home Computer have a bearing on the content and style of this book. First, there is a reference manual that has all the specifications and capabilities of BASIC as implemented on the TI Home Computer. Very few will be able to make use of this information initially.

On the other end of the spectrum is a primer ("Beginners BASIC") designed to quickly acquaint you with the operation of the computer and the elements of BASIC programming. After becoming familiar with the material in the primer, many will feel comfortable writing programs and will use the reference manual to answer questions as they arise. However, it is felt that most beginners will feel there is a "gap" between the primer and the reference manual. The purpose of this book is to bridge this gap. Consequently, topics will be developed very leisurely. If you are a person who is "computer smart," you will find the presentation slow. If, on the other hand, you are a bit nervous about the whole idea of learning to program a computer, hopefully you will appreciate the easy pace, and will be able to master the material without difficulty.

Acknowledgments

Texas Instruments Incorporated provided generous assistance in the production of this book. Several employees of Texas Instruments deserve special mention. Mr. Alfred Riccomi, Mr. Charles Watkins, and Ms. Susan Naff gave valuable encouragement and advice. Particular thanks are due Mr. Robert O'Dell who read the entire manuscript and provided a most useful critical review.

The errors that remain are, of course, due to me. Comments or suggestions for improvement of this book will be appreciated.

Herbert D. Peckham

CHAPTER
ONE

THE TEXAS INSTRUMENTS HOME COMPUTER AND BASIC

Computers are now a common part of our lives. We may not see them, but they are there, involved in some way in most of our daily activities. Business of all sizes, educational institutions, various branches of government—none would be able to handle the bewildering quantity of information that seems to characterize our society without using computers. Only recently, however, has it been possible to bring small, inexpensive computers into the home or classroom. For the first time, people in all walks of life, from students to senior citizens, are becoming involved with computers. As the price of computers continues to drop, this trend will surely go on. More and more people will want to know how to use computers to enable them to participate fully in our society.

1-1 WHAT IS BASIC?

You are about to embark upon the study of a computer language called BASIC using a very powerful home computer manufactured by Texas Instruments. BASIC is a very specialized language designed to permit you and the computer to understand and communicate with one another. This language is certainly much easier to use than a spoken language such as Spanish or French. Even so, BASIC does have a simple vocabulary consisting of a few words, a grammatical structure, and rules of usage just like any other language. The first task will be to learn the vocabulary of BASIC and become used to its rules of grammar. Next, we will see how the language permits you to use the computer in a wide range of activities. The level of mathematics involved has intentionally been kept very low. Therefore, if you feel a bit rusty in your mathematical skills, don't be too concerned. As we proceed through BASIC, you will have an opportunity to brush up on some elementary mathematics.

A very effective way to learn is to observe details and characteristics while actually performing a task: the "discovery" method. This is the strategy that will be used in this book. You will be asked to begin each chapter with a discovery session on the computer. After following the directions and watching closely what the computer does in response to your instructions, you will begin to acquire a "feel" for BASIC and how the computer operates.

Once you have this type of understanding, you can proceed more profitably to study the written material that summarizes what you have learned. Thus, the directed exercise on the computer is a key part of learning about BASIC as presented in this book.

1-2 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, work began on the concept of time sharing on a computer and the creation of a programming language written from a user's point of view. A very interesting sidelight is that much of the actual programming on the project was done by undergraduate students at Dartmouth. The birthday of BASIC is May 1, 1964, so the language is still a teen-ager.

The success of this pioneering effort at Dartmouth soon attracted national attention, and very quickly other institutions became interested. The rest is history. Today, nearly every time-sharing computer supports the BASIC language. The most recent development is the implementation of BASIC on small home computers. Each year, the percentage of total computer activities done in BASIC increases compared to other languages. What started as a project at a single college is now an established part of the computer industry throughout the world.

1-3 WHAT IS THE TEXAS INSTRUMENTS HOME COMPUTER?

The concept of a powerful computer, priced about the same as the average color television set, capable of doing most if not all the tasks that formerly required large computers in air-conditioned isolation, is a new and somewhat unsettling idea. However unsettling the concept may be, this is precisely what has happened. The home computer produced by Texas Instruments Incorporated promises to be a major force in changing traditional attitudes about computers and how they are used.

Before starting to learn how to program in BASIC on your TI Home Computer, we should pause to examine its origins, and point out some of its remarkable characteristics. Above all else, two things are important about the home computer. First, the price is such that large numbers of people will either own or have access to one. This raises the second point that needs to be emphasized. The question of accessibility to computer facilities has always been difficult to deal with. Often, it seemed that barriers, some real and some imaginary, were placed in the paths of those who desired to use computers. With the TI Home Computer, all such barriers are gone. Thus, this new personal computer will be found in homes, offices, and classrooms across the country. By definition, "personal computing" has to be "accessible computing." The whole point of the TI Home Computer is to bring powerful computing facilities within the reach of all!

The heart of the home computer is a microcomputer on a chip.The first such microcomputers were manufactured in 1973, so a very new technology is involved. Several extremely important characteristics are embodied in the TI Home Computer. The computer output is through a color TV display which means that color and sound can be utilized.Under control of a BASIC program, you can generate graphic designs (in any of sixteen desired colors) on the TV display accompanied by music or sound. The old Chinese adage that "one picture is worth a thousand words" is certainly applicable here and points up one of the powerful characteristics of the TI

Home Computer. One could add that "one picture accompanied by music is worth a million words.!"

Second, it was clear that BASIC would be the language of the Texas Instruments Home Computer, and that it would have to be powerful enough to permit a wide range of uses from the novice playing games to the professional programmer. BASIC was selected because of good earlier experience with the language on time-sharingcomputers. BASIC is a "friendly" and nonthreatening language that is easy to learn but has powerful capabilities. In short, it is ideal for personal computing and has been almost universally adopted for home computers.

For several decades, Texas Instruments has been one of the pioneers in electronic technology. The appearance of the TI Home Computer is the logical extension of this leadership position. Backed by the years of experience and record of corporate responsibility, you can be assured of the continued and effective support of your home computer.

1-4 HOW TO BEGIN

You should approach each chapter in the book in the same way. The material has been organized with special learning patterns in mind, and any change will be less effective and require more of your time.

Each chapter begins with a brief statement of the objectives. These should be studied carefully in order for you to get a clear picture of precisely what is to be done. (It's nice to know where you're going!) When asked, you should record the computer output in the space provided. Occasionally you will be asked to answer questions. The purpose of this activity is to lead you through the ideas involved and let you see BASIC working. It is important that you try to think about what will happen in situations that will be set up. Quite often you will be deliberately led into error situations. The purpose, of course, is to draw you into the meat of programming! This is an active relationship between you and the computer that should not be slighted. Time spent in this activity will save you much more time later on.

Following the discovery exercises in each chapter, a complete discussion is given to cover all the objectives a second time. 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 great detail to point out how the parts are pulled together to produce a complete BASIC program. Of course, the ultimate goal in all the chapters is for you to learn how to write and execute BASIC programs on the Texas Instruments Home Computer. Be sure to allow sufficient time to study and understand all the examples.

Each chapter after Chapter 4 has a collection of problems. You should plan to work enough problems to satisfy yourself that you can write programs at the level appropriate to that chapter. Solutions to the odd-numbered problems are given at the end of the book.

Finally, each chapter (except the first) has a practice test. The purpose of this test is to review your understanding of the material and point out any areas that need further study. The answers to the practice tests are in a section at the end of the book.

CHAPTER

TWO

GETTING ACQUAINTED WITH YOUR HOME COMPUTER

Since your first contact with the computer may seem a bit strange and complicated, we will proceed very slowly. Rest assured that 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 needed.

2-1 OBJECTIVES

In this chapter we want to get familiar with the computer and start learning how it operates. No BASIC programming will be done until the next chapter. However, learning how the keyboard operates, and how information is entered and modified, is fundamental to all that will follow. This material is very easy to master, but do make sure that you understand all the objectives thoroughly.

Connecting The Computer to Your TV Display.

The TI Home Computer uses a color TV display as the primary output device. See your owner's reference manual for details about how to connect the computer to the TV display.

Immediate Mode

One of the easiest ways to use the computer is in the immediate mode. No programming is involved; rather the computer carries out instructions as they are entered. In due time we will learn how to do much more indeed, but for the present, simple operations in the immediate mode are a nice introduction to operation of the computer.

Screen Editing

Rarely can information be entered into a computer without making mistakes. We need to be able to easily change or correct material that has been entered. A thorough knowledge of this capability will save you a great deal of time later on.

2-2 DISCOVERY ACTIVITIES

Before beginning work on the computer, we must establish several important points. On a typewriter, the L is often used for the numeral 1. A different key is used, however, on the computer. The numeral 1 is found with the other numeral keys at the top of the keyboard. One of the most frequent mistakes made by the beginner is to type L when the numeral 1 is desired. Next, don't use the upper case letter O for the numeral 0. Like the numeral 1, the 0 on the computer keyboard is found with the numeral keys.

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

Now we are ready to begin work. Sit down in front of the computer, get comfortable, and let's go!

1. First, turn on the TV. Then turn on the computer with the switch located at the lower right front of the cabinet. After a few moments you will see a message ending with PRESS ANY KEY TO BEGIN. Follow the instructions and press any key on the keyboard. This causes a selection list of capabilities to be displayed. Since the first selection is "TI BASIC," and since this book is solely about BASIC you should always type the number 1 at this point. Now type

```
PRINT 1+4
```

and stop. Has anything happened?

____

Now press the ENTER key and record below what happened.

____

2. Now you know how to make the computer do addition. Let's explore this some more. Type

```
PRINT 20.1+54
```

and press ENTER. What happened?

____

3. Type

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

and press ENTER. Record the output below.

____

4. All right, the + and - are simple enough. Type the following expression

```
PRINT 12/2
```

and press ENTER. What happened?

____

What arithmetic operation does the / call for?

____

5. If, when typing in material, you make an error, you can move the cursor back to the error by pressing the shift-S key. Each time the shift-S key is pressed, the cursor will move one place to the left. When you reach the error, retype the line

correctly. When you press the ENTER key, computer may come back with * INCORRECT STATEMENT. If this happens, try to see what the problem is and retype the line.

6. Your TV screen should be fairly full now. Type CALL CLEAR and press the ENTER key. What happened?

   ____

7. Now that you know how, you can clear the screen any time you desire. If the screen is full and new lines are entered, old lines will scroll off the top. Let's go on exploring the immediate mode. Type

   ```
   PRINT 2*50
   ```

   and press ENTER. What happened?

   ____

   What arithmetic operation is called for by the *?

   ____

8. Type in the following expression but don't press ENTER when finished.

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

   What do you think will happen when you press ENTER?

   ____

   Press ENTER and record below what did happen.

   ____

9. Now on to a new wrinkle. Type

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

and press ENTER. What did the computer do?

10. What will happen if you type

```
PRINT "BAD DOG"
```

and press ENTER?

Try it and see if you were correct.

11. Now let's move on to a different topic. First, clear the screen. If you have forgotten how, look back at step 6. Type the following line. Press the ENTER key when through.

```
GRADE = 95
```

Now type

```
PRINT GRADE
```

and press ENTER. What happened?

12. Let's go on with this new idea. Take a few moments to examine the lines below.

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

What do you think the computer will do if you type in these lines?

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

13. Study the lines below briefly.

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

What will the computer do with these instructions?

Clear the screen and type in the lines. Remember to press ENTER after each line. What did the computer do?

14. We have seen one example of the CALL statements in CALL CLEAR which clears the screen on the TV display. Let's look at some of the other CALL statements that are available. First, clear the screen. Now type

```
CALL HCHAR(12,1,88,32)
```

and press ENTER. What happened?

15. All right, now clear the screen and type the following:

```
CALL HCHAR(12,16,65,32)
```

Press the ENTER key and record what happened on the screen.

16. Clear the screen and try the following:

```
CALL VCHAR(1,1,90,768)
```

This time watch closely what happens when you press the ENTER key. What happened?

17. OK, let's go on to a different topic. Clear the screen, and type the following lines. Remember to put in spaces where indicated. At the end of each line press the ENTER key. Make sure the volume control on the TV display is up.

```
TIME = 1000
NOTE = 440
CALL SOUND(TIME,NOTE,0)
```

You should have heard a pure tone on the TV. Did you?

Experiment with this a bit more. In particular, try setting TIME to 100 and 3000. Try other values for NOTE (stay in the range 440 to 880). After a few trials you should be able to figure out how the CALL SOUND statement works.

18. This concludes the discovery material for now. Type BYE and press the ENTER key. Then turn the computer off and go on to the discussion material.

2-3 DISCUSSION

Now we will go back over the topics that you have just worked with on the computer. With this experience you will be in a far better position to understand the discussion.

Turning The Computer On and Off

The computer is simplicity itself to turn ON and OFF! As you have already seen, this is done with the switch at the lower right front of the computer cabinet. After first turning on your TV display, when the computer is turned on, you are greeted with the message:

```
      TEXAS INSTRUMENTS
       HOME COMPUTER

  READY-PRESS ANY KEY TO BEGIN
```

If you press any key, the computer obliges with the following list of options.

```
1 FOR TI BASIC
2 FOR EQUATION CALCULATOR
3 (optional)
```

If one of the command modules that are available from Texas Instruments is in the computer, item number 3 will tell you what it is. If no package is inserted, the third item in the list is blank. We will always be concerned with the first option—TI BASIC.

One important point; if at any time things get away from you, if you have lost touch, or if the computer seems out of control, you have a foolproof escape mechanism. Simply press the shift key and type Q. This puts you back at the initial level encountered when the computer is turned on. As an aside, pressing the shift-Q key is equivalent to typing BYE. At any rate, once you either type BYE or press the shift-Q key, all the former ills will be forgotten and the computer will once again be ready for business. This remedy is not without disadvantage, however, since you will lose any programs or information in memory at the time you typed shift-Q. However, it is an absolute way for you to regain control. Of course, if you should inadvertently type shift-Q while entering material into the computer, you will suddenly find yourself out of BASIC and back at the initial part of the turn-on sequence. This is something to be careful about.

Immediate 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 "immediate" mode. As we shall see in the next chapter, BASIC stores instructions and commands in a series of numbered lines, and then is directed by you to perform all the instructions at the same time. If, however, the instructions are typed in without a line number, the computer assumes you want an immediate answer and does what you asked it to do, if possible.

When material is typed in, nothing happens until you press ENTER. The ENTER key tells the computer you are through typing and to begin processing the information. Remember, when you are through typing anything at all and want to let the computer know, press the ENTER key.

There are a few cases where the computer responds to a single keystroke and does not require that the ENTER key be pressed. An example of this is the instruction PRESS ANY KEY TO BEGIN that is part of the turn-on sequence. However, such cases are the exception rather than the rule.

We 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 desired. There are a number of other clever operations that can be done, but we will postpone discussion of these to later chapters.

If you type

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

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

If you type

```
PRINT "ABCDEFG"
```

and press ENTER, 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," and is an important concept which we will return to throughout the balance of the book.

The computer can keep track of a number of pieces of information in the immediate mode. Thus

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

will cause 5 to be printed on the screen. There is a very important point in connection with this concept. If we type

```
PRINT TAX
```

and press ENTER, the numeral zero will be displayed. Since we gave no value to TAX, the computer assigned the value 0 and then printed it out.

The computer is very relaxed about names for quantities used either in the immediate mode or in BASIC programs. You can use "long" names like WIDTH or RATE as well as "short" names like W or R. However, this ability to use long names does create something to be careful about. The names are set off by spaces. Thus, the spaces are significant both in the immediate mode and in BASIC programs. Certain words cannot be used for variable names since they are reserved for use by the computer. See the reference manual for a list of reserved words.

This very brief introduction to the notion of variable names suffices for our discussion of the immediate mode. We will return for a more complete discussion of the concept later in the book.

In the discovery work you saw several examples of CALL statements. You also encountered these statements in the primer supplied with the computer. CALL statements should be used in BASIC programs to be most effective. Since we are just beginning the study of BASIC, we will delay a full discussion of CALL statements until Chapter 11. The only reason for bringing the subject up here is that the CALL statements can be used in the immediate mode.

However, we do need to discuss an important point with regard to the CALL CLEAR command. As you will see in Chapter 7, the characters that appear on the screen come from a numbered set. In particular, character number 32 is a space. CALL CLEAR fills the TV screen with character number 32. Of course, this simply clears all the material from the screen which is what we want to happen. It is possible to redefine character number 32 to some different character. If this is done (probably without notice), CALL CLEAR will fill the screen with this new character. To say the least, you would be surprised to see the screen filled with a strange character upon a CALL CLEAR rather than the expected clear screen, and might mistakenly assume that something was wrong with your computer. If you should experience this, simply be aware of what is taking place.

Screen Editing

The TI Home Computer has line editing commands that can be used to make changes. These are most effective when used to modify BASIC programs. However, since some of the commands can be used in the immediate mode, we should look at the process in detail.

We will limit our discussion to changes in a line before the ENTER key has been pressed. First, the cursor can be moved back and forth with the shift-S and shift-D keys. The arrows on these keys help you remember what their function is. The cursor can be moved over characters in the line without changing them. If a character is typed, that character replaces the character under the cursor. We can also insert or delete characters in a line. If you press the shift-G key and then type characters, the new material is inserted in the line beginning at the position of the cursor. The old material in the line is shifted to the right as the new characters are inserted. If the shift-F key is pressed, the character under the cursor is deleted and all material to the right is shifted left one place. By pressing the shift-F key several times, as many characters as desired can be deleted from a line.

These simple editing commands can be used in the immediate mode to make changes or corrections. Remember, though, that they work only if you haven't yet pressed the ENTER key. In the next chapter we will see much more capability when the editing commands are used on BASIC programs.

2-4 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 at the end of the book.

1. When you are through typing a line, how do you let the computer know?

____

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

____

3. What symbol is used to indicate 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 ENTER?

7. What will happen if you type

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

and then press ENTER?

8. Suppose you type PRING 2+3*4 and before you press ENTER note a G where a T should be in the word PRINT. Describe exactly how to correct this.

CHAPTER

THREE

INTRODUCTION TO BASIC

Now we are ready to begin learning about programming in BASIC. In this chapter we will see how to write and execute some very simple programs.

3-1 OBJECTIVES

The objectives are simple but important as they are your first introduction to BASIC. The objectives are listed below.

Requirements for BASIC Programs

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

Telling The Computer What to Do

System commands tell the computer to do something to or with a BASIC program. These action words are used to control a program. We will look at the following system commands: LIST, RUN, NEW, RES, and NUM.

Entering and Controlling Programs

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

Variable Names in BASIC

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

3-2 DISCOVERY ACTIVITIES

In the discovery activities that follow you will be directed to enter various programs. If you see an <ENTER> in the instructions, press the ENTER key. Remember from your experiences in Chapter 2 that pressing the ENTER key tells the computer you are through typing. Now go on to the activities below.

1. Turn on your computer and go to BASIC. Type in

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

   This is the first line of a BASIC program. Note the ">" prompt at the left of the screen where the next line will go.

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

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

   If you make mistakes while typing in the program, either retype the line or correct it using the method learned in Chapter 2.

3. Clear the screen using the CALL CLEAR command. What happened to the program you just typed in?

   ____

4. Fortunately, all is not lost. The computer has remembered what you typed in even though the screen is blank. Type LIST and press the ENTER key. What happened?

   ____

5. On the TV display you should see the program 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 the computer is told to carry out the instructions, what do you think will happen?

____

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

____

6. All right, now type

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

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

____

7. If you tell the computer to execute this program what do you think will happen?

____

This time watch the change in the screen color when the program is executed. Type RUN, press the ENTER key, and record what happened. Were you right?

____

8. Now type

```
140       <ENTER>
```

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

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

9. Now RUN the program. What happened?

Does the END statement that formerly was in line 140 appear to be required by the computer?

10. Let's experiment a bit more. Often we want to clear out the program in the computer's memory. This is done with the NEW command. Type NEW and press the ENTER key. What happened?

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

11. We have learned how to clear out a program in memory, but now have no program left! To get our program back we must enter it again. Type in the program below.

```
100 LET A=1       <ENTER>
110 LET B=8       <ENTER>
```

```
120 LET C=A+B     <ENTER>
130 PRINT C     <ENTER>
140 END     <ENTER>
```

Check all the lines to make sure they were entered correctly. If a line needs to be changed retype it. If you had to retype lines, clear the screen with CALL CLEAR and redisplay the program by typing LIST.

12. Now type

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

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

____

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

____

Type RUN, press the ENTER key, and record below what the computer did.

____

14. In the original program the line numbers were not consecutive (like 100, 101, 102, 103, etc.) but had gaps (e.g., 100, 110, 120, 130, and 140). Can you think of a reason for doing this now? (Hint: See step 12.)

____

15. How do you insert lines in a BASIC program? (Hint: See steps 12 and 14.)

____

16. Clear out the program in memory by typing NEW and pressing the ENTER key. Enter the program below.

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

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

____

18. Now RUN the program and record what the computer did.

____

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

____

19. Type the numeral 10 and press the ENTER key. What took place?

____

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 we want to get out of the program. Press the shift key and the C key at the same time. From now on we will refer to this as "shift-C." What happened?

____

22. Clear out the program in memory. Type in the following program.

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

23. RUN the program and record below what happened.

____

When you get tired watching the display, press the shift-C key. What happened?

____

24. Try it once more. RUN the program and after a few numbers are typed out, interrupt the program. How do you stop a BASIC program running on the computer?

____

25. Clear the screen and display the program in memory. Type the lines below. Note the absence of spaces in the first line and the extra spaces in the second.

```
100LETA=1        <ENTER>
1 2 0 L E T A = A + 1       <ENTER>
```

What happened?

____

Now clear the screen and LIST the program. Clearly 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 ENTER key. Type in the program below.

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

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

____

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

____

28. All right, now type the numeral 13, press the ENTER key, and record below what happened.

____

29. This simple program illustrates that we can arrange for BASIC programs to print out messages as well as numbers.

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

```
100 LET A=1        <ENTER>
110 LET A$="HOUSE"      <ENTER>
120 PRINT A        <ENTER>
130 PRINT "A"         <ENTER>
140 PRINT A$        <ENTER>
150 PRINT "A$"        <ENTER>
160 END       <ENTER>
```

31. This program contains something new. Look at the A$ in line 110. 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$. RUN the program and record the output.

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

```
155 PRINT B       <ENTER>
```

33. Clear the screen and display the program with the LIST command. Note that the only place B is mentioned is in line 155 in the PRINT statement. What do you think will happen if we RUN the program?

    OK, now RUN the program and record what happened.

34. As you saw, 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 be considered while writing programs. We will return to this issue later.

35. Now we want to look at something that can help you a great deal while writing programs. Clear the screen and display the program. Focus your attention on the line numbers. Now type RES 1000,10 and press the ENTER key. Display the program. What has happened?

____

36. Let's try this once more. Type RES 200,5 and press the ENTER key. Display the program. Now what happened?

____

Do you see the purpose of the RES command?

____

37. This concludes the discovery activities for this chapter. Type BYE and press the ENTER key. Now turn off the computer and go on to the next section.

3-3 DISCUSSION

Now that you have been through the discovery activities at your computer and have seen some of the features of BASIC in action, we can summarize what has taken place.

Correcting Mistakes

Since most of us make mistakes while typing, we need to be able to correct errors sent to the computer. Suppose a mistake is made while you are typing a line. How it is corrected depends upon whether you have pressed the ENTER key yet, and where the error is. Before you press ENTER, the cursor can be moved back and forth in the line to make corrections. The shift-S key moves the cursor left and the shift-D key moves the cursor right. You saw in Chapter 2 that characters can be inserted after pressing the shift-G key. Or, characters can be deleted with the shift-F key. A keyboard overlay is available which identifies the function of the keys used in line editing. When all the corrections are made, press the ENTER key. Note that the cursor does not have to be at the right end of the line when ENTER is pressed. When you press the ENTER key, the line you have been typing (mistakes included if they haven't been corrected) is analyzed by the computer. Some errors will be picked up at this point in which case the computer will type out * INCORRECT STATEMENT.

Some errors may not show up until you RUN the program. If the computer detects an error at this point, it will type out an error message and an associated line number. Suppose the computer found an error in line 350. If you type EDIT 350, line 350 will be displayed on the screen. Now the line can be corrected using the editing keys.

We have another option open to us after making changes in a line. If we press the ENTER key, the changes are posted but the computer then leaves the edit mode. However, if we don't press the ENTER key, the computer stays in the edit mode. If there are other changes to be made in nearby lines, we press the up- or down-arrow keys as needed to bring the desired line on the screen. When new lines are brought on the screen using the up- or down-arrow keys, the computer posts the changes made in a line when a new line is brought on the screen. However, in this case, the computer stays in the edit mode. When all the changes have been made, press the ENTER key to post the final modifications and leave the edit mode.

Requirements for BASIC Programs

Several important facts about BASIC programs have been demonstrated. To have a program to use for discussion purposes, we will return to the original program used in the discovery activities:

```
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. In the program above, there are three types of BASIC statements: assignment (identified by the = sign), PRINT, and END. The first two will be treated fully in the next chapter. For the time being, the use of each of these statements in the program is clear. The END statement, however, has particular significance. As you saw in the discovery material, the END statement is optional. However, in this book we will always use the END statement as it is a clear indication that the program is finished.

The highest numbered statement in the program must be the END statement.

Generally the line numbers in a BASIC program are not numbered consecutively (such as 100, 101, 102, etc.). The reason is that we may want to insert additional statements later if we discover errors or want to modify the program. If the lines were numbered consecutively, changes might involve retyping segments of the program. With gaps in the line numbers, statements can be inserted by simply typing in the new statements using line numbers not already in the program.

Quite often we want to sort out the line numbers in a program after changes have been made. This has nothing to do with the execution of the program, but merely makes the program look nicer. The resequence command is used to renumber the lines in a program. By typing RES M,N where M and N are numbers, the program lines are numbered beginning with M and are spaced N apart. Thus RES 1000,100 would number the first line in the program 1000. The second would be 1100, the third 1200, and so on. Later on we will discover that BASIC programs can branch to any line numbers in the program. The resequence command takes care of these branch line numbers as well as the lines themselves.

The computer doesn't care what order the lines in a BASIC program are entered. If, for example, we type

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

and this new program is displayed, the computer will sort out the statements and display them in numerical order. In the same way, if we told the computer to RUN the program, the statements would be sorted into numerical order before starting execution.

You can remove a BASIC statement from a program by typing the line number and pressing the ENTER key. Statements can be modified by retyping the lines involved, pressing the ENTER key after each line is typed, or by using the editor. As indicated above, statements can be added by using line numbers not already in the program. Thus, BASIC statements can be added, removed, or changed as desired. The ability to change programs easily is one of the powerful characteristics of BASIC.

If desired, you can direct the computer to provide the line numbers automatically when typing in programs. If you type NUM 1000,10 the computer provides the line number 1000 and waits for you to type in the line. When the ENTER key is pressed, the line number 1010 is displayed for the next line at which point the computer waits for the next statement to be typed in. In general, NUM M,N causes line numbers to be provided automatically beginning with M, and spaced N apart. When you are finished typing in a program and the line number for the next line is displayed, press the ENTER key to get out of the automatic line numbering mode.

One last point about BASIC involves spaces in the statements. The computer permits spaces in BASIC statements only at certain locations. Common sense is the best guide here. Don't put spaces in the line numbers, within variable names, in key words (like LET, PRINT, etc.), or within numbers. For example, the following BASIC statement is incorrect.

```
1 06L ETX=1.03 58
```

There are spaces in the line number, the key word LET, and in the number assigned to X. With spaces dribbled through the statement, it's hard to read! Generally, put spaces where they make the statement the easiest to read and you will have no difficulty. The following statement illustrates how this should be done.

```
106 LET X=1.0358
```

This not a high-anxiety item. If you make a mistake, the computer will let you know about it. After a few hours of programming, correct location of spaces in BASIC statements will become second nature to you.

Telling The Computer What to Do

We must make a sharp distinction between the statements in a BASIC program and system commands. System commands tell the computer to do something with a program. We have seen several of these in the discovery material and will briefly review the use of each.

Quite often we want the computer to type out the program it has in memory. This could be because of changes in the program that produce a cluttered screen. Or, you and the computer may be in a state of mutual confusion about the program. The way to resolve the issue is to instruct the computer to display the program presently in its memory. This is done with the LIST command. If you type LIST and then press the ENTER key, the computer will display the program on the screen. Usually, you would clear the screen first so a clean copy of the program would be displayed. Since only twenty-four lines can be displayed at a time, lines may scroll off the screen when listing long programs By modifying the LIST command though, we can look at any part of a program we desire no matter how big the program. If, for example, we type LIST 300-400, the computer will display those BASIC statements in the program from 300 to 400 inclusive. Or, LIST -200 will cause the computer to print out all the program statements from the beginning of the program up to line 200. List 300 will display only line 300. Finally, LIST 400- will instruct the computer to display the program lines from 400 through the end of the program. Clearly, you can change the numbers involved in the LIST command to look at any part of the program you desire.

A BASIC program is simply a set of instructions to be acted upon by the computer. However, the computer must to be told to start this process. This is done with the RUN command. When the RUN command is received, the computer goes to the lowest numbered statement in the program, carries out the instructions, goes to the next higher numbered statement, and keeps on carrying out instructions in numerical order, unless the program directs a statement to be done out of order. Remember then, when you want the computer to start acting on the instructions contained in a BASIC program, type RUN and press the ENTER key.

One of the very nice features of the TI Home Computer is that the color of the TV display changes during program execution. When you are typing in material the letters are black on a light blue screen. However, if the program is being executed, the screen color changes to a light green. The screen color therefore gives you an easy way to determine if the computer is in program execution or not.

Suppose you are finished working with a program and decide to go on to another. You can clear the screen, but this does not clear the current program out of memory. The computer has one portion of memory that keeps track of the screen display. A separate part of memory holds the current program. Thus, clearing the screen clears out everything in the screen portion of memory. The NEW command is used to erase the current program in memory. As you saw in the discovery activities the NEW command has an implied CALL CLEAR in it. Thus, any time you use the NEW command, the screen will be cleared. You should be careful to use the NEW command when you are finished with a program. If the old program is not erased, a new program goes into the same space with the very confusing result that the computer may have parts of two different programs in memory.

Entering and Controlling Programs

So far, when you have been instructed to type in commands or program statements, the <ENTER> prompt was given to remind you to press the ENTER key. This habit should be well doveloped by now, so we will not use the <ENTER> prompt in further work.

Situations come up where we need to be able to control a program that is running. Certainly one of the most dramatic cases is when a program is in a closed loop and will keep on running forever if we don't interrupt it. We can break into such a program by pressing the shift-C key. When this is done, the computer breaks the program execution, tells us * BREAKPOINT AT (whatever line was being processed when the interruption took place). A different situation is when the computer is in an input loop waiting for a number to be typed in. If we want to get out of such a situation, again press the shift-C key. The computer then jumps out of the program execution back to the READY mode.

Variable Names in BASIC

Now we come to one of the ideas in BASIC that most often causes problems for the beginner. It concerns variable names and the distinction between the name and the quantity stored in memory under that name. In the BASIC statement

```
100 LET A=2
```

the letter A names a variable. By "variable" we mean that different values can be assigned to A. Statements that have an = sign in them are called "assignment" statements. In the case above, the variable A is assigned the value 2. Actually, what is taking place is that the computer has named a memory location A, and has stored a 2 in that location. You must be careful to separate the name of a location in memory from the contents of that location. It's the same notion as the difference 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 be changed at any time.

The use of the LET in assignment statements is optional. As far as the computer is concerned you can use LET or not. In this book we will always use LET in assignment statements for a reason to be explained below.

Consider the following statement.

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

This 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 means to evaluate what is on the right and assign it to the variable named on the left.

To pursue this issue further, suppose we have a BASIC statement such as

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

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

$$B = B + 1$$

By subtracting B from both sides of this equation we have

$$0 = 1$$

which is very strange indeed! It is certainly clear that the = sign in a BASIC statement does not mean the same as 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 the number, and put the result back into the storage location named B. The use of LET in assignment statements helps us remember that the equal sign implies assignment, not equality.

If we 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 storage location called A and put the number 1 in that location. Line 120 tells the computer to multiply 2 by 3 and store the product in memory location A. Note that the 1 stored previously in memory location A has been lost.

This brings us to the heart of the issue. The letter A, which identifies a storage location, is called a variable because the contents of A can be changed. The name of the location does not change, but the number stored there can be changed as desired.

To be precise, the variable A referred to above is called 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, and now we must tie up some loose ends.

As far as names are concerned, it is easy to distinguish between numeric and character-string variables. A, B, M, and P would all identify numeric variables and name numeric quantities. A$, B$, M$, and P$ all name strings of characters. The $ symbol that is appended identifies the name as a character-string variable. In the BASIC statement

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

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

The TI Home Computer has very relaxed rules for variable names. Usually, BASIC permits only a letter or a letter followed by a single digit to name numeric variables, and the same combination with a $ appended to name character strings. The TI Home Computer permits you to use "long" names for either numeric variables or character strings. You can use up to 15 characters (including the $ character in the case of character strings) in long names. The computer has a set of "reserved" words that are used in BASIC and for system commands. These words cannot be used to name variables. See the reference manual for the list of reserved words. If you make a mistake and use one, however, the computer will let you know!

The use of long names is very nice since the name of the variable defines what it means. For example, LENGTH, TIME, NAME$, and MILEAGE need no further definition. whereas L, T, N$, and M would have to be explained. However, if you do use long names in programs there is a disadvantage that you must be aware of. You must spell the names correctly each time they are used. The computer will treat MILEAGE and MILAGE as two different names.

Let's go over the important points once more. A variable name in BASIC identifies a storage location in memory. If the variable is numeric, a number is stored in the memory location. If the variable is a character string, a collection of characters is stored in the memory location. The contents of the storage location can be modified, but the names of the storage locations remain the same.

The assignment statement evaluates what is on the right side of the equal sign and assigns the result to the storage location named on the left side. Thus,

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

instructs the computer to evaluate the expression using the numbers stored in memory locations named A, B, and C. The results are then stored in the memory location named D.

We have just scratched the surface with regard to character-string variables. We will return to this topic several times during the balance of the book.

3-4 PRACTICE TEST

Take the test below to discover how well you have learned the objectives of Chapter 3. The answers to the practice test are given at the end of the book.

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

   ____

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

   ____

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

   ____

4. What will happen if the following program is RUN?

```
100 LET A=1
110 LET B=2
120 LET C=B-A
130 PRINT C
140 END
```

5. How long can "long" variable names be?

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

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 display the program in memory?

10. How do you erase the screen?

____

11. How do you erase a program from memory?

____

12. How do you command the computer to start executing a program in memory?

____

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

____

CHAPTER

FOUR

COMPUTER ARITHMETIC AND PROGRAM MANAGEMENT

4-1 OBJECTIVES

Now that you have been introduced to BASIC, we are ready to go on to more interesting tasks.

Arithmetic on the Computer

Ultimately, all mathematics on a computer is done using the simplest arithmetic operations. It is essential to have a clear understanding of how these arithmetic operations are done.

Parentheses ( ) in Computations

As we shall see, all mathematical expressions must be typed a line at a time to enter them into the computer. Some expressions can be handled this way only by organizing parts of the expression in parentheses. Thus, the effective use of parentheses is a necessary skill.

E Notation for Numbers

Both very large and very small numbers arise in computer work. "E notation" is used by the computer to describe such numbers. We need to be able to recognize and interpret E notation since the computer may type out numbers in this form.

Storing and Retrieving Programs

We have already seen some system commands. Additional system commands will be introduced in this chapter which will permit us to store and retrieve programs from the cassette unit that can be attached to the computer.

4-2 DISCOVERY ACTIVITIES

The discovery activities in this chapter introduce the characteristics of computer arithmetic on the computer. Additional system commands for program management will be explored.

Now let's go on to the discovery material for this chapter.

1. Turn your computer on, go to BASIC, and type in the following program:

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

What arithmetic operation is called for by the + in line 120?

____

2. Let's see if you are right. RUN the program. When the computer goes to line 100, it will type out a question mark, halt, and wait for you to type in a value for A. In this case, type in 10. The computer will then go to line 110, type out a question mark, halt, and wait for you to type in a value for B. Type in 20. What did the computer print out?

____

3. Change the + in line 120 to - by editing the line. Clear the screen and LIST the program. RUN the program and at the first question mark (INPUT prompt) type in 30 for A and at the second prompt, type in 12 for B. What happened?

____

What arithmetic operation is done with the - in line 120?

____

4. Change the - in line 120 to *. Clear the screen and display the program. RUN the program and type in 5 for A, and 6 for B when the INPUT prompts (question marks) come up. What did the computer print out?

   What arithmetic operation does the * call for?

5. Now change the * in line 120 to /. RUN the program and when the INPUT prompts come up, enter 45 for A and 15 for B. What was printed out?

   What arithmetic operation does the / call for?

6. Thus far we have seen only a single arithmetic operation on a line. Let's look at an example in which there is more than one operation. Type

```
120 LET C=A+B-B/3
```

   Clear the screen, 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 happen?

   RUN the program, enter the values above, and write down what happened.

7. Clear out the program in memory by typing NEW and pressing the ENTER key. Then type

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

The upward pointing carat mark in line 110 is the shift-J on the key board. Make sure you have entered the program correctly. Then RUN the program and record the results below.

____

Compare the numbers printed out with the expressions in the lines where they were computed. See if you can figure out what is taking place.

8. Change lines 100 and 110 to read as follows:

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

RUN the program and write down the computer did.

____

9. Change lines 100 and 110 to read as follows:

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

RUN the program. What happened?

____

What is the ^ symbol used for in BASIC?

10. Clear the screen and the program in memory. Enter the following program:

```
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, it is clear that the same numbers are involved in each of the calculations in lines 100, 110, 120, and 130. The only difference is the grouping in the lines. RUN the program and record what the computer did.

Study the program and the numbers typed out until you see what is taking place in the program. There are very specific rules that the computer uses in such situations. If you aren't able to see clearly what these rules are, don't worry; we will go over the topic completely later in the chapter.

11. Clear the screen and then clear the program in memory with the NEW command. Now enter the following program:

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

Line 120 will be folded on two lines on the screen when you type it in. When folding takes place, the computer does not provide the ">" prompt at the left side of the screen. This is to indicate that the line is a continuation of the one above. RUN the program and record the output.

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

12. Change the first three lines in the program to read as follows:

```
100 LET A=3/100
110 LET B=3/(100*100*100)
120 LET C=3/(100*100*100*100
*100)
```

Again, line 120 will be folded when it is typed in. RUN the program and record the output.

Again, can you see what is taking place in the output? Count the zeros in the denominators in lines 100, 110, and 120.

13. If an E shows up in a number printed out by the computer, what does it mean? Explain in your own words.

If you still do not fully understand the purpose of the E notation, relax! We will return to it later.

14. Obtain a clear tape and place it in the cassette unit connected to the computer. (If you don't have a cassette unit, go on to the discussion material.) If there are any questions about connecting the cassette unit to the computer, see the reference manual. Remember that we have a program in memory. LIST the program to make sure it is there. Now type

```
SAVE CS1
```

The CS1 refers to cassette number one. What happened?

All right, follow the instructions as they are displayed on the screen.

15. If you followed all the instructions properly, the program in memory will be recorded on the tape. After the recording process is finished, the computer displays the message

```
* CHECK TAPE (Y OR N)?
```

Suppose we do want to check the tape. Press the Y key for YES and follow the instructions.

---

16. Now you have recorded a program on the tape cassette. Let's see how to load the program back into the computer. First clear the program from memory with the NEW command. Then type

```
OLD CS1
```

What happened?

Follow the instructions displayed on the screen until the program is loaded. To make sure that everything has worked properly, LIST the program after loading. Once the program has been loaded from tape, you can work with it as if it had been typed in at the keyboard.

17. This completes the discovery activities for now. Remove your tape cassette, type BYE, turn the computer off, and go on to the next section.

4-3 DISCUSSION

A number of very important points have been introduced in the computer work. Probably you didn't meet with too much difficulty going through the discovery material, but this shouldn't make you ignore the fundamental ideas involved. Lack of understanding at this point will return to haunt you later on in the book. Consequently we will go over each of the objectives of the chapter in great detail to ensure that they are mastered.

Arithmetic on the Computer

We are concerned with five arithmetic operations. These are addition, subtraction, multiplication, division, and exponentiation. The first four are certainly familiar to you, and the last (exponentiation) might be frightening mainly because of the fierce-looking word used to define the process. Let's go over each of these operations and see how the computer handles them.

Addition and subtraction are done precisely as you would expect. The symbols used to define the operations (+ and -) mean the same thing to the computer that they mean in mathematics classes.

Multiplication is handled the same way on the computer as in arithmetic but has a different symbol to define the process, the * character. Thus 2*3 is 6. A*B signals the computer to look up the numbers stored in A and B, then multiply them together. Usually, X is used to indicate multiplication. However since X can be a variable name in BASIC, we can't use this symbol to call for multiplication. This is the reason the symbol * is used.

Division is indicated with the / symbol. A/B means to divide the number stored in location A by the one stored in B. Likewise, 8/2 means to divide 8 by 2.

Finally, the exponentiation operation is defined by the ^ symbol. Exponentiation means "raised to the power." Therefore, 34 means "3 raised to the fourth power," which in turn means 3 multiplied by itself four times, giving 81 as the result.

We must be very careful to understand the order in which arithmetic operations are done by the computer. Consider the following expression:

```
2+3^2/5-1
```

If the computer simply goes through the expression from the left, performing operations as they are met, the result would be 2 plus 3 (giving 5), raised to the second power (giving 25), divided by 5 (giving 5), minus 1, producing an answer of 4. However, suppose addition and subtraction are done first, then exponentiation, then multiplication and division. This would give 5 raised to the second power (giving 25), divided by 4, for an answer of 6.25.

BASIC expressions are scanned from left to right.

Clearly, we could go on with different rules for the order of arithmetic operations and might get different answers each time. The point is that there are well-defined rules in BASIC for the order and priority of arithmetic operations, and we must understand them. Here they are:

The order of operations is from left to right using the priority rules given below.

The priority for arithmetic operations is (1) exponentiation, (2) multiplication and division, and (3) addition and subtraction.

The priority rules are

1st	^
2nd	* and /
3rd	+ and -

Now, if we go back to our example of

```
2+3^2/5-1
```

we scan left to right for any exponentiation. Since there is an exponentiation indicated (32), it is done first. Now the expression is

```
2+9/5-1
```

Scanning from left to right, we again look for exponentiation, and finding none, look for 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 left in the expression, we 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 until they become second nature to you. We will look at the rules again when the use of parentheses is discussed in the next section.

One very important point must be made about arithmetic operations on any computer. Mathematical theory assumes that an infinite number of significant digits will be handled in all numbers in all operations. Thus, 1/3 is really .333333... with the pattern going on for ever. But the computer can handle only a fixed number of significant digits in any number. The TI Home Computer, for example, would print 1/3 as .3333333333. There are a lot of threes in this expression but not an infinite number of them! Thus the computer treatment of arithmetic is an approximation of the true situation. This is further compounded by the fact that calculations are carried out in base 2 arithmetic rather than base 10. This leads to conversion errors.

The reason from bringing this whole issue up is that sometimes computer results are very close, but not exactly equal to expected results. If the square root of 4 turns out to be 1.999999999, don't be too upset! It is simply a consequence of the inherent errors in any computing machine.

Parentheses in Computations

The rules for order and priority of arithmetic are not the whole issue, however. There is often a bit more involved. To see this, consider the following more complicated example:

$$\overbrace{(\overbrace{(2{*}3+4\wedge 2)}^{A}{*}2+5)}^{B}{*}\overbrace{(3\wedge 2-4)}^{C}$$

Obviously, the difference between this expression and the ones we have been studying is the use of parentheses to group parts of the expression. We will go through this example in great detail to show you how the computer attacks the arithmetic involved.

The computer starts by scanning from left to right and meets the open parenthesis of B. It then looks inside to see if there are any open parentheses and finds one for A. The next parenthesis met is a close 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 our problem has become

$$\overbrace{(22*2+5)}^{B}*\overbrace{(3\wedge 2-4)}^{C}$$

On the next scan, the computer isolates parentheses B, does the arithmetic inside, and the problem is now

$$49*\overbrace{(3\wedge 2-4)}^{C}$$

Since only the C parentheses are left, the arithmetic inside is done, giving

49*5

which after the final multiplication yields the final answer 245.

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

One final point about parentheses is that they must be balanced. That is, there must be as many open parentheses "(" as close parentheses ")". In complicated expressions you should always count the number of open and close parentheses to make sure they are equal. This doesn't guarantee that the parentheses are grouped correctly, but it will catch obvious errors involving missing parentheses.

E Notation for Numbers

Numbers are printed out by BASIC in different forms. In particular, numbers are sometimes printed out in what is known as the "E notation." Examples of this notation are 2.456E+06 or 6.032E-14. Now we will go back over the ideas introduced in the computer work to clarify the idea of E notation.

It is easy to see why such a special notation is needed for either very large or very small numbers. The computer prints out ten digits in a number, like 1.853695325 even though it uses as many as fourteen digits in the calculation. A problem comes up if we want the computer to print out a number like 4681063270000000 which would require sixteen digits. The computer will print this as 4.68106E+15, which means that the decimal point belongs fifteen places to the right of its present position. Notice that the E+15 took the place of the last four digits in the normal ten character display. A number like 89560000000000 would be printed out as 8.956E+13. The E+13 means that the decimal point belongs thirteen places to the right. In no case are more than ten characters typed out for a number (including the four character "E" part). We can also express very small numbers in the same way. For example, the computer will print out the number 0.0000000006835984 as 6.835984E-10. The E-10 means that the decimal point belongs ten places to the left. The table below should help you understand how to convert from decimal to E notation or from E 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 convert from decimal to E notation, count the number of places the decimal must be moved until there is a single digit to the left of the decimal point. The number of places moved is the number that follows E in the E notation. If you had to move the decimal to the left, the sign following E is +. If you moved the decimal point to the right, the sign following E is –.

To change from E 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.

E notation is not something to get tense about since you will rarely use it when setting up programs on the computer. The main reason for bringing up the issue is that the computer may print out numbers in the E notation. Consequently, you should be able to recognize what is happening.

Storing and Retrieving Programs

If every time we turned on the computer, we had to type in the programs that we wanted to use, very little work would get done. One of the nice features of the TI Home Computer is provision for attaching a tape cassette to store programs. Once we type in a long program and troubleshoot it, we don't want to have to go through the process again every time we want to use the program. Programs can be stored on tape cassettes and subsequently loaded back into the computer any time we desire.

Before getting involved in the system commands for storing and retrieving programs on the tape unit, we should pause to consider some fairly obvious facts about tape cassettes. First, if we record a program over a previously recorded program, the original information will be lost. Therefore, if a tape has programs already recorded on it, we must be careful to position the tape so that anything new we want to save will go on unused tape. Another important point in this connection is that many short cassette tapes with one program per tape is much better than one long tape with many programs. It is difficult to position a long tape to a particular program unless you have an expensive cassette unit with digital position readout. The easy way around the problem is to use very short tapes and record only a single program per tape. A final comment is not to skimp on tape quality as low quality tape may increase the probability of recording errors.

Now let's see how to save a program on the cassette unit. Of course, the cassette unit must be properly connected to the computer. Also, there must be a program in memory that we desire to save. The process starts by typing

```
SAVE CS1
```

The CS1 refers to cassette unit number one. By identifying the output device, we have allowed for more than one output device to be connected to the computer at the same time. At any rate, after you type the command above, the computer prints back the message

```
* REWIND CASSETTE TAPE CS1
  THEN PRESS ENTER
```

This instruction is to make sure the tape is positioned properly. If there are no programs on the tape, you rewind it. If programs have been recorded, you should position the tape to the beginning of the unused portion. Either way, when the tape is positioned, press the ENTER key. At this point the computer will display the message

```
* PRESS CASSETTE RECORD CS1
  THEN PRESS ENTER
```

After the record switch on the tape cassette unit is pressed the and ENTER key is pressed, the computer starts recording the program in memory on the tape. When this starts, the computer displays the message

```
* RECORDING
```

After the program is recorded on the tape cassette you will see

```
* PRESS CASSETTE STOP CS1
  THEN PRESS ENTER
```

Follow the instructions and stop the cassette unit. In all these cassette instructions, the purpose of pressing the ENTER key is to let the computer know you have done what was requested.

After recording the tape the computer asks

```
* CHECK TAPE (Y OR N)?
```

If you press the N key (for no), you are put back into BASIC. If you press Y (for yes), the computer will give you the instructions to read the program on tape and compare it to the program in memory. It's a good practice to always check the tape. The messages involved are

```
* REWIND CASSETTE UNIT CS1
  THEN PRESS ENTER
```

and

```
* PRESS CASSETTE PLAY CS1
  THEN PRESS ENTER

* CHECKING
```

Assuming no errors are detected, the computer will display the following messages:

```
* DATA OK

* PRESS CASSETTE STOP CS1
  THEN PRESS ENTER
```

If errors are found, you will see one of the following messages:

```
* ERROR - NO DATA FOUND

* ERROR DETECTED IN DATA
```

After one of these messages comes up on the screen, you will see

```
PRESS R TO RECORD CS1
PRESS C TO CHECK
PRESS E TO EXIT
```

If you press R, the whole recording process starts over again. C causes the checking process to commence again. Finally, if you press E, you are put back into BASIC.

The procedure is reversed to load a program into the computer from the tape cassette unit. Put the tape in the cassette unit, clear out the memory in the computer, and type

```
OLD CS1
```

The computer will come back with

```
* REWIND CASSETTE TAPE CS1
  THEN PRESS ENTER
```

After you do this, the computer will display the message

```
* PRESS CASSETTE PLAY CS1
  THEN PRESS ENTER
```

When this is done you will see the message

```
* READING
```

indicating that the program is being read from the tape.

After the program is loaded, and assuming that no errors are detected, you will see

```
* NO ERROR DETECTED

* PRESS CASSETTE STOP CS1
  THEN PRESS ENTER
```

At this point, the program is loaded and ready for use.

If errors are encountered during the loading process, one of the following error messages is displayed.

```
* ERROR - NO DATA FOUND

* ERROR DETECTED IN DATA
```

Then you are given the following options:

```
PRESS R TO READ
PRESS E TO EXIT
```

If you press R, the whole process of reading the tape starts again. If you press E, you are returned to BASIC.

It may seem that there are a great many details involved in recording programs on tape and subsequently loading them back into the computer. However, once you type SAVE CS1 to start the recording process, or OLD CS1 to start the reading process, all the necessary instructions are displayed on the screen. After you go through the process several times, you should encounter no problems.

One final comment has to do with characteristics of cassette units. Generally more problems are encountered reading programs from tape than in recording programs on tape. Thus, if errors show up while loading a program, reload several times and more than likely you will get a successful load.

4-4 PRACTICE TEST

The practice test that follows is provided for you to check how well you have learned the key points and objectives of the chapter. Check your answers against the key given at the end of the book.

1. Write down the symbols that are used to carry out the following arithmetic operations in BASIC expressions: subtraction, multiplication, addition, exponentiation, and division.

   ____

2. When evaluating arithmetic expressions, there is a priority of operations. What is this priority?

   ____

3. When scanning arithmetic expressions, the computer does the search in a specific direction. What is this direction?

   ____

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

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

   ____

5. If the following program is RUN, what will be typed 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) 567300000000000 and (b) 0.000003814275168.

7. Convert the following numbers to decimal notation: (a) 7.258E+06 and (b) 1.437E-03.

8. In the expression below, give the order in which the operations will be done by the computer.

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

9. How do you save a program on the tape cassette?

10. How do you retrieve a program from the tape unit?

____

CHAPTER

FIVE

INPUT, OUTPUT, AND SIMPLE APPLICATIONS

5-1 OBJECTIVES

In this chapter we will get down to the business of writing programs to carry out tasks. We will also increase our knowledge of BASIC by looking at some details about input and output. The objectives are as follows.

Getting Numbers into a BASIC Program

There are only three ways that we can enter numbers into the computer for a BASIC program. We need to understand how this is done.

Printing Out Variables and Strings

After information is computed, it must be printed out. Different choices are available for how the output is to take place. Usually we will want to output strings of characters as well as numbers. The string output is handled essentially the same way as numbers, but needs special attention.

Spacing the Printout

The previous objective is concerned with the output of numbers and strings of characters. Here we are concerned with the spacing of that output.

The REMark Statement

The wise programmer includes comments in programs to help explain or interpret what is being done. The REMark statement in BASIC permits us to do this.

Simple Applications

Our ultimate goal is to learn how to write and troubleshoot programs. In this chapter we will begin with some modest programming assignments.

5-2 DISCOVERY ACTIVITIES

Let's go straight to the computer work.

1. Turn your computer on, select BASIC, and type in 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 we RUN this program?

____

RUN the program. When the first question mark is typed out (the input prompt for A), type in 2. Likewise, when the second question mark comes up, type in 3, and finally, at the last question mark, type in 5. Record what happened below.

____

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 and see if you are right. Then type

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

Display the program. What has happened?

3. RUN the program, and when the INPUT prompt (?) is output, 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 again, and this time when the INPUT prompt is output, type

```
2,3
```

What happened?

What is the problem?

5. The computer is still waiting for input. This time type

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

What happened?

____

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

____

What will happen if you do?

____

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

____

What will happen if you do?

____

8. Type

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

Display the program. What has happened?

____

RUN the program and record what the computer did.

____

9. Now type

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

and display the program. What has happened?

____

10. RUN the program and record what happened.

____

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

____

11. Name two different methods (other than the assignment statement) for getting numbers into a program. (Hint: See steps 2 and 8.)

____

12. Display the program in memory. Delete the DATA statement and then type

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

Since we can't edit line numbers, we must enter the line with the new number. Display the program again. What has happened?

____

13. RUN the program and record the output.

____

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

____

14. Clear out the program in memory with the NEW command. 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 DATA ERROR message associated with the READ statement or the DATA statement?

15. Delete the DATA statement in line 140 from the program. Now enter

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

Display the program. What has taken place?

16. If we RUN the program, what do you think will be typed out?

RUN the program and see if you were correct. Record the output below.

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

Does it seem to 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 what took place.

19. Now type

```
110 PRINT "A"
```

and display the program. What has happened?

What will happen if we RUN the program?

RUN the program and record what the computer printed out.

20. Type

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

and display the program. What do you think will happen if we RUN the program now?

RUN the program and record what did happen.

21. Now let's try a different wrinkle. Type

```
105 LET B=2
110 PRINT "B = ";A
```

Display the program and study it carefully. If we RUN the program, what do you think will happen?

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

22. Type

```
95 REM DEMO PROGRAM
```

Display the program. What has happened?

RUN the program. What was output?

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

23. Clear out the program in memory and enter the following program:

```
100 REM CONVERSION PROGRAM
110 REM CONVERT LBS TO GMS
120 PRINT "INPUT LBS.";
130 INPUT P
140 LET G=454*P
150 PRINT P;" POUNDS IS"
160 PRINT G;" GRAMS"
170 GOTO 120
180 END
```

Display the program and check to see that it is correct. Study the program carefully and try to guess what will happen if we RUN it. Now 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. Remember that this is done by pressing the shift-C key. What is the purpose of the REM statement?

24. We can handle the input somewhat differently. Type

```
120
130 INPUT "INPUT LBS.":P
```

Display the program and look carefully at the changes. Note the character string in the INPUT statement in line 130. What do you think will happen if we RUN the program?

RUN the program and record what happened.

This ability to have a prompt displayed in an INPUT statement is a nice feature of TI BASIC.

25. Type

```
115 INPUT P
120 PRINT "INPUT LBS.";
130
170 GOTO 115
```

and then display the program. What has 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.

26. Let's experiment with this program a bit more. Clear out the program from memory and enter it again, modified as follows:

```
100 REM CONVERSION PROGRAM
110 REM CONVERT LBS TO GMS
120 PRINT "INPUT LBS.";
130 INPUT P
```

```
140 PRINT P;" POUNDS IS"
150 PRINT G;" GRAMS"
160 LET G=454*G
170 GOTO 120
180 END
```

Can the program be RUN in this form?

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

Explain in your own words what is wrong. Remember that if a variable is not defined initially in your program, the computer will set it equal to 0.

27. Jump the computer out of the INPUT loop. Clear out the program in memory and enter:

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

RUN the program and record what happened. Pay particular attention to the spacing of the numbers.

28. Add a comma after the A in line 110. RUN the program and record what happened.

29. Now replace the comma after the A in line 110 with a semicolon. RUN the program and record what happened.

30. 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 27.)

Suppose the variable is followed by a comma?

What will happen if the variable is followed by a semicolon?

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

```
100 LET A=10
110 READ B
120 PRINT TAB(A);B;
130 LET A=A+10
140 GOTO 110
150 DATA 1,2,3
160 END
```

RUN the program and record what happened.

32. Change the A+10 in line 130 to A+5. RUN the program and record what happened. Again, pay particular attention to the spacing.

33. Now change the A+5 in line 130 to A+3. RUN the program and record what happened.

34. What does the TAB in the print statement appear to control?

35. This concludes the computer work for now. Type BYE, turn your computer off, and go on to the discussion material.

5-3 DISCUSSION

In this chapter we have begun to get away from the mere mechanics of controlling the computer. Instead, we will concentrate more on writing and troubleshooting programs. This skill doesn't come naturally to most students, 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 3 we saw one way to get numbers into a program. That was by assigning 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. We need to examine other ways in which numbers can be introduced into a BASIC program.

Let's look first at the INPUT statement and how it is used. An example might be

```
260 INPUT G
```

When the computer executes this line, it will print out a question mark as a prompt that input is expected from the keyboard; it will then halt and wait for you to type in the number. In the case above, the number typed in will be known as G.

More than one variable may be called for in a single INPUT statement, such as

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

In this case the same INPUT prompt (the question mark) is typed out, but now the computer is expecting four numbers to be typed in, separated by commas. If only three numbers are entered and the ENTER key is pressed, the computer will come back with an error message that it didn't get the input expected, and will ask you to try again. If more than four numbers are typed in initially, the computer will type out an error message as above and will wait for you to retype the input.

Usually it is wise to precede an input statement with a message explaining what is to be typed in. You can include such a message in the input statement itself. An example of this is

```
150 INPUT "ENTER WEIGHT":W
```

If this statement were executed, the message ENTER WEIGHT would be printed out. Then the computer would halt and wait for you to type in the value of W. There is no question mark typed out in this variation of the INPUT statment. Notice the colon which separates the character string from the input variables. This colon must be present or an error message will be printed out.

One final comment about input statements. You can ask for input of either numeric or character-string variables. An example might be

```
130 INPUT A,B$
```

In this case the computer is expecting a number, a comma, and a character string to be typed in. It is important that the actual input matches by type the input that is expected. If in the example above you were to type in

```
2L3,HOUSE
```

The computer would detect an error and ask you to input the data again. The problem is with the L in the number. As pointed out previously, a common mistake is to type L instead of 1. In this case, the numeric input was supposed to be 213 but the computer detected the L which can't be in a number. Just be careful to enter numbers when numbers are expected and character strings when they are expected, and you will have no problems. If you do make a mistake, the computer will let you know about it!

The last method of providing for numerical input into the computer is with the READ and DATA statements. The statement

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

is handled by the computer in the same manner as the INPUT statement, with two exceptions. First, the computer does not stop. There is no need to, as will be seen. The second exception is that the numbers called for are read from DATA statements contained within the program rather than being entered at the keyboard in response to an INPUT prompt.

To illustrate the READ and DATA statements, 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 four numbers from the DATA statements and prints out the sum of the numbers. It makes no difference where the DATA statement is in the program except that the END statement still must be the highest numbered statement. There can be more than one DATA statement, and they need not be grouped together at the same place in the program. As numbers are called for by READ statements, they are taken in order from the DATA statements, beginning with the lowest numbered statement. Should more numbers be requested after all numbers have been used from the available DATA statements, the computer will print out an DATA ERROR message and halt. On the other hand, it is possible for a program not to use all the numbers in the DATA statements in which case no error message will be generated.

To sum up, there are three methods by which numbers can be introduced into BASIC programs. They are (1) the assignment statement, (2) the INPUT statement, and (3) the READ and DATA statements. There are times when each of these methods can be used to advantage. You will become familiar with the advantages and disadvantages of each method as we spend more time writing programs.

You can put numbers in a BASIC program with: LET (assignment), READ-DATA, and INPUT statments.

Printing Out Variables and Strings

Output from the computer is quite simple. The computer can print out either the numerical value of a variable (a number) 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 since X is enclosed in quotation marks. Line 120 prints 2 since that is the number stored in location X.

The rule is clear. Any characters contained within quotation marks are called strings. Strings are printed out exactly as listed. The computer does not attempt to analyze or detect what is in the strings. If a variable in a PRINT statement is not contained within quotes, the computer prints out the numerical value of that variable.

It is possible to do computations within 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, followed by the number stored in D.

Spacing the Printout

The version of BASIC implamented on the TI Home Computer has a "built-in" standard spacing mechanism that prints two numbers spaced equally on one line. This standard spacing is used 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 second position on a line and encounters a comma in a PRINT statement, it does a return and prints the number on the first position on the next line. Thus

```
100 PRINT A,B,C
```

would cause the numerical values of A and B to be printed on a line in the two standard positions. The numerical value of C would be printed below the value of A on the next line.

Another type of spacing is produced by the semicolon between variables, such as

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

The semicolon produces closer spacing than the standard spacing obtained with the comma. However, the spacing is not always uniform, since numbers may be typed out in different formats. We will let it go with the statement that

```
100 PRINT A;B
```

produces closer spacing of output than

```
100 PRINT A,B
```

Finally, we can closely control the spacing on a line by using the TAB function in PRINT statements. The TAB function works in the same way as a tabulator setting on a typewriter. There are twenty-eight printing positions on a single line on the display screen.

The statement

```
100 PRINT TAB(5);A;TAB(20);B
```

signals the computer to space over to the fifth printing position, print the numerical value of A, space over to the twentieth printing position, and finally print the numerical value of B. It is also possible to have a variable tab setting that is controlled by the computer:

```
100 PRINT TAB(X);A
```

Here the computer must first look up the value of X, then space over to the printing position determined by the nearest integer to X (for example, if X = 23.14350826, the computer will space over to the twenty-third printing position), then print out the numerical value of A.

Since there are only 28 printing positions on a line, you might wonder what would happen if the computer tried to execute

```
100 PRINT TAB(40);B
```

What happens is that the computer will keep subtracting 28 from the number in the TAB function until it is less than or equal to 28. In this case, one subtraction yields 40–28 = 12 which is less than 28. Then, the computer will space over to the twelfth printing position and print the value of B.

Use the TAB function to produce variable spacing in a line.

We can produce vertical spacing in the output by using a PRINT statement as follows:

```
100 PRINT
```

Since the computer looks for the quantity to be printed and finds none, it then looks for punctuation and finding none orders a return and drops the cursor down one line. If we wanted two or three empty lines in the printout, we can obtain the vertical spacing by using as many empty PRINT statements as desired.

Another variation on the PRINT statement is to use the colon to separate the variables to be printed. The colon produces a return to the beginning of the line and drops the cursor down one line. Thus

```
100 PRINT A:B:C
```

and

```
100 PRINT A
110 PRINT B
120 PRINT C
```

produce exactly the same results in a program.

You can print out character strings with a PRINT statement. An example might be

```
100 PRINT A$,B$
```

If A$ and B$ are both short enough, the computer will print them on the same line. However, if B$ is too long, A$ will be printed on one line and B$ on the next. Finally, if A$ is too long for a single line, it will be split with the balance on the next line.

The REMark Statement

The REM (stands for "remark") statement is quite different from the statements we have seen previously. As soon as the computer senses the characters REM following the line number, it ignores the balance of the statement and goes on to the next line. What, then, is the purpose of the REM statement if the computer pays no attention to it? The REM statement is a way of providing information for the benefit of the programmer or someone reading the program. This information makes it much easier to follow what is taking place in the program. The wise programmer will use REM statements liberally.

Put information in a program with REM statement.

To illustrate the use of REM statements, two programs will be presented. They both will produce identical results, but the second uses REM statements to describe what is happening in the program. You can be the judge of which program is easier to follow.

No REM statements:

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

With REM statements:

```
100 REM COMPUTE THE AVERAGE
OF FOUR NUMBERS
110 REM INPUT FOUR NUMBERS
120 INPUT A,B,C,D
130 REM COMPUTE AVERAGE
140 LET X=(A+B+C+D)/4
150 REM PRINT THE AVERAGE
160 PRINT X
170 END
```

Note that in the program above, line 100 is longer than the maximum number of twenty-eight characters that can go on a line. Thus, the surplus is printed on the next line. Very long lines will be folded on more than two successive lines on the screen. As we get into more complicated programs, you will see this happening more frequently. When lines are folded, the computer does not provide the ">" prompt that is normally at the left of the screen. One thing to be careful of is that numbers in a long line may fold over into the next line where, if it happened to be at the right place, they could be mistaken for a line number of the next line. Admittedly this would be a rare occurence, but since it could happen you should be forewarned.

5-4 PROGRAM EXAMPLES

As we said earlier, we will spend progressively more time writing and debugging programs. The examples chosen for this chapter are very simple but illustrate the ideas we have been discussing. Study each example carefully until you are certain that you understand all the details. You might want to enter the programs into your computer and RUN them to verify that they work as designed.

Example 1 - Unit Prices

Our problem is to write a program to compute unit prices on supermarket items. We will let T stand for the total case price, N for the number of items in the case, and U for the unit price. We can compute the unit price with the following relationship:

$$U = T/N$$

As an example, suppose that a case of twelve large cans of fruit juice costs $6.96. The unit cost per can would then be

$$U = 6.96/12 = 0.58$$

We want the program to be designed so that when RUN it will produce the following typical output:

```
TOTAL PRICE ? 6.96
HOW MANY ITEMS ? 12
UNIT PRICE IS
.58
```

The numbers after the question marks are typed in when the program is RUN. For any total price and number of items, the program should compute and print out the correct unit price. Remember that if we desired, long variable names like TOTAL, NUMBER, and UNIT could have been used instead of T, N, and U.

Examine the first line of the desired output. There is a message printed, followed by a question mark and the input of a number from the keyboard. We can do this easily with the following statements:

```
100 PRINT "TOTAL PRICE ";
110 INPUT T
```

Remember that T stands for the total price. The semicolon at the end of line 100 prevents the return of the cursor to the left side of the screen. The next two lines in the program are written in the same style as the first two.

```
120 PRINT "HOW MANY ITEMS ";
130 INPUT N
```

N stands for the number of items. We must now compute the unit price which will be called U.

```
140 LET U=T/N
```

All that remains is to print out the final two lines of output, and add the END statement.

```
150 PRINT "UNIT PRICE IS"
160 PRINT U
170 END
```

Now we pull the whole program together.

```
100 PRINT "TOTAL PRICE ";
110 INPUT T
120 PRINT "HOW MANY ITEMS ";
130 INPUT N
140 LET U=T/N
150 PRINT "UNIT PRICE IS"
160 PRINT U
170 END
```

Study the program to make sure you see the purpose of each line as related to the original description of what was desired. Experiment with various total prices and number of items until you see exactly how the program works.

Example 2 - Converting Temperatures

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

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

In this expression, C stands for degrees Celsius and F stands for degrees Fahrenheit. If, for example, F is 212, then C is determined to be

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

As in the first example, we will write the program after seeing how we want the output to appear. Let's suppose that if we RUN the desired program, we want to see the following typical output:

```
HOW MANY DEG. F
? 212
THAT'S 100 DEG. C
```

Notice that the first two lines of the desired output are slightly different than Example 1. In this case the question mark and input from the keyboard are on the second line. This is accomplished by omitting the semicolon at the end of the first message.

```
100 PRINT "HOW MANY DEG. F"
110 INPUT F
```

Now we compute the number of degrees Celsius using the relationship given above.

```
120 LET C=(5/9)*(F-32)
```

Finally we print out the last message and the answer.

```
130 PRINT "THAT'S ";C;" DEG.
C"
140 END
```

Line 130 illustrates how strings of characters and numeric variables can be printed out in the same PRINT statement. Since C is not in quotes, its numeric value is printed out.

The complete program is listed below.

```
100 PRINT "HOW MANY DEG. F"
110 INPUT F
120 LET C=(5/9)*(F-32)
130 PRINT "THAT'S ";C;" DEG
C"
140 END
```

As with Example 1 you might want to experiment with this program using different values of F.

Example 3 - Monthly Mortgage Payment

Now let's turn to an example which is more complicated (and also more useful). We want to write a program to compute monthly mortgage payments. The relation to compute this is

```
M = (PI/1200)/(1-1/(1+I/1200))^(12N)
```

In this relation P is the initial amount of the mortgage in dollars, I is the annual interest rate in percent, N is the length of the mortgage in years, and M is the monthly payment in dollars. We want the output to appear as follows when the program is RUN:

```
PRINCIPAL($) = 50000
INT. RATE(%) = 8.5
TERM (YEARS) = 30
MONTHLY PAYMENTS($)
384.4567450
```

As before, the input from the keyboard follows the prompt and represents a typical case. The monthly payment is shown as the computer will print it out. In a subsequent chapter, we will learn how to round off the value to the nearest cent.

By now, the first few lines of the program should follow without difficulty. Note that we are handling the messages and input slightly differently compared to the preceding examples.

```
100 INPUT "PRINCIPAL($) = ":
P
110 INPUT "INT. RATE(%) = ":
I
120 INPUT "TERM (YEARS) = ":
N
```

Using the values of P, I, and N that have been input, we must now compute the monthly payment. This will be done in three steps.

```
130 LET X=P*I/1200
140 LET Y=(1+I/1200)^(12*N)
150 LET M=X/(1-1/Y)
```

Study the original expression and lines 160, 170, and 180 until you are sure you understand how the computation is done. The final lines of the program are

```
160 PRINT "MONTHLY PAYMENTS(
$)"
170 PRINT M
180 END
```

The complete program is given below.

```
100 INPUT "PRINCIPAL($) = ":
P
110 INPUT "INT. RATE(%) = ":
I
120 INPUT "TERM (YEARS) = ":
N
130 LET X=P*I/1200
140 LET Y=(1+I/1200)^(12*N)
150 LET M=X/(1-1/Y)
160 PRINT "MONTHLY PAYMENTS(
$)"
170 PRINT M
180 END
```

This program has practical value when house hunting. You can quickly determine if a given house is within your economic means.

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 program that will call for the input of four numbers, 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, and 5. The program must work for any set of four numbers that you decide to type in. Oh yes, you can use only two lines in your program in addition to the END statement.

3. What will be output if we 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
```

4. 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
```

5. 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 (in feet) and T is the time of fall (in seconds). Using long variable names, write a program that when RUN will produce output similar to the following:

```
TIME OF FALL (SEC) ? 2
OBJECT FALLS 64 FEET
```

6. The volume of a box can be computed as V = LWH where L, W, and H are the length, width, and height, respectively. If these are all measured in centimeters, for example, the volume will be in cubic centimeters. We want a program that will produce output similar to the following when RUN:

```
LENGTH (CM) ? 4
WIDTH (CM) ? 2
HEIGHT (CM) ? 3
VOLUME IS 24 CUBIC CM.
```

The program below is incorrect and will not produce the output called for above. 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
```

7. In the program below two numbers, A and B, are called for in the INPUT statement. The problem is to 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
```

8. Suppose the odometer on your car reads R1 miles when the gas tank is full. You drive until the odometer reading is R2 at which point G gallons of gasoline are required to fill the tank. The computation to give you the miles per gallon you got on the drive is M = (R2 - R1)/G. Write a program to figure out the mileage for the following data:

R1	R2	G
21423	21493	5
05270	05504	13
65214	65559	11.5

9. 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%, and N = 5 years,

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

Write a program that when RUN will produce output similar to the following:

```
PRINCIPAL ?1000
INT. RATE (%) ? 6
TERM (YEARS) ? 5
TOTAL VALUE IS
1338.22558
```

10. 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/100J)^(JN)
```

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, b. semiannually, c. monthly, d. weekly, and e. daily. If a savings and loan company does a big advertising production about computing the interest every day instead of each week, should you get interested?

5-6 PRACTICE TEST

The practice test that follows is for you to check how well you have mastered the key points and objectives of the chapter. Check your answers against the key given at the end of the book.

1. What will be output if the following program is executed?

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

2. Describe three ways that numbers can be brought 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 the following program is RUN?

```
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 may there be in a program?

____

9. What is the TAB function used for in BASIC?

____

10. What will happen if the following program is RUN?

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

____

11. The program

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

is RUN, and in response to the INPUT prompt you type the numbers 10, 12, and 13. Describe exactly what will happen.

12. Miles can be converted to kilometers by multiplying by 1.609. Thus, 10 miles equals 16.09 kilometers, and so on. Write a program that will produce output similar to the following when RUN:

```
HOW MANY MILES ? 2.5
2.5 MILES IS THE
SAME AS 4.0225 KM.
```

CHAPTER

SIX

DECISIONS, BRANCHING, AND APPLICATIONS

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 capability and will go on with the continuing task of learning to program in BASIC. The objectives are as follows:

Making Decisions in Programs

Decisions made in a program 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 upon values of the variables in the program. The effective use of these conditional and unconditional transfer statements makes simple programs produce powerful and useful results.

Program Applications

As in the previous chapter, we will go on learning how to apply the techniques we study to BASIC programs.

Finding Errors in Programs

Almost all programs have errors in them when first written. Troubleshooting programs is a vital skill that, like programming itself, can be learned.

6-2 DISCOVERY ACTIVITIES

Let's go straight on to the computer work.

1. Bring up BASIC on your computer 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 translates as "If X is less than 5 then 110." Study the program carefully. What do you think will be printed out if you RUN the program?

RUN the program and record what did happen.

2. Now type

```
100 LET X=2
```

Display the program. What will be output now?

RUN the program and write down what was printed out.

3. Now let's make a few more changes in the program to see if you are following what is taking place. Type

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

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

____

Execute the program and see if you were right. Copy below what actually took place.

____

4. We want to explore another idea in connection with the program you have in memory, but need to make some changes. If desired, you can modify the program to make it agree with the one below or clear out 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 this program and record what happened.

____

Compare the output recorded above with that which you copied down in step 1. Is there any connection?

____

5. In the program in step 4 there is an assertion stated in line 130. The assertion is X >=5, which is read as "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 suppose we look closely at the program in step 4. If the program is RUN, 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 go to next?

____

6. Only two conditions have been used so far in the programs. They are < (less than) and >= (greater than or equal to). How would you write the conditions for "greater than"?

____

   What about "less than or equal to"?

____

   How about "equal to"?

____

   Finally, what about "not equal to"?

____

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

7. Now on to some applications using IF THEN statements. Clear out 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 ELSE 100
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 program is RUN and the computer types out the INPUT prompt, you are supposed to type in either 1, 2, or 3. Which value or values 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 out SWEAT. What value of Y should be entered?

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 out BLOOD?

____

How about making the computer type out TEARS?

____

Check each of the responses you made 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 in 4 in response to the INPUT prompt. What do you think will happen?

____

RUN the program, type in 4 in response to the input prompt, and record below what happened.

____

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. Of course, what happened when you typed in 4 was due to the ELSE in line 140. Now jump the computer out of the INPUT loop.

11. Clear the screen and clear the program from memory. Enter the following program:

```
100 A$="BLACK"
110 B$="WHITE"
120 C$="CAT"
130 D$="DOG"
140 INPUT X
150 ON X GOTO 160,180,200,22
0
160 PRINT C$
170 GOTO 140
180 PRINT D$
190 GOTO 140
200 PRINT A$&C$
210 GOTO 140
220 PRINT B$&D$
230 GOTO 140
240 END
```

The program has some new features. First, note that the character-string variables introduced in Chapter 3 are used in the program. The variables are defined in lines 100, 110, 120, and 130. Study the program a few moments to try to see what it does. Now let's try it out. RUN the program and at the INPUT prompt, type 1. What happened?

12. The program is waiting for more input. Type in 2. What happened?

This time, try the number 3.

Enter 4 and record what happened.

13. It should be clear by now that the program is being switched in line 150 to different line numbers depending on the value of X. We have four line numbers in statement 130, and have tried X = 1, 2, 3, or 4. What do you think will happen if we entered 10?

Try it and record below what happened?

We hope that by now you have figured out what is taking place. If not, don't fret as we will go over it again later. Jump the computer out of the INPUT loop.

14. One last program and we will be finished with the discovery activities. Clear the program from memory and enter the following:

```
100 INPUT A$
110 INPUT B$
120 IF A$<B$ THEN 160
130 PRINT B$,A$
140 PRINT
150 GOTO 100
160 PRINT A$,B$
170 PRINT
180 GOTO 100
190 END
```

It is clear that in this program the computer will expect character strings to be typed in at the INPUT prompts. The new and interesting idea in the program is in line 120. Look at this carefully. What do you think the "less than" symbol means with regard to character strings?

15. Now let's see how the program works. If you RUN the program and at the first INPUT prompt type CAT, and at the second input prompt type DOG, what do you think the computer will do?

____

Try it and record what happened.

____

16. All right, the computer has looped back and is waiting for more input. This time, type in the words ORANGE and APPLE. What happened?

____

Now try AARDVARK and ARK. Write down what was printed out.

____

This exercise opens the door to some very interesting non-numerical applications.

17. Jump the computer out of the INPUT loop. This concludes the discovery activities for now. Type BYE, turn your computer off, and go on to the discussion material.

6-3 DISCUSSION

In this chapter we are concerned with two topics. The first is the concept of the transfer statements, both conditional and unconditional, as well as their use in programs. The second topic is the very important skill of troubleshooting and tracing programs.

Transfer without Conditions

From the beginning of this 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 RUN 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 we 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, 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. This is why the GOTO statement is known as an "unconditional" transfer statement. It is also clear that, in this case at least, the GOTO statement 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 by pressing the shift-C key.

GOTO is an unconditional transfer statement.

To sum up, if at some point in a program you want the computer to 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.

Transfer on Conditions

By now you have most likely established the connection between the IF THEN statements you met in the computer work 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
```

All IF THEN statements have this same format. The IF and the 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.

IF THEN is a conditional transfer statement.

In all the examples we have used so far with the exception of the one above, the relations have been either numeric variables, character-string variables, or constants. This is the type of assertion most often used in programs. Examples might be

```
100 IF U<3 THEN 250
340 IF S$>T$ THEN 220
```

There are instances, however, in which we might want to use more complicated expressions in the IF THEN statements. In the example following the description of the IF THEN statement, the first relation was

```
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. To further illustrate what takes place in a program, 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 first substituting the values of X, Y, and Z. This changes the statement to

```
200 IF 1>6 THEN 360
```

Sooner or later, all IF THEN statements involving numeric variables come down to this form in which the computer must judge whether an assertion established by two numbers and a condition is true or false. If character-string variables are involved, the comparison is done differently, as will be pointed out later. In this case the assertion 1 > 6 is false. However, an assertion like 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.

The IF THEN statement branches if the condition is true. If the condition is false, the computer goes to the next higher line number.

We can employ a different version of the IF THEN statement if desired. An example of this new statement is:

```
300 IF X>Y THEN 240 ELSE 435
```

If the assertion in this statement is true, the program would branch to line 240; if false, the control would go to 435.

Several conditions may be used in the IF THEN statements. These 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

Multiple Branch Statements

In the computer work we saw that it was possible to branch a program to several different points using only a single statement. Let's use the following program segment to see how this is done.

```
200 ON A GOTO 310,320,330
210 B=A+2
```

In line 200 the decision concerning which line to branch to is based on the value of A. If, for example, A were 1, the program would branch to the first line number in the list. In this case that would be line 310. Likewise, if A were 3, the program would branch to line 330, the third number in the list.

In the example above A should be either 1, 2, or 3 since there are three line numbers in the branch list. You might wonder what would happen if A had some other value, say 8. The answer is that when the computer is unable to locate an appropriate line number from the branch list, it prints * BAD VALUE IN 200 and stops. The line numbers in the branch list following ON GOTO do not have to be in any particular order. Moreover, the same line number can be repeated in the list if desired. If you think about this a bit, you can see there is a lot of power involved here.

The ability to control the branching process by changing the values of a numeric variable is the heart of the ON GOTO statement. This multiple branch statement provides a very useful switching device that has many applications in BASIC programs.

Non-Numeric Branching

As you have seen, we can use character-string variables in IF THEN statements. The comparison between strings of characters is based on the alphabetic position. Thus, A is less than B because A occurs before B in the alphabet. Likewise, Z is greater than T since it occurs after T.

We can extend this idea to words in which case the comparison is made character by character. For example, CAT is greater than CAP. The first two characters in both words are identical, hence no difference is detected in the character strings. However, on the third character T occurs after P, so CAT is judged to be greater than CAP. In the case of character-strings of unequal length, the comparison is made as far as possible, limited by the length of the shorter character string. Thus, CAT is less than CATALOG. The comparison is equal for the first three characters (the length of the shorter character string), but there are characters following this in CATALOG, hence the judgement. Of course, CAW would be judged greater than CATALOG.

Once this idea of character comparison is understood, character-string variables can be used in conditional transfer statements in the same manner as numeric variables. It should be clear that this capacity to compare character strings is very powerful and makes sorting and alphabetizing lists of words very simple. We will see several examples of this later on.

6-4 PROGRAM EXAMPLES

Up to this point our programs have suffered from a serious fault. On one hand, the program might involve repetition but there was no way to stop the process. On the other hand, the program stopped but often tended to be trivial. What we want is a way to have the program accomplish a useful task (which may involve repetition) and then shut itself off. The conditional transfer statements just learned provide a mechanism to do this. Now we will look at several programs that illustrate this capability.

Example 1 - Printout of Number Patterns

Our problem is to write a program that will print out the following number pattern when RUN:

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

There are several characteristics of this pattern which we must think about when writing the program. The first number is 2, and succeeding numbers are spaced across in the standard spacing (two numbers to a line). Each number is 1 greater than the previous one. The last number printed out is 9, then the computer should stop.

Several solutions are possible. A program that is not the most elegant but would still work is

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

You might check this program to see that it does in fact produce the correct number pattern. It also illustrates a very important concept. There really 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. The beginner should be concerned with whether or not the BASIC program will produce the desired results, not with questions of style.

Now back to the problem at hand. One way to approach the problem is to make the computer print out the first number in the pattern. We also want to organize the program so that only a single print statement is required. This will require that the program print out the value of a variable that will be changed as the program runs. We can start our 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 we must generate the next value to be printed. Note that at any point in the number pattern, the next number is just 1 more than the present number. This can be done with

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

Now all that remains is to make a decision about whether or not to loop back to the PRINT statement. As long as X is less than or equal to 9, we want to loop back. We can do this with a conditional transfer statement.

```
130 IF X<=9 THEN 110
```

The program is finished by an END statement.

The complete program is

```
100 LET X=2
110 PRINT X,
120 LET X=X+1
130 IF X<=9 THEN 110
140 END
```

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

Example 2 - Automobile License Fees

Let's assume that in an attempt to force consumers to use lower-horsepower cars and conserve energy, the state adopts a set of progressive license fees based upon 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

We want a program that will produce the following typical output when RUN.

```
INPUT AUTO HP ? 325
LICENSE FEE IS 500

INPUT AUTO HP ? 85
LICENSE FEE IS 30

(etc.)
```

Clearly, the only difficult part of the program will be to decide what the fee is. This decision-making process is made to order for the IF THEN statement. To get started we must provide for input of the power rating. We will use P to stand for the power rating of the car. Follow through the development of the program, but don't attempt to type it in the computer until it is complete. We will leave parts of the program out initially and will return later to fill in the details. If you attempted to type in the lines with details missing, the computer would signal errors.

The program can begin with

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

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

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

The line number following THEN is missing for a reason. 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 will return later and insert the proper value. The note after the blank line number is there to remind us 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. In that case we want to see if 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. There are three branch statements left to determine completely which category contains 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<=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)
170 LET F=0
180 GOTO          (PRINT statement)
```

Again, in line 180 we have a missing line number. The reminder is that we want to transfer to a PRINT statement. 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 go on filling 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 we may as well use for the PRINT statement. The rest of the program follows easily. The complete 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
```

Now that all the missing line numbers have been supplied, you can enter the program into the computer and verify that it works properly.

You may have noticed that the conditional transfer statement in line 160 is not necessary. To see why, consider the assertions in the IF THEN statements. If the assertion in line 120 is false, we know that P must must be greater than 50. Likewise, if each of the following assertions are false, the computer goes to the next higher line number. In particular, suppose the computer reaches line 150 and determines that the assertion is false. This directs the computer to line 160, but then we know that P must be greater than 300 and can therefore print out the fee without any more testing. If we assign the license fee of $500 in line 160, the result is 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
```

Both versions of the program will work equally well, and you may have your own version. How you prefer to handle the branches is a matter for you to decide. The only question to be answered is whether your program works or not.

We have gone through this program in detail because it often proves difficult for the beginner to write programs involving such search rules. You should study the program until you are convinced that it does accomplish what was desired. Also, try to remember to use the technique of leaving line numbers out when you do not know what they should be, then returning later to fill in the proper values. The comments at the right in these cases will help you remember what you want to happen at that branch point in the program. However, also remember that if you leave line numbers out while writing the program, don't try to enter the lines into the computer until the program is complete.

Example 3 - Averaging Numbers

Suppose we have numbers in a DATA statement which we wish to average. The problem is that we don't know in advance how many numbers there are. So, we will use the strategy of 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 for our flag, but you could select one of your own choice if desired.

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

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

The flag 9999 is put in the data after the last number to be averaged. In the program, each time we read a number from the DATA statement we must check to see if it is 9999. If not, we know that the number just read is part of the data to be averaged. If the number is 9999, we know that we have read in all the data and can go on to the rest of the program.

An average is computed by dividing the sum of the numbers by the number of numbers. In our program we must compute both these quantities. We will use S to stand for the sum of the numbers and N for the number of numbers. When the program is executed, we do not know what these values will be, so we must set them equal to 0 and then develop their values as we read in numbers from the DATA statements.

The programs begins by setting up the initial values of S and N.

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

We really didn't have to do this since the computer will automatically zero out numeric variables. However, it makes the program easier to understand if the statements are present. Now we can read a number from the DATA statement and check for the flag value.

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

We are using the method, introduced previously, of leaving a line number blank in the conditional transfer statement until we know what it should be. In this case, if the assertion (X = 9999) is true, then we know that all the numbers in the DATA statement have been processed and we are ready to compute the average. If the assertion is false, then the number just read must be part of the data and should be processed. This is done 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 incremented by 1 to record the fact that another number has been processed.

Having processed the value of X, we loop back to the READ statement to continue the process.

```
160 GOTO 120
```

Now we can fill in the missing number in line 130, since the next line number in the program would normally be 170. In line 170 we compute the average, which we will 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, we can have as many DATA statements as needed to hold 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 us out of the READ loop and lets us know when to go on to compute the average. The conditional transfer statement, coupled with the idea of a flag variable, gives us a powerful tool to use in programs.

Example 4 - Mortgage Down Payment

The down payment required on a mortgage is determined by the total amount of the mortgage. Suppose a bank has the following set of rules: 20% of the first $75,000, 15% of the next $35,000, 10% of the remainder up to $150,000, and no loans made in excess of $150,000.

Our problem is to write a program to call for the input of the amount to be borrowed, then compute and print out the required down payment. If the amount exceeds $150,000, we will output a message that no loan can be made.

First, let's call for the input of the value to be borrowed.

```
100 PRINT "AMOUNT OF MORTGAG
E";
110 INPUT P
```

Now we should check to see that P is not greater than the limit.

```
120 IF P<=150000 THEN 150
130 PRINT "NO LOAN ALLOWED"
140 GOTO      (END statement)
```

We will leave the line number blank in line 140 until we know what the line number of the END statement is. The comment at the right is to remind us of where the transfer is to be. As an aside, commas are usually used in big number. For example, one hundred fifty thousand is written as 150,000 with the comma setting off the thousand position. Remember though that commas can't be used in numbers in BASIC programs because the commas are used to separate the numbers. Thus, if you typed in 150,000 to represent one hundred fifty thousand, the computer would assume you meant the two numbers 150 and 000. Using commas in this way is an easy mistake to make!

Next, we should check to see if P is greater than or equal to $110,000, or greater than or equal to $75,000. Depending on the outcome we can compute the down payment.

```
150 IF P>=110000 THEN   (?)
160 IF P>=75000 THEN    (?)
170 LET D=.2*P
180 GOTO          (PRINT statement)
```

Notice that if the assertion in lines 150 and 160 are false, we know that P is less than $75,000 and can compute the down payment in line 170. The blank in line 180 will be the line number of the final PRINT statement when we know it. Since the next line would be 190 we can use it for the missing line number in line 150.

```
150 IF P>=110000 THEN 190
160 IF P>=75000 THEN    (?)
170 LET D=.2*P
180 GOTO          (PRINT statement)
190 LET D=.2*75000+.15*35000
+.1*(P-110000)
200 GOTO          (PRINT statement)
```

Now we can use line number 210 for the missing line number in line 160.

```
150 IF P>=110000 THEN 190
160 IF P>=75000 THEN 210
170 LET D=.2*P
180 GOTO          (PRINT statement)
190 LET D=.2*75000+.15*35000
+.1*(P-110000)
200 GOTO          (PRINT statement)
210 LET D=.2*75000+.15*(P-75
000)
```

The PRINT statement can go in line 220, followed by the END statement.

```
220 PRINT "DOWN PAYMENT IS "
;D
230 END
```

Now we know that the PRINT statement is line 220 and the END statement is line 230. Putting these numbers in the appropriate blanks, we pull together the complete program.

```
100 PRINT "AMOUNT OF MORTGAG
E";
110 INPUT P
120 IF P<=150000 THEN 150
130 PRINT "NO LOAN ALLOWED"
140 GOTO 230
150 IF P>=110000 THEN 190
160 IF P>=75000 THEN 210
170 LET D=.2*P
180 GOTO 220
190 LET D=.2*75000+.15*35000
+.1*(P-110000)
200 GOTO 220
210 LET D=.2*75000+.15*(P-75
000)
220 PRINT "DOWN PAYMENT IS";
D
230 END
```

6-5 FINDING ERRORS IN PROGRAMS

The ability to look at a program and determine whether or not it will accomplish what it is supposed to do is certainly one of the most important skills a beginner can acquire. Probably more to the point, when a program is not doing what it is supposed to do, can you find out what is wrong and correct it? These abilities are strange in that until learned, they appear to be very difficult. However, once learned, the programmer usually has great difficulty understanding why everyone doesn't have the same abilities.

Two separate tasks are involved in troubleshooting programs. First, you must be able to translate a BASIC statement into what it means to the computer. Next, you must be able to trace a BASIC program, detailing each step and action as it takes place. We are now far enough into the task of learning about BASIC that we can profitably spend some time on troubleshooting programs. The time spent doing this is golden and will be paid back many times over in time saved in the future.

Translating BASIC Statements

We have been using several different types of BASIC statements. We want to review just what the computer does when it executes these statements. As an example, suppose the computer evaluates the statement

```
140 LET X=3
```

This statement instructs the computer to set up a memory location, name it X, and store a 3 in that location. Likewise

```
160 LET B=0
```

causes the computer to name a memory location B, and store a zero in that location.

The situation is a bit more complicated with the following statement:

```
135 LET X=A+B-2
```

Now the computer is directed to get the numbers stored in A and B, add them together, subtract 2, and store the result in a location to be named X. This is all right provided that the computer can find memory locations named A and B. If these have not been set up prior to the statement being executed, the computer will search for the locations A and B, and finding none, will set them up, place zeros in both locations, and proceed. Of course this might not be what we wanted at all, so this is something to be careful about.

What happens when the computer encounters a statement like

```
185 IF M=N THEN 240
```

which directs the computer to get the numbers in M and N and see if they are equal? If the numbers are equal, then the next line number to be executed would be 240. If not, the computer would go to the next higher line number in the program. If the computer can't find locations M and N, it will set them up containing zeros. This ensures that the assertion will be true. Again, we must be careful to see that all variables are set up as dictated by the problem or strange things may happen!

Now we want to use the knowledge of how to translate BASIC statements to locate any errors that may be in a program.

Troubleshooting BASIC Programs

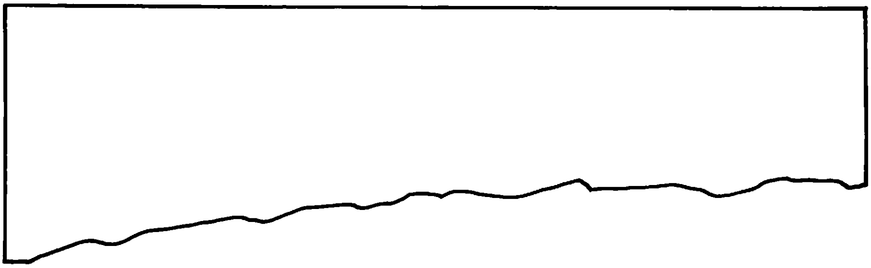
The program developed in Example 3 in the previous section will be a good one to use to learn how to troubleshoot. The program is given again below for your reference.

```
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
```

The job at hand now is to convince you that the best and most foolproof aid to programming is a blank sheet of paper! Used correctly, this "little dandy" programming aid will enable you to find all the errors in your programs and reveal how to correct them. This sounds like a big order for such a simple device as a blank sheet of paper, but it's true! Later we will see how to use special features of the TI Home Computer to help you troubleshoot programs, but first we will see how to do it by hand.

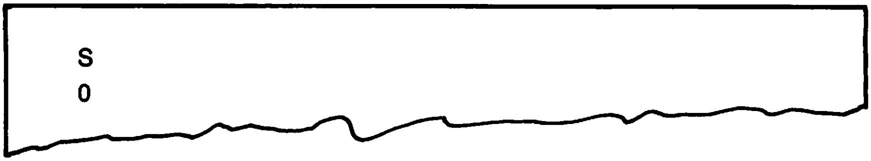
First, copy the program on a lined sheet of paper and follow through our discussion using this copy. Place a blank sheet of paper over everything except the first line of the program.

```
100 LET S=0
```



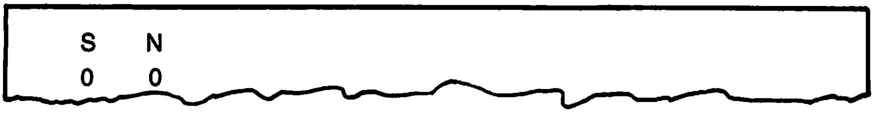
Now we translate the statement, which tells the computer to set up a memory location called S, and store a zero there. We will use our blank sheet of paper to keep track of what is in the computer memory. So we write down an S and underneath place a 0.

```
100 LET S=0
```



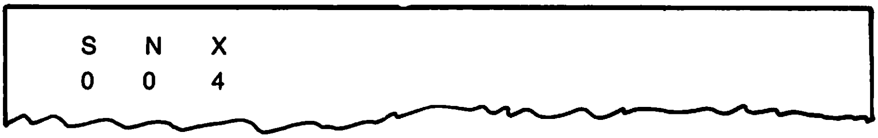
This finishes the first line in the program. Slide the sheet of paper down to reveal the next line and do what is directed. Remember that you are playing the part of the computer and are using the sheet of paper to record what is in the computer memory as well as to let you see only one line of the program at a time.

```
110 LET N=0
```



Now on to line 120.

```
120 READ X
```



Here the computer is instructed to read a number from the DATA statement in the program, which in this case is 4. The 4 is stored in a location called X.

Let's pause to review what we are doing. We are going through the program one line at a time, writing down what the computer is directed to do. Since we have yet to meet any transfer statements, we simply evaluate a statement, then go on to the next higher numbered statement. Now on to line 130.

```
130 IF X=9999 THEN 170
```

S	N	X
0	0	4

The assertion in line 130 (X = 9999) is evaluated using the value of X that appears on the paper. Since at this point in the program, X has the value 4, the assertion (4 = 9999) is false. Consequently, instead of going to 170, we drop through to the next line in the program.

```
140 LET S=S+X
```

S	N	X
~~0~~	0	4
4		

We get the number in S (0) and the number in X (4), add them together, and store the sum of 4 in S. Note that this destroys the previous value stored in S. We will simply line out any destroyed value to indicate that it has been lost. At any point in our analysis of the program, the value of a numeric variable will be the last number written down in that column. Now the computer goes to line 150.

```
150 LET N=N+1
```

S	N	X
~~0~~	~~0~~	4
4	1	

Here the number 1 was added to the 0 in N, and the sum was then stored in N, destroying the 0 stored there previously. Line 160 directs the computer to go back to the READ statement in line 120. Then the whole process starts again. We stay in this loop until all the data are read in and processed. If you keep tracing the program until the flag 9999 is read into X, your sheet of paper should look as follows:

```
130 IF X=9999 THEN 170
```

S	N	X
~~0~~	~~0~~	~~4~~
~~4~~	~~1~~	~~2~~
~~6~~	~~2~~	~~3~~
~~9~~	~~3~~	~~6~~
~~15~~	~~4~~	~~5~~
20	5	9999

Since the value of X is now 9999, the assertion (X = 9999) is true, and the computer is branched to line 170.

```
170 LET A=S/N
```

S	N	X	A
~~0~~	~~0~~	~~4~~	4
~~4~~	~~1~~	~~2~~	
~~6~~	~~2~~	~~3~~	
~~9~~	~~3~~	~~6~~	
~~15~~	~~4~~	~~5~~	
20	5	9999	

The computer sets up a location called A, divides the number in S by the number in N, and stores the result in A. Finally, the computer is directed in line 180 to print out the value stored in A. Our analysis has revealed that the computer is doing what we intended and is producing the correct results.

Now let's look at a program that is incorrect and use the the technique described above to find out what is wrong. The program is supposed to compute the sum of numbers typed in from the keyboard. Each time the computer prints out an INPUT prompt (the question mark), we type in one number. When all the numbers are in, we type in 11111 as a flag to indicate that we are through. The computer is then supposed to type out the sum of the numbers entered prior to the flag. The program below is incorrect.

```
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
```

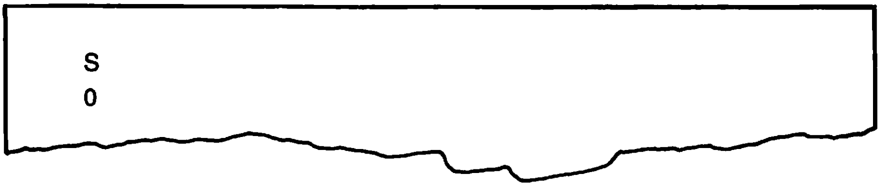
We will use our little dandy programming aid to find out what is wrong. To test the program we will assume that the following sequence of numbers is typed in as the INPUT prompts are displayed:

3, 1, 6, 5, 11111

The sum of the numbers before the flag is 15, so we know in advance that this is what the computer should print out.

We begin with the blank sheet of paper and the first line of the program,

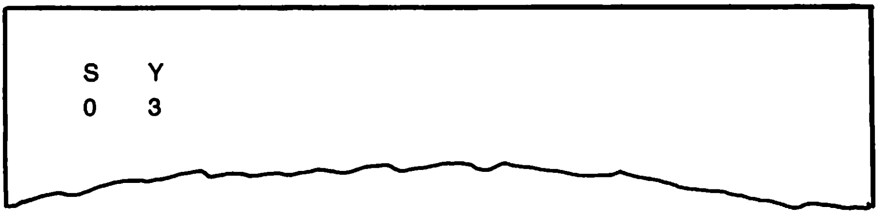
```
100 LET S=0
```



S
0

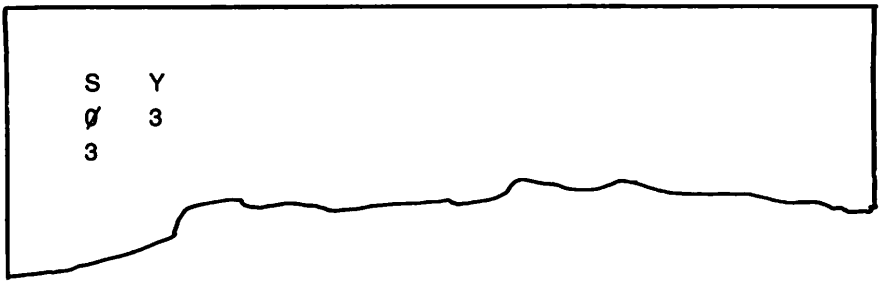
Then

```
110 INPUT Y
```



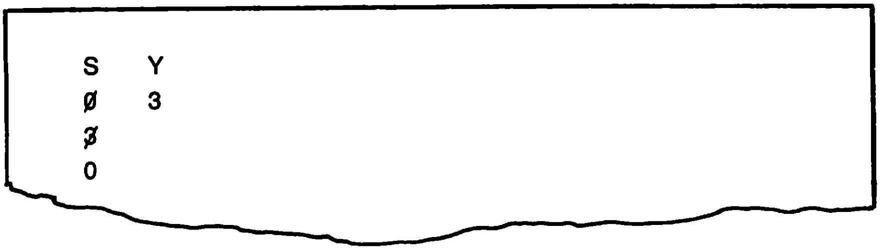
Since Y is not 11111, we go to line 130,

```
130 LET S=S+Y
```



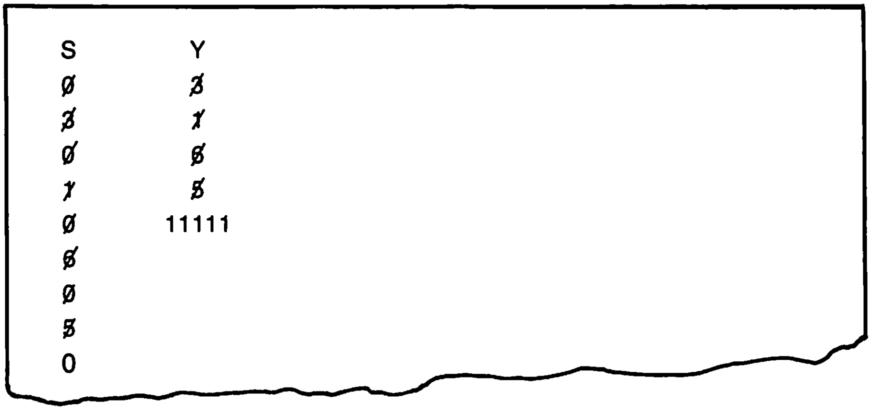
After line 130 we transfer back to line 100,

```
100 LET S=0
```



If you follow the program until the flag 11111 is entered, your sheet of paper should look as follows:

```
120 IF Y=11111 THEN 150
```



S	Y
~~0~~	~~3~~
~~3~~	~~1~~
~~0~~	~~6~~
~~1~~	~~5~~
~~0~~	11111
~~6~~	
~~0~~	
~~5~~	
0	

Since at this point, Y contains the value 11111, the computer jumps to line 150, which calls for the number in S to be printed out. But the number in S is 0, which is clearly incorrect. If you followed through, tracing the program and writing down all the steps, then you have probably already discovered what is wrong. The error is in the unconditional transfer statement in line 140. With the transfer to line 100, the value in S (which is supposed to contain the sum of the numbers as they are typed in) is set equal to 0 each time a number is entered. The problem is easily corrected by changing the line to

```
140 GOTO 110
```

Several features have been included in BASIC for the TI Home Computer to assist you to troubleshoot a program and find out what is wrong. To illustrate this, let's go back to the program to compute averages.

```
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
```

If before you RUN a program you type TRACE, and then RUN, the comuter will print out the line numbers as it goes through the program. For the program above, this would produce the following screen display:

```
<100><110><120><130><140>
<150><160><120><130><140>
<150><160><120><130><140>
<150><160><120><130><140>
<150><160><120><130><140>
<150><160><120><130><170>
<180> 4
<200>
```

If you compare the line numbers printed out by the TRACE to the program, the flow followed by the computer can be seen. In this case it happened to work out that the READ statement in line 120 wound up in a vertical column in the trace. After six READS (the last one was the data 9999), the branch is to line 170. After 180 the answer of 4 is printed out. Note that since there is no punctuation following PRINT A in the program, the next value in the trace is printed at the left side of the next line on the screen.

It should be clear that the ability to turn on the TRACE function and follow a program by line numbers as it is executed by the computer is a very powerful tool. However, if a program does run, and is still not producing correct results, the error may not be revealed by turning on TRACE.

The TRACE function stays on once it is turned on. When you are finished tracing a program, you can turn off the trace facility by typing UNTRACE.

Another useful tool is the BREAK command. Again, referring to the averaging program above, if you type

```
BREAK 130,150
```

The computer will stop when line 130 is encountered. At this point you are back in the immediate mode. Thus, if you type PRINT X, the computer will type out the current value of X. This permits you to inspect (or for that matter, to change) any of the variables in the program. When you are ready to go on, type CON (for continue) and the computer will pick up where it left off.

Each time a break point is encountered and the computer halts, the break point is removed. Thus, if the computer loops through a program segment containing breakpoints, they will stop the computer only the first time through. There is a way around this as will be explained below.

To remove the BREAK facility, type UNBREAK. If you want to remove only certain breakpoints, specify which ones you want. An example might be

```
UNBREAK 140
```

TRACE, UNTRACE, BREAK, and UNBREAK can be used in program statements in BASIC programs. Suppose you suspected there were problems in a segment of a program. Use of these commands in program statements can help you locate the errors. An example of how this might be done is

```
(top part of the program)
        •
        •
540 TRACE
        •
        •
620 BREAK
        •
        •
780 UNTRACE
        •
        •
(balance of the program)
```

In this hypothetical example, when line 540 was reached, the TRACE facility would be turned on and the line numbers would be printed out as the computer went through the program. When the computer reached line 620, the BREAK command would generate a halt, and we would be free to inspect the variables in the immediate mode. When we type CON, program execution picks up at the point of interruption. Finally, the trace facility would be turned off in line 780.

The BREAK and TRACE facilities provide tools that permit you to pick your way through very complicated programs and see exactly what is happening. Add this powerful capability to the "little dandy" paper and pencil method, and you should be able to troubleshoot any program!

Take the time to learn how to troubleshoot your work If you don't, much time will be lost later on in wasteful speculation about what is wrong with your programs.

6-6 PROBLEMS

1. Write a BASIC program to call for the input of two numbers. Then print out the larger.

2. Write a BASIC program to READ three numbers from a DATA statement and then print out the smallest.

3. Write a program to compute and print out the sum of all the whole numbers between 1 and 100 inclusive.

4. Describe in your own words what will happen if the following program is RUN.

```
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, substitute the following DATA statement:

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

   Troubleshoot the program by hand and write down what will be output if the program is RUN.

6. Troubleshoot the program below by hand using the inputs indicated. In each case, find what will be printed out. The inputs are

   a. 1, 2, 3

   b. 3, 2, 1

   c. 2, 2, 2

   d. 3, 1, 3

```
100 INPUT A,B,C
```

```
110 IF A<B THEN 150
120 IF B>=C THEN 170
130 LET D=A+B+C
140 GOTO 180
150 LET D=A*B-C
160 GOTO 180
170 LET D=A+B*C
180 PRINT D
190 END
```

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. There is an interesting sequence of numbers called the Fibonacci numbers. The set begins with 0, and 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, ... and so on. Write a BASIC program to compute and print out the first twenty numbers in the Fibonacci sequence.

9. 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 larger and the smaller.

10. 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– 49	F

Write a program to produce the following typical output when RUN:

```
INPUT EXAM GRADE ? 73
YOUR GRADE IS C
```

11. If you use 8 percent more electricity each year, in nine years your consumption will double. Thus, your "doubling time" is nine years. It turns out that 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. This is the rule of seventy-two. 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

$$Q_{\text{new}} = Q_{\text{old}}(1 + R/100)$$

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 would be the doubling time. Using this approach, write a program that will produce the following typical output when RUN:

```
GROWTH RATE (%) ? 3
NUMBER OF GROWTH PERIODS
TO DOUBLE IS 24
```

Use the program to check out the accuracy of the rule of seventy-two for many different growth rates.

12. A set of integers (whole numbers) is chosen at random from the set 1, 2, 3, and 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
```

6-7 PRACTICE TEST

Check your progress with the following practice test. The answers are in the key at the end of the book.

1. What will be output if the following program is RUN?

```
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 the following program is RUN?

```
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 that you decide to buy a number of widgets. The manufacturer is pushing sales and will give reduced prices for quantity purchases. The price detail is as follows:

# 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 typical output when RUN:

```
HOW MANY WIDGETS ? 40
PRICE PER WIDGET IS 1.8
TOTAL COST OF ORDER IS 72
```

Then keep looping back through the program.

4. Write a program that will print out the number pattern shown below and then stop. Assume that the numbers are spaced in standard column spacing.

```
0                 5
10                15
20                25
        etc.
100               115
```

5. If you get a ticket for speeding, your fine is based on how much you exceeded the speed limit. 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 typical output when RUN:

```
SPEED LIMIT ? 45
SPEED ARRESTED AT ? 56
FINE IS 10 DOLLARS
```

CHAPTER

SEVEN

LOOPING AND FUNCTIONS

7-1 OBJECTIVES

In this chapter we will learn about two interesting characteristics of BASIC which will provide new and powerful programming capability. The objectives are as follows.

Built-in Looping

We have already learned how to loop programs using either the unconditional or conditional transfer statements. BASIC has special statements to take care of looping automatically. These statements simplify programming and provide flexibility in programs.

Built-in Functions

BASIC contains a number of built-in functions that can be called on to perform specific tasks. We will look at some of the simpler of these functions and see how they can be used to advantage in BASIC programs.

Program Applications

We 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 for the TI Home Computer.

7-2 DISCOVERY ACTIVITIES

We will go straight to the discovery activities.

1. Turn your computer on, and bring up BASIC. Then 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 RUN it. Record what happened.

Which statement in the program determines the difference in the numbers that were typed out?

---

2. Clear out the program. Now 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 with that obtained from the program in step 1.

3. Since the two programs just executed produce the same output, it is reasonable to assume that the statements must be related in some way. Modify line 100 to

read as follows:

```
100 FOR Y=10 TO 50 STEP 10
```

Display the program and study it. What do you think will happen if we RUN the program?

____

See if you were right. RUN the program and record the results below.

____

4. Now let's try a few different ideas out on the program. Modify line 100 to read as follows:

```
100 FOR Y=0 TO 5 STEP 1
```

Display the program. What do you think this program will do?

____

RUN the program and write down the output below.

____

5. Now change line 100 to

```
100 FOR Y=0 TO 5
```

Display the program. What do you think this program will do?

____

RUN the program and record what happened.

Now compare line 100 in the program just RUN 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. Change line 100 to read as follows:

```
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. All right, now change line 100 to

```
100 FOR Y=10 TO 20 STEP -2
```

Display the program. What do you think will happen now if we RUN the program?

RUN the program and write down what happened.

What we have done here is to lead 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 without any problems. Let's try a new step size that might not come out even when compared with the limits in the FOR NEXT statement. Change line 100 to read

```
100 FOR Y=2 TO 9 STEP 3
```

Display the program. Write down what you think will be printed out?

RUN the program and record what happened.

9. We will go on now to some more involved situations involving FOR NEXT statements. Use the NEW command to clear out the program in memory. 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 change line 100 to read

```
100 FOR X=1 TO 2
```

RUN this new program and record the output.

Compare the two number patterns you have just 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. Change lines 100 and 110 to read as shown below.

```
100 FOR X=1 TO 3
110 FOR Y=1 TO 2
```

Display this program and study it. What do you think will be output if it is RUN?

Try it and see if you were right.

12. One more time. Change lines 100 and 110 to read

```
100 FOR X=1 TO 2
110 FOR Y=1 TO 2
```

Display the program and write down what you think will be printed out when the program is RUN.

RUN the program and record the results below.

Clear the screen and LIST the program. Mentally, draw a line from the FOR X statement to the NEXT X statement. Do the same thing for the FOR Y and the NEXT Y statements. Do these imaginary lines cross?

13. Now change lines 100 and 110 as follows

```
100 FOR Y=1 TO 2
110 FOR X=1 TO 2
```

Display the program. Now, what do you think will be output by this program?

RUN the program and record what happened.

Clear the screen and LIST the program. Again, draw imaginary lines between the FOR X and NEXT X line numbers as in step 12. Do the same thing for the FOR Y and the NEXT Y statements. Do these lines cross? Compare with the same 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. In Chapter 5, we experimented with the TAB function to get variable spacing in the output. Now that we have the FOR NEXT statements at our disposal, let's go back to the TAB function. Clear out the program in memory, and enter the following program:

```
100 FOR X=1 TO 5
110 PRINT TAB(X);
120 FOR Y=X TO 5
130 PRINT "Y";
140 NEXT Y
150 PRINT
160 NEXT X
170 END
```

Take a few moments to trace the program using the technique developed in the last chapter. Be sure to take the program step by step and write down all the values of the variables in the program as they occur. What output do you think the program will produce?

See if you were right. RUN the program and record the output below.

15. Clear out the program you have in memory. Now enter the program below.

```
100 INPUT A
110 B=SQR(A)
120 PRINT B
130 GOTO 100
140 END
```

RUN the program and at the INPUT prompt, type 4. What happened?

____

Now type in 9 and record the results.

____

One more time. Type in 25. What happened?

____

Finally, type in 10. What happened?

____

All right, 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. Now change line 110 to read

```
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. Modify line 110 to read as follows:

```
110 LET B=SGN(A)
```

Display the program. Review the program structure to refresh your memory about how it 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 above carefully. What does the SGN function do?

18. On to the next function. Jump the computer out of the INPUT loop. Change line 110 to read

```
110 LET B=ABS(A)
```

Examine 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	
2	
-8.45	
8.45	

Examine the output. What does the ABS function do?

___

19. Now let's go back to the concept of the character-string variable that was introduced earlier. In particular we want to investigate the characteristics of some functions that pertain to character strings. Clear out the program in memory, clear the screen, and then enter the following program:

```
100 LET A$="ELECTRONIC"
110 LET B$="CALCULATOR"
120 LET C$=SEG$(A$,1,2)
130 PRINT C$
140 END
```

The new topic here is the SEG$ function in line 120. Can you guess what this does?

___

RUN the program and record what was printed out.

___

20. OK, now change the SEG$ in line 120 to read

```
120 LET C$=SEG$(A$,1,4)
```

RUN the program and record what happened.

___

Have you figured out what the SEG$ function does yet?

____

21. Let's try this once more. Change the SEG$ function in line 120 to read

```
120 LET C$=SEG$(B$,3,4)
```

What will happen now?

____

RUN the program and see if you were right.

____

22. Change line 120 to read as follows:

```
120 LET C$=SEG$(A$,10,1)
```

Now what will be output?

____

See if you were right. Record the output below.

____

23. Change the SEG$ function to read

```
120 LET C$=SEG$(B$,2,5)
```

What will the computer print out now?

RUN the program and write down what happened.

24. By now you should have a good idea of what the SEG$ function does. Let's try something different. Change line 120 to read

```
120 LET C$=SEG$(A$,8,10)
```

We are asking for a bigger piece of the string than there is! What do you think will happen?

RUN the program and record what happened.

25. Finally, change the SEG$ function in line 120 to read

```
120 LET C$=SEG$(B$,5,-2)
```

Now what will happen?

RUN the program and record what took place.

____

26. So much for the SEG$ function. Clear out the program in memory and enter the following:

```
100 INPUT A$
110 PRINT LEN(A$)
120 GOTO 100
130 END
```

RUN the program and when the input prompt comes up, type in CAR. What happened?

____

27. Type in a word with a different length. What was typed out?

____

Try a number of different words. What does the LEN function do?

____

28. Jump out of the INPUT loop. Clear the program from memory. Now enter the program below.

```
100 INPUT N
110 PRINT CHR$(N)
120 GOTO 100
130 END
```

The new function in this program is the CHR$ function. RUN the program and at the input prompt, type in the number 65. What happened?

____

29. The computer is waiting for another number. This time type in 90. What was printed back?

____

Do you see what the CHR$ function does yet?

____

30. Experiment with this program. Type in numbers in the range 33 to 90. You should see rather quickly what is going on. Explain in your own words (if you can) what the CHR$ function does.

____

31. Jump the computer out of the INPUT loop. Clear out the program from memory and enter the following:

```
100 INPUT A$
110 LET N=VAL(A$)
120 PRINT N
130 GOTO 100
140 END
```

In this program we will examine the VAL function in line 110. Note that the program asks for the INPUT of a string. This string (A$) is acted upon by the function and assigns the numeric results to the numeric variable N which is printed out. RUN the program and at the input prompt, type in the number 25 (remember that the computer is treating this as a string). What happened?

32. Try several different numbers for inputs. In each case record what was returned from the program

Do you have a clue yet as to the purpose of the VAL function?

33. Now let's try something different. Type in A3B6. What happened.

This is enough for now. Hopefully you have begin to see what VAL does. We will discuss it completely later in the chapter. Jump the computer out of the INPUT loop.

34. Now, on to the next function. Clear out the program in memory. Enter the following program.

```
100 INPUT N
110 LET A$=STR$(N)
120 PRINT A$
130 GOTO 100
140 END
```

This program is a reversal of the previous one in step 31. That program asked for a string input and printed out a number. This one asks for a numeric input and prints out a string. RUN the progrm and when the input prompt comes up, type in 45. What was printed out.

35. Try several different numbers. What was printed out in each case?

Have you figured out what STR$ does yet?

36. This time, type in ABC. What happened?

We will go back over this later. Obviously the VAL and the STR$ functions are closely related. Jump the computer out of the INPUT loop.

37. Clear out the program in memory and enter the following:

```
100 LET A$="MISSISSIPPI"
110 LET B$="IS"
120 PRINT POS(A$,B$,1)
130 END
```

RUN the program and record below what was printed out.

38. Change the POS statement in line 120 to read POS(A$,B$,3). Now RUN the program and record what happened.?

39. Finally, change B$ in line 110 to "SI" and RUN the program. What was printed out?

Do you see what the POS function does? If not, don't worry. We will review this latter.

40. This concludes the computer work for now. Turn your computer off and go on to the discussion material.

7-3 DISCUSSION

The techniques explored in the computer work can bring new power to the programs we write. We need to understand exactly how these new tools can be used to best advantage.

Built-In Looping

In the previous chapters we learned how to loop programs under the control of transfer statements. The unconditional (GOTO) statement was useful but sometimes resulted in a loop with no way out. The conditional (IF THEN) statement provided a way to loop the program as desired and also a way to get out of the loop. Both of these are good techniques. However, BASIC has a very elegant way to take care of looping which takes a large burden from the back of the programmer. 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 only things that can change or that are different 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. We can write many different forms of the FOR statement. A few are given below to illustrate the range of possibilities.

```
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 U
```

In general, we can write any legal BASIC statement in the relations involved in the FOR statement provided, of course, that the variables used have been properly defined in the program.

Use FOR NEXT statments to control looping in BASIC programs.

The FOR statement opens a loop. We close the loop with the NEXT statement. How this is done is illustrated in the following example:

```
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 completely understand how these loops work. In the example above, when the computer reaches line 200 the first time, X is set equal to 2. Then the computer works through the lines until line 340 is reached. This closes the loop and directs the computer back to the line following 200 and the next value of X, which in this case would be 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 next line number following the NEXT statement used to close the loop. Let's look at an example to see the FOR NEXT statements in action once more.

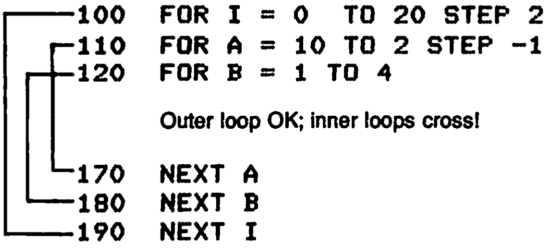
```
100 LET A=1
110 FOR X=1 TO 6 STEP 2
120 LET A=2*A
130 PRINT A,X
140 NEXT X
150 END
```

Since only two variables are involved in this program (A and X), we will list the line numbers in the order the computer encounters them and the corresponding values of the variables.

Line Number	*A*	*X*
100	1	
110	1	1
120	2	1
130	2	1
140	2	3
120	4	3
130	4	3
140	4	5
120	8	5
130	8	5
140	8	7 (Jumps out of loop)
150	(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.

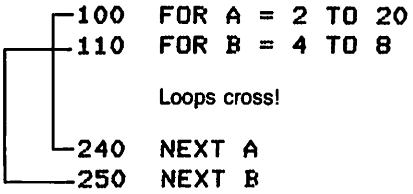
Quite often, more complicated loop structures are required in a program. The structure can be as involved as desired provided that the loops do not cross. The example below illustrates a segment of program with crossed loops.



```
100  FOR I = 0  TO 20 STEP 2
110  FOR A = 10 TO 2 STEP -1
120  FOR B = 1 TO 4
170  NEXT A
180  NEXT B
190  NEXT I
```

Outer loop OK; inner loops cross!

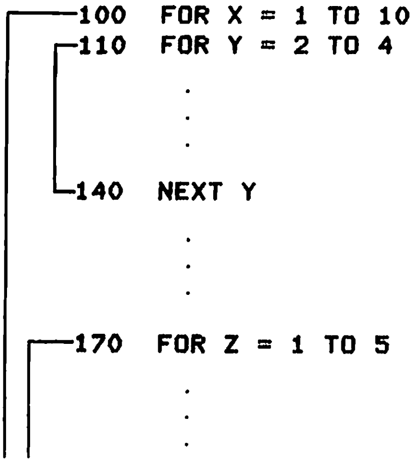
Another example with crossed loops is



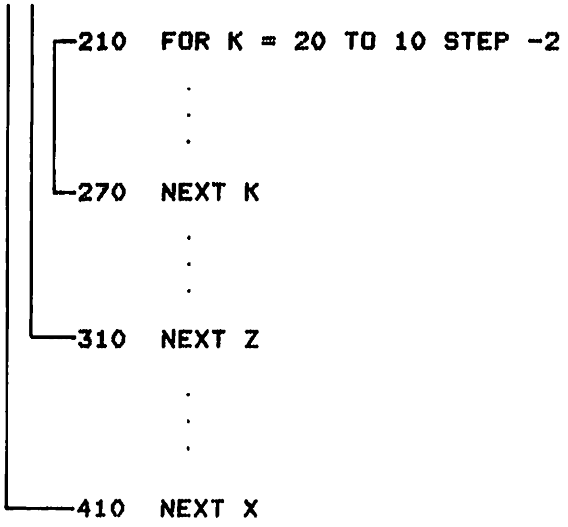
```
100  FOR A = 2 TO 20
110  FOR B = 4 TO 8
240  NEXT A
250  NEXT B
```

Loops cross!

The following example illustrates a complicated loop 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 though, that any combination 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!

Built-in Functions

One of the advantages of a modern digital computer is that sets of instructions can be preprogrammed to accomplish any desired task. Since there are many computing tasks that are routinely needed, the manufacturers have preprogrammed some of these in the form of functions. With these built-in functions in BASIC, the programmer can perform very complicated operations without difficulty. We will look at several of these functions and see exactly how they work.

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 use the first function, SQR(X), to see how the functions operate in general. First, X is called the "argument" of the function. If this definition bothers you, then think of X as "what the function works on." If we use SQR(X) in a program, we are instructing the computer to look up the value of X, and then to take the square root of the number. For example,

SQR(36) = 6
SQR(64) = 8
SQR(100) = 10
SQR(2) = 1.414213562

and so on. The only limitation is that we can't take the square root of a negative number. If the computer tried 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 like SQR(X+4*Y), it will look up the values of X and Y, carry out all the calculations indicated between the parentheses, and then take the square root. This characteristic is true for all the functions.

Any BASIC expression can be the argument of BASIC functions.

INT(X) takes the integer part of X. The term "integer" is just a high-class way to say "whole number." Thus, 2 is an integer while 23.475 is not. If we take the integer part of a positive number, we 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 is really happening when we take the integer part of a number is that we go to the first integer less than the number. Using this rule we see that

INT(−2) = −2
INT(−.93) = −1

and so on. Note carefully that the INT function does not round off a number. Often this can be somewhat confusing.

The integer part of a number is the first integer less than the number.

SGN(X) is a very interesting function. If X (the argument of the function) 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(−5.9031) = −1
SGN(−4) = −1

At this point it may not be clear to you why such a function could be useful. It turns out that the SGN function is very useful, however, and has many applications. For the time being, we will be content just to learn how the function works.

ABS(X) simply tells the computer to ignore the sign of X. In effect, it converts all values of X to positive numbers. So

ABS(4.5) = 4.5
ABS(−4.5) = 4.5
ABS(95.34) = 95.34
ABS(−95.34) = 95.34
ABS(0) = 0

The functions that operate on character strings are powerful and very useful. The first of these, SEG$(A$,M,N), causes the computer to select N characters from the character string A$ beginning with the Mth character. SEG stands for "segment" and, of course, means a segment of a character string. Any character string can be operated on by the SEG$ function.

To see how the function works, suppose B$ = "TELEVISION" in which case

```
SEG$(B$,1,3)="TEL"
SEG$(B$,5,1)="V"
SEG$(B$,3,8)="LEVISION"
```

In the example above, the quotation marks set off, but are not part of, the substring. As you can see, the ability to work with segments of a string opens new possibilities to us.

The next function we looked at in the discovery work was the LEN function. This is a very simple function to explain as it gives the number of characters in a string. Thus, if T$ = "AARDVARK" then LEN(T$) = 8. The quotation marks that set off the string are not counted as characters. Quite often if we are working with strings of unknown or variable length, the LEN function is a lifesaver!

The CHR$ function is used to generate characters from a master list used in the computer industry. In this list (called the ASCII character set) there are 127 characters that are referred to by number. Thus

```
100 LET A$=CHR$(N)
```

will assign the Nth character from the ASCII set to A$. A complete list of all the ASCII characters is in your owner's reference manual for your home computer.

In the discovery work, you probably were able to see the rough outlines of the ASCII character set. CHR$(65) is A and CHR$(90) is Z. The other upper-case letters are in between. The numeral 0 is character number 48; 9 would be character number 57. It would be wise for you to refer to the complete list of characters in the reference manual for your computer. For now, we will be content to see what the CHR$ function does and what its connection is with the ASCII character set.

We should discuss a function that was not covered in the computer work but which is closely related to the CHR$ function. This new idea is the ASC function. A typical statement might be

```
200 LET X=ASC(A$)
```

This causes the ASCII numeric value of the first character in A$ to be assigned to x. If A$ were HOTEL, then X would be assigned the value 72 which is the ASCII numeric value of H.

The next two functions we will go over are the VAL and STR$ functions. Before discussing these functions we should review an important fact. The number 25 can be either the number twenty-five or the character string 25. The computer handles numbers and strings differently. For example, the number 25 will require much more storage space in memory than the characters 2 and 5. For this reason, if memory is limited you can save space by storing all numbers in string form.

Numbers can be converted to strings with the STR$ function. Thus

```
120 LET A$=STR$(17)
```

converts the number 17 into the character string 17. Often there is a fair amount of confusion about this since the number and the string representation of the number look exactly the same when printed out. However, to the computer they are completely different quantities.

The VAL function converts from the string representation of a number to the numeric form. Therefore, the program segment

```
150 LET A$="23"
160 LET N=VAL(A$)
```

converts the string "23" to the number 23 and assigns it to the variable N. Of course if we tried to convert "28B" to the numeric representation, the computer would detect a character other than a digit (the B), signal an error, and stop.

The POS function is the last one we will discuss. The form of this function is always POS(A$,B$,X). This looks complicated but is not difficult to understand. If N = POS(A$,B$,X), N will be the character number in A$ where the string B$ is detected, where we begin the search at character position X. Thus if A$ = "AUTOMOBILE" and B$ = "O", then POS(A$,B$,1) = 4, and POS(A$,B$,5) = 6. If no match can be found, the value of the function is zero. Using the example above, POS(A$,B$,7) = 0.

There are other built-in functions in BASIC. However, most of these involve more mathematical knowledge than we can assume in this book. If you have had 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 the reference manual for your computer.

The built-in functions we have been discussing are used in BASIC statements. Examples of lines that utilize the functions might be

```
100 LET X=SQR(Y)
100 LET Z=3*INT(C)+ABS(D)
```

The built-in functions can be used within functions. An example of this is

```
100 LET Y=INT(SQR(X)+3*ABS(Z
))
```

7-4 PROGRAM EXAMPLES

The example programs that we will study have been chosen to show you how we can use automatic looping and the built-in functions to make programming easier.

Example 1 - Finding an Average

In the previous chapter, we used the problem of finding an average for one of the example programs. Let's return to the same problem, but use a different method. We want the program to produce printout similar to the following when RUN:

```
HOW MANY NUMBERS ? 5
ENTER THE NUMBERS,
ONE AT A TIME
? 12.5
? 10.8
? 11.3
? 14.1
? 12.8
THE AVERAGE IS 12.3
```

The first few lines should be easy for you to write by now.

```
100 PRINT "HOW MANY NUMBERS"
;
110 INPUT N
120 PRINT "ENTER THE NUMBERS
,"
130 PRINT "ONE AT A TIME"
```

Now we 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 we will set S (which will be used to sum the numbers) equal to 0.

```
140 LET S=0
```

The input of N numbers and the summing up of them is an ideal task for the FOR NEXT statements.

```
150 FOR I=1 TO N
160 INPUT X
170 LET S=S+X
180 NEXT I
```

Notice that we 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 higher line number after 180. When this happens, S will contain the sum of all the values of X that were typed in. Since we know that N is the number of numbers typed in, we can immediately compute the average.

```
190 LET A=S/N
```

The rest of the program follows without difficulty.

```
200 PRINT "THE AVERAGE IS";A
210 END
```

The complete program is

```
100 PRINT "HOW MANY NUMBERS"
;
110 INPUT N
120 PRINT "ENTER THE NUMBERS
,"
130 PRINT "ONE AT A TIME"
140 LET S=0
150 FOR I=1 TO N
160 INPUT X
170 LET S=S+X
180 NEXT I
190 LET A=S/N
200 PRINT "THE AVERAGE IS";A
210 END
```

Example 2 - Temperature Conversion Table

In one of the earlier programs we 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 we should print out the heading and the space before beginning the table itself.

```
100 PRINT "DEG. F","DEG. C"
110 PRINT
```

We 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, we need the END statement.

```
160 END
```

The whole program is given below.

```
100 PRINT "DEG. F","DEG. 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 - An Alphabet Problem

Suppose we want to write a program to print out the pattern shown below.

```
ABCDEF
 BCDEFG
  CDEFGH
    etc.
```

The pattern should continue until we have run through the complete alphabet. We will need a character-string function to do this. First, however, we will set up a character-string to define the alphabet.

```
100 LET A$="ABCDEFGHIJKLMNOP
QRSTUVWXYZ"
```

If you look carefully at the desired pattern, you will see that twenty-one lines will have to be printed out. Each line will have six characters. We will have to arrange to print each line one space further to the right than the previous one.

The commands necessary to do this are

```
110 FOR I=1 TO 21
120 PRINT TAB(I);
130 PRINT SEG$(A$,I,6)
140 NEXT I
```

The printing is positioned by the TAB function in line 120. Groups of six characters are picked out by the SEG$ function in line 130.

After adding the END statement, the complete program is

```
100 LET A$="ABCDEFGHIJKLMNOP
QRSTUVWXYZ"
110 FOR I=1 TO 21
120 PRINT TAB(I);
130 PRINT SEG$(A$,I,6)
140 NEXT I
150 END
```

This is a good program to experiment with. First, RUN the program to see that you do get the correct letter pattern. Then, you might want to change some of the parameters in the program and see what happens.

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 is decreased each year (due to use, wear, and tear), 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 five 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. Each year the value of the equipment is decreased by the amount of the depreciation. At the end of the last year's useful life, the equipment's value will be zero.

We want to write a BASIC program to generate depreciation schedules. First, we must know what the value of the equipment is, and its useful lifetime.

```
100 PRINT "ASSET VALUE ($)";
110 INPUT P
120 PRINT "ASSET LIFE (YEARS
)";
130 INPUT N
140 PRINT
150 PRINT "YEAR","DEPREC.","
VALUE"
160 PRINT
```

The sum-of-the-years-digits is computed easily.

```
150 LET S=0
160 FOR I=1 TO N
170 LET S=S+I
180 NEXT I
```

Now we compute the schedule and print it out. We will use the variable P1 to keep track of the current asset value.

```
190 LET P1=P
200 FOR I=1 TO N
210 LET F=(N+1-I)/S
220 LET D=P*F
230 LET P1=P1-D
270 PRINT "YEAR ";I
280 PRINT "DEPREC. IS";D
290 PRINT "VALUE IS ";P1
300 NEXT I
```

In line 210, 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 220, D is the depreciation. The only thing missing now is the END statement. The complete program is

```
100 PRINT "ASSET VALUE ($)";
110 INPUT P
120 PRINT "ASSET LIFE (YEARS
)";
130 INPUT N
140 PRINT
150 LET S=0
160 FOR I=1 TO N
170 LET S=S+I
180 NEXT I
190 LET P1=P
200 FOR I=1 TO N
210 LET F=(N+1-I)/S
220 LET D=P*F
230 LET P1=P1-D
270 PRINT "YEAR ";I
280 PRINT "DEPREC. IS";D
290 PRINT "VALUE IS";P1
300 NEXT I
310 END
```

Try out the program for different inputs. Of course, you can use this program to set up schedules to be used on your tax returns. Impress the Internal Revenue Service with your 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 as follows:

N	SQR(*N*)
2.0	1.414213562
2.1	1.449137675
2.2	1.483239697
etc.	
3.9	1.974841766
4.0	2.000000000

2. The problem is to evaluate the expression

$$X^2 + 3X - 4$$

for X = 0, 0.1, 0.2, ..., 1.9, 2.0. Print out the values of X and the corresponding values of the expression on the same line.

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 using FOR NEXT statements to read ten pairs of numbers from DATA statements. For each pair, print out the numbers and their sum.

5. Trace the following program. What will be output if it is RUN?

```
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
```

6. Explain what the following program does:

```
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
```

7. N! is read "N factorial" and means the product of all the whole numbers from 0 to N inclusive. For example

$$3! = (1)(2)(3) = 6$$

$$5! = (1)(2)(3)(4)(5) = 120$$

and so on. Write a program to call for the input of N. Then compute and print out N! If you try out this program on the computer, you may be surprised to find that values of N that don't seem large at all to you produce factorials too large to handle. The factorial of N is an extremely rapidly increasing function!

8. Write a BASIC program to call for N grades to be input. Compute and print out (1) the highest grade, (2) the lowest grade, and (3) the average of the grades.

9. 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
```

10. What will be output if the following program is RUN?

```
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
```

11. Suppose you decide to invest $1000 on the first day of each year for 10 years at an annual simple 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:

```
P2=(P1+I)(1+R/100)
```

In this formula, R is the annual interest rate in percent, I is the annual investment in dollars, P1 is the value of the investment at the beginning of each year, and P2 is the value of the investment at the end of the year. Thus, P2 becomes P1 for the next year. Write a BASIC program which will produce the following typical output when RUN:

```
ANNUAL INVESTMENT ? 1000
INTEREST RATE (%) ? 8
HOW MANY YEARS ? 20
AT THE END OF THE
LAST YEAR, THE VALUE
OF THE INVESTMENT
WILL BE 49422.9215
```

12. 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 190 contains a weekly record for one employee. The data are an 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.

13. 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% each toward the overall grade and the last grade is weighted 50%.

14. Write a program to input a character string and print out the number of times each vowel occurs in the string.

15. Write a program using FOR NEXT statements to print out all 127 members of the ASCII character set.

7-6 PRACTICE TEST

See how well you have learned the material in the chapter by taking this practice test. The answers are given at the end of the book.

1. What will be printed if the following program is RUN?

```
100 FOR Y=20 TO 1 STEP -2
110 PRINT Y,
120 NEXT Y
130 END
```

2. What will be printed if the following program is RUN?

```
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 a table similar to the following:

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 25,21,24,21,26,27,2
5,24,23,24
```

   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. Briefly explain the purpose of each of the following functions; ABS, SGN, INT, SQR, SEG$, and VAL.

CHAPTER

EIGHT

WORKING WITH COLLECTIONS OF INFORMATION

8-1 OBJECTIVES

In this chapter we will apply some of the ideas learned earlier to collections of information. New concepts will be introduced which will expand the capability of BASIC. The objectives are as follows.

Subscripted String Variables

The notion of a character-string variable can be extended to subscripted character-strings. This capability makes powerful non-numeric applications possible.

Subscripted Numeric Variables

Much more powerful programs dealing with numbers can be written using subscripted variables. Therefore we will see what subscripted numerical variables are and how to use them.

Program Applications

We will study BASIC programs that take advantage of both subscripted numeric variables, and subscripted character-string variables.

8-2 DISCOVERY ACTIVITIES

Since beginners often tend to have difficulty with this material, some introduction is needed before the computer work is started.

When working with groups of information we must be able to distinguish members of the group from one another. This is the reason for subscripts. Before getting into subscripts, howevever, we need to add an important word to our computer vocabulary. We could use the word "collection" to describe a group of pieces of information, but it turns out that another word is more commonly used. The word is "array." For our purposes array means a "collection of pieces of information." The pieces of information in the collection can be either numeric or character-string.

To see how this works, let's look at the array given below.

```
Y(1) =  9
Y(2) = 10
Y(3) =  7
Y(4) = 14
Y(5) = 12
Y(6) = 15
```

The name of this numeric array is Y. Its size is six, since there are six "elements" or "members" in the array. The numbers 9, 10, 7, 14, 12, and 15 are the elements in the array. The numbers printed in parentheses to the right of the Ys are called "subscripts." Each subscript points to one element in the array. Thus, Y(4) means the fourth number in the array, which is 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 but one which still uses the ideas introduced above.

```
Z$(1,1) = "DOG"     Z$(1,2) = "ON"      Z$(1,3) = "NOTE"
Z$(2,1) = "BUT"     Z$(2,2) = "RED"     Z$(2,3) = "NOT"
```

In this example there are six elements in the character-string array Z$. Since it is a character-string array, the elements of Z$ are words. 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$(2,1) is read as "Z string sub two one" and means the element of Z$ found in the second row and first column. In this case, Z$(2,1) is the word "BUT". Likewise, Z$(1,3) is "NOTE". and so on.

We can also have three-dimensional arrays on the TI Home Computer. The idea is an extension of one- and two-dimensional arrays. Now we have row, column, and "page" numbers. Thus, A(2,3,5) means the numerical element of array A at row 2, column 3, and page 5. Likewise, T$(1,4,2) identifies the character string in the collection T$ at row 1, column 4, page 2.

MATRIX and ARRAY both mean "collections" of information.

To sum up, we will work with three kinds of arrays. The one-dimensional array needs only a single number to locate an element in that array. The two-dimensional array needs two numbers (a row number and a column number) to locate an element. The three dimensional array needs three numbers (a row, column, and page number) to locate an element. The arrays can be either numeric or character-string.

The one-dimensional array is associated with the idea of a single-subscripted variable. Likewise, the double-subscripted variable is used in the two-dimensional array, and the triple-subscript is used in the three-dimensional array. With this brief introduction, you are ready for the computer work.

1. Bring up BASIC on your computer and enter the following program:

```
100 LET A$(1)="HOUSE"
110 LET A$(2)="BARN"
120 LET A$(3)="SHED"
130 LET A$(4)="STORE"
140 LET A$(5)="CABIN"
150 PRINT A$(4)
160 END
```

What do you think will be printed out if we RUN this program?

____

RUN the program and record what happened.

____

2. OK, change line 150 to read

```
150 PRINT A$(1),A$(3)
```

Now what do you think will happen?

____

RUN the program and write down what was printed out.

____

3. Change the comma in line 150 to "&" so that the line now reads

```
150 PRINT A$(1)&A$(3)
```

RUN the program and record what happened.

____

What does the & do when printing out character strings?

____

4. Clear the program from memory. Enter the following program:

```
100 FOR I=1 TO 5
110 READ B$(I)
120 NEXT I
130 DATA "RED","WHITE","BLUE
"
140 DATA "GREEN","BROWN"
150 PRINT B$(3)
160 END
```

Study the program for a few moments. What do you think will be printed out if the program is RUN?

RUN the program and see if you were right.

5. Delete lines 150 and 160 from the program. Enter the following additions:

```
150 FOR I=1 TO 5
160 PRINT B$(I),
170 NEXT I
180 END
```

Now what do you think will happen?

RUN the program and record what the computer did.

6. Change line 150 to read

```
150 FOR I=5 TO 1 STEP -2
```

RUN the program and write down the output.

7. Now let's extend the subject a bit. Clear out the program in memory and enter the following:

```
100 LET C$(1,1)="WHITE"
110 LET C$(1,2)="BLACK"
120 LET C$(1,3)="BROWN"
130 LET C$(2,1)="CAR"
140 LET C$(2,2)="BIKE"
150 LET C$(2,3)="PLANE"
160 FOR I=1 TO 2
170 PRINT C$(I,2)
180 NEXT I
190 END
```

This program is more complicated but you should be able to figure out what it does. RUN the program and record what took place.

8. OK, change line 170 to read

```
170 PRINT C$(I,3)
```

What will be output now?

RUN the program and record what happened.

9. Change lines 160, 170, and 180 to read

```
160 FOR J=1 TO 3
170 PRINT C$(1,J)
180 NEXT J
```

What will the program do now?

RUN the program and record the output.

10. Change line 170 to read

```
170 PRINT C$(2,J)
```

Now what will be output?

RUN the program and write down what took place.

11. So far we have been working with collections of words. We can work equally well with collections of numbers. Clear the program from memory and enter the following:

```
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 happen if we RUN this program?

RUN the program and record what happened.

12. Now modify the program to print out the fourth value of X. RUN the program. Did it work?

13. OK, change line 150 as follows:

```
150 PRINT X(3)+X(4)
```

Display the program and study it briefly. What do you think will happen if we RUN the program?

RUN the program and see if you were right. Record below what actually was printed out.

14. 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 by this program?

See if you were right. Record below what happened when the program was RUN.

15. Modify this program to print out only the first three values of the array X. Record below what happened when you tried this.

16. Again, modify the program, but this time so that the first value of the array, and then every other value, will be printed out. Record what happened below.

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

```
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 what was printed out.

18. 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.

____

19. 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 we RUN this program?

____

RUN the program and record what was printed out.

____

Explain in your own words what is taking place in the program.

____

20. 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 write down what was printed out.

Again try to explain in your own words what is happening.

21. Now type

```
162 FOR I=1 TO 2
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 with what was going on in steps 19 and 20. What does this program do?

RUN the program and record what was typed out.

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

```
100 DIM X(12),Y(12)
110 FOR I=1 TO 12
120 READ X(I),Y(I)
```

```
130 NEXT I
140 PRINT X(1)+Y(4)
150 DATA 2,1
151 DATA -1,3
152 DATA 5,6
153 DATA 2,4
154 DATA 3,1
155 DATA 8,4
156 DATA 5,1
157 DATA 3,4
158 DATA 6,2
159 DATA 1,1
160 DATA 7,7
161 DATA 5,3
170 END
```

Display the program and check to see that you have entered it correctly. Study the program carefully. If we RUN the program, what will be typed out?

____

RUN the program and see whether or not you were right. Record below what was typed out.

____

23. Type

```
100
```

Now display the program. What has happened?

____

RUN the program and record what happened.

____

Does the DIM statement that was originally present in the program appear to be necessary?

24. Type

```
100 DIM X(9),Y(9)
110 FOR I=1 TO 9
```

Display the program. What will happen now if we RUN the program?

Try it and see if you were correct.

25. Type

```
100
```

Doing this deleted line 100 from the program. Will the program work now that the DIM statement has been taken out?

Try it and record the output.

Compare the results of step 23 with those of step 25. Sometimes the DIM statement must be present and other times it need not be. We will return to this question later.

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

```
100 DIM A(4,3)
110 FOR I=1 TO 4
120 FOR J=1 TO 3
130 READ A(I,J)
140 NEXT J
150 NEXT I
160 FOR I=1 TO 4
170 FOR J=1 TO 3
180 PRINT A(I,J);
190 NEXT J
200 PRINT
210 PRINT
220 NEXT I
230 DATA 1,3,1
240 DATA 4,2,5
250 DATA 1,4,2
260 DATA 3,2,5
270 END
```

Make sure that you have entered the program correctly, then take a few minutes to study it. Can you see what will be printed out if we execute the program?

____

RUN the program and record the output.

____

Compare what was printed out with the numbers in the DATA statements in the program.

27. Now that we have looked at one- and two-dimensional arrays, let's look briefly at one with three dimensions. Clear out the memory and enter the following program.

```
100 DIM A(2,3,2)
110 FOR P=1 TO 2
120 FOR R=1 TO 2
130 FOR C=1 TO 3
140 READ A(R,C,P)
```

```
150 NEXT C
160 NEXT R
170 NEXT P
180 REM PAGE 1
190 DATA 5,3,6
200 DATA 2,1,2
210 REM PAGE 2
220 DATA 3,4,3
230 DATA 1,5,1
240 PRINT A(1,1,1)+A(2,1,2)
250 END
```

This program looks complicated, but by now you should be able to see what it does. In particular, focus on the concept of row, column, and page indicated by the subscripted variable A(R,C,P) in line 140. If you RUN the program, what do you think will be printed out?

RUN the program and see what took place.

---

28. Now make the following changes.

```
250 LET S=0
260 LET P=1
270 FOR R=1 TO 2
280 FOR C=1 TO 3
290 LET S=S+A(R,C,P)
300 NEXT C
310 NEXT R
320 PRINT S
330 END
```

What will happen now if we RUN the program?

Try it and record what happened.

29. Now change line 260 to set P equal to 2. If we RUN the program now, what will happen?

RUN the program and write down what was printed out.

30. For the next few steps you will need a tape cassette connected to the computer. If you don't have one go on to the discussion material.

31. Clear out the memory and enter the following program.

```
100 OPEN #1:"CS1",OUTPUT,FIX
ED 64
110 FOR I=1 TO 3
120 READ A$,N
130 PRINT #1:A$
140 PRINT #1:N
150 NEXT I
160 CLOSE #1
170 DATA "HERB",215
180 DATA "MARY",142
190 DATA "JACK",193
200 END
```

This program has several new features that you haven't seen before-namely the OPEN and CLOSE statements in lines 100 and 160 as well as the different form of the PRINT statements in lines 130 and 140. Make sure the tape cassette unit is properly connected to the computer and has a blank tape inserted. RUN the program. What happened?

32. All right, follow the instructions on the screen and then press the ENTER key. What did the computer do?

____

Again follow the instructions and press the ENTER key. What happened on the cassette unit?

____

33. At this point you should see the cassette tape turning. As you probably suspected, data is being written on the cassette tape. What happened when the tape stopped turning?

____

34. Follow the instructions displayed on the screen and then remove the tape from the cassette unit. Now let's reverse the process and read the data back into the computer from the tape, then print it out. Clear the memory and enter the program below.

```
100 OPEN #1:"CS1",INPUT,FIXE
D 64
110 FOR I=1 TO 3
120 INPUT #1:A$
130 INPUT #1:N
140 PRINT A$,N
150 NEXT I
160 CLOSE #1
170 END
```

Study the program for a moment and note the similarities and differences when compared to the previous program.

35. Now RUN the program and follow the instructions at each step as they are displayed on the screen. What finally happened?

____

36. This concludes the discovery activities for now. Turn your computer off and go on to the discussion material.

8-3 DISCUSSION

It is natural to be a bit confused at this point about arrays, both numeric and string. Therefore it is important that you pay particular attention to the discussion material to clear up any questions that might have arisen in the discovery activities.

Subscripted Variables

The need for subscripted variables becomes obvious when we must handle large collections of information. It makes no difference whether the information is string or numeric. If, for example, we were writing a program that involved only four numbers, we would have no difficulty naming them. We might call the numbers X, Y, U, and V. But suppose we needed to work with 100 numbers? For this, and other reasons, it is often very useful to have subscripted variables. Fortunately BASIC has provisions for subscripts that can be applied to either string or numeric variables that are ready and waiting for our use.

Consider the following set of numeric information:

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 with the single name Y. Thus, Y is a "collection of numbers" or an "array"—both of which mean roughly the same thing for our purposes. To locate a number in the array, we must have the array name (in this case Y) and the position within the array. Here is where the I column is used. Thus Y(3) which is read as "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. Generally we can speak of Y(I), which we read as "Y sub I" and which denotes any element of the array depending on the value of I. If I were 8, then Y(I) would be 3 in our example. 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 numeric array.

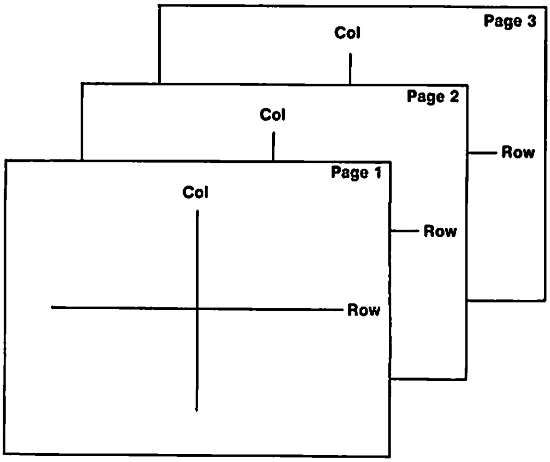
$Y_{I,J}$	1	2	3	4
1	3	−1	10	8
2	2	4	5	6
3	1	−2	9	3

Now we need two numbers to locate an element in the array. Given a row number and a column number, we can find any element of the array we desire. 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. In general, we denote an element in the two-dimensional array as 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:

$$Y_{3,2} = -2$$
$$Y_{1,4} = 8$$
$$Y_{3,3} = 9$$
$$Y_{2,1} = 2$$

By extending the previous ideas to one more dimension we have a three-dimensional array. Now the third subscript indicates the "page" number. It should be pointed out that the use of the term "page" is not widespread but it is a convenient way to think about the third subscript. The diagram below shows how the subscripts are organized.



Thus, to locate an element, we must specify which page to look at, then the row and column number on that page. Using this notation X(5,3,2) means the element of the numeric array X located at row 5, column 3, on page 2.

In BASIC, subscripts are enclosed in parentheses following the array name. Thus, Y(2) means "Y sub two" and does not indicate Y multiplied by (2). B$(5,8) means "B$ sub five eight." 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 positive numbers or zero. However, there is an important point to be remembered. Suppose we want to look up 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. Accordingly, the computer will round the number to the nearest integer and, in this case, X(A+B) works out to be X(4), the fourth element in the array X.

Everything that has been said about numeric arrays applies to character-string arrays. By this time you should be familiar enough with the concept that we do not need the word "character" any more. It should be clear that "string array" refers to a collection of characters. So, from this point on we will use the terms "string array" and "string variable" rather than "character-string array" and "character-string variable."

An example of a one-dimensional string array is

```
X$(1) = "SON"
X$(2) = "DAUGHTER"
X$(3) = "MOTHER"
X$(4) = "FATHER"
X$(5) = "UNCLE"
X$(6) = "AUNT"
```

The words comprise the elements of the array. The numbers 1 through 6 are the subscripts that locate a particular word in the array. The computer handles subscripts in string arrays in the same manner as it handles numeric arrays.

An example of a two dimensional string array is

```
A$(1,1) = "AA"        A$(1,2) = "AB"
A$(2,1) = "BA"        A$(2,2) = "BB"
A$(3,1) = "CA"        A$(3,2) = "CB"
```

Here the elements are pairs of characters to illustrate that string array elements are just collections of characters. They need not be words.

One final comment about string variables. String variables can be read from DATA statements in the same fashion as numeric variables. If strings are to be used in DATA statements, be safe and enclose them in quotation marks. If a READ statement contains both numeric and string variables, you must be careful that the information in the DATA statements matches the type of information being asked for. If, for example, the computer is trying to read a string variable, and the next information in the DATA statements is numeric, the computer will halt and signal an error.

Saving Space for Arrays

Before discussing the DIMension statement, we must look closer at the notion of subscripted arrays. In particular, the question comes up "What is the lowest subscript possible?" In the discovery work the issue wasn't raised and you probably tacitly assumed that the lowest subscript possible was one.

It turns out that some computers use one for the lowest subscript and others use zero. The TI Home Computer will permit either one! This is done through the use of the OPTION statement. When you turn the computer on, the lowest (or base) subscipt is set to zero. It was zero all through the discovery work but you were deliberately steered from situations where the fact would be noticed.

To change the base to 1 you insert the statement OPTION BASE 1 in a program. This should be done at the beginning of the program before any references are made to arrays. There can be only one OPTION statement in a program. To avoid confusion, it is probably a wise practice to include either an OPTION BASE 1 or OPTION BASE 0 statement in all programs using arrays. This way there is no question about the base of the subscripts in the arrays used in the program.

There are instances in which the zero base for subscripts is valuable. However, if there is no specific need for the zero base, it is wise to declare option base 1 as less memory is then required to hold the arrays.

Save space for arrays with a DIM statement.

The computer must know how big an array is for two reasons. First, there is a question of how much space to save in memory to hold the array. Next, the computer must know the size of the array in order to carry out arithmetic operations properly. Actually, for small arrays, BASIC saves space automatically. If a one-dimensional array is used in a program, BASIC automatically sets up space for ten elements (option base 1) or eleven elements (option base 0) if there is no DIM statement. If a two-dimensional array is used, BASIC will save enough space in memory for either a ten by ten or an eleven by eleven array if no DIM statement is in the program depending on the option base. The same thing happens for a three-dimensional array. Whether space is saved for a ten by ten by ten, or an eleven by eleven by eleven array depends on the option base in effect.

It probably isn't wise to use this automatic space saving feature of BASIC. We will emphasize the routine use of DIMension statements in all programs regardless of the size of the arrays. Troubleshooting a program that uses arrays is very difficult if no DIM statement is present.

An example of a DIM (for "DIMension") statement is

```
100 DIM B(5,20),Y(3,4,6),Z(
34),X$(3,6)
```

Four arrays are dimensioned in line 100. B is a two-dimensional numeric array having five rows and twenty columns. Y is a three-dimensional numeric array with three rows, four columns, and six pages. Likewise, Z is numeric, one-dimensional, and has thirty-four elements. Finally, X$ is a string array with three rows and six columns. It's a good practice to place the DIM and OPTION statements at the beginning of the program. This way it is easy to glance at the beginning of the program to see the sizes of the arrays that will be used. At any rate, the DIM and OPTION statements must be before any other statements that refer to arrays. As indicated above, it is also a good practice to use a DIM statement in all programs, whether or not BASIC demands it.

Subscripted Variables and FOR NEXT Loops

Since subscripts involve collections of data and operations with collections of data almost always involve repetition, it seems reasonable that we should employ FOR NEXT statements to handle arrays. As an example, the following program segment will set up a six by four array, then load 5s into all the elements.

```
100 DIM A(6,4)
110 OPTION BASE 1
120 FOR R=1 TO 6
130 FOR C=1 TO 4
140 LET A(R,C)=5
150 NEXT C
160 NEXT R
```

If we trace this program segment, the details of the process become clear. When line 140 in the program is reached 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-,two-, or three-dimensional arrays can be handled in this fashion using subscripts. Loops and arrays provide a new measure of muscle for the computer and begins to reveal the power it possesses.

Writing Information to Files

In the discovery work you were lead through an example in which data (strings and numbers) were written on a tape in the cassette unit. Any serious computer work usually involves setting up and maintaining data files. The ability to record such data on tapes where it can be retrieved at some subsequent time is fundamental to almost any type of information management. Now we will look carefully at the process by which data is recorded on tape.

In a program which is to record data we must first open a communication path to the cassette unit. This is done with the OPEN statement, an example of which is shown below.

```
100 OPEN #1:"CS1",OUTPUT,FIX
ED 64
```

In this statement, the #1 refers to the communication channel number over which data will be sent to the tape cassette. This number can be any integer between 1 and 255. The only reason for using (or needing) more than one channel number would be if the computer were communicating with more than one device at the same time. Since we will limit our activities to working with a single cassette unit, we will always select channel #1.

The characters in quotes which follow the channel number name the file to which data will be written. In this case, CS1 indicates that cassette number one will be used. Next, the type of file is specified. Since we want to record data or information on a tape, we specify the file to be OUTPUT. Finally, FIXED 64 indicates that information will be recorded in fixed length blocks (or records) sixty-four characters long. It is important to understand that if, for example we wanted to record a word with fifteen characters in it, the computer will still record sixty-four characters on the tape with blanks filling out the unused portion of the fixed length block.

The OPEN statement sets up everything needed to send data from the comuter to the cassette unit. As long as you are dealing with a single output device (the cassette unit), and limit the output to fixed-length records of sixty-four characters, you can always use the OPEN statement above.

To send information to the tape, we use the PRINT statement. An example is

```
200 PRINT #1:X$
```

Notice that this PRINT statement differs from the ones used previously in that we must specify the channel over which the information is to be printed. The channel number should agree with the one used in the OPEN statement. We will always use channel number one. In this example PRINT statement, the string X$ is to be printed over channel number one. Of course, we could print a number over the channel if desired. It makes no difference where the information comes from or how it is generated. It is sent to the tape with the PRINT command. A final comment is that it will simplify matters if only a single quantity (string or number) is recorded in each PRINT statement.

When all the information has been sent to the tape unit we must sever the communications channel. This is done with the following statement:

```
300 CLOSE #1
```

As you might expect we have to specify which channel we are closing. Since we have agreed that only a single channel will be used at a time, this can always be channel #1.

All programs to record data to files on tape will have the following form:

```
100 OPEN #1:"CS1",OUTPUT,FIX
ED 64
                 .
                 .
                 .
(generate material to be recorded)
                 .
                 .
                 .
500 PRINT #1: (a string or number)
                 .
                 .
                 .
800 CLOSE #1
900 END
```

The program should loop past the PRINT statement until all the data has been recorded. Before recording any data, it is a very good idea to first record the number of pieces of data that will subsequently be recorded. This way when the material is read back into the computer, the program can read the first number on the tape and know how many pieces of data are to be read.

When a program is RUN, if an OPEN statement is detected, the following message is displayed.

```
* REWIND TAPE CASSETTE CS1
  THEN PRESS ENTER
```

Next, if the OPEN statement sets up an OUTPUT file (the case here) the following message is displayed.

```
* PRESS CASSETTE RECORD CS1
  THEN PRESS ENTER
```

Of course, after both these messages, you press the ENTER key to signal the computer you have carried out the requests. At this point, data will be written on the tape.

When the CLOSE statement is reached, the computer prints out

```
* PRESS CASSETTE STOP CS1
  THEN PRESS ENTER
```

Once you have carried out these instructions, the recording process is complete.

It is not difficult to record data files on cassette tapes. Remember to set up communications with the OPEN statement (OUTPUT type) and sever communications with the CLOSE statement. Material is sent to the file with the PRINT statement. In all three of these statements, use file channel #1. At RUN time follow the instructions as they are displayed.

Reading Information From Files

Having written (or recorded) information in data files on cassette tape, we now must be able to write programs to input information back into the computer from these files. All programs to do this will have the same general form.

```
100 OPEN #1:"CS1",INPUT,FIX
ED 64
                    .
                    .
                    .
300 INPUT #1: (string or number)
                    .
                    .
                    .
500 CLOSE #1
600 END
```

This structure is the same as the program to record data. The OPEN statement has the same purpose as before except now it is an INPUT file. We loop through the INPUT statement as many times as needed to input the data from the tape. The CLOSE statement severs communications with the cassette as before. We will always use file channel #1 in these three statements.

Several comments should be made here. First, you should generally make sure that the first piece of information recorded in a data file gives the number of pieces of information to be recorded subsequently. Then by reading this "quantity information" first, a program to input data from a tape can be structured to ask for the proper number of pieces of information. The second comment is that input of information must agree as to type. If the program calls for input of a string, the next information on the tape must be a string. Likewise if input of a numeric variable is called for, the next item on the tape should be a number. Finally, let's agree to ask for input of only a single quantity (string or number) in a single INPUT statement.

When a program to input data from a tape is RUN, the only difference in the prompts is that you will be instructed to

```
* PRESS CASSETTE PLAY CS1
  THEN PRESS ENTER
```

When writing information to data files, or reading information from data files, write very modest programs initially. Once you have a feel for the process and understand clearly what is taking place, more ambitious data management programs will be in order.

8-4 PROGRAM EXAMPLES

The use of subscripted variables and data files permits many interesting problems to be handled easily in BASIC. We will look at several programs to illustrate how to tackle such problems.

Example 1 - Examination Grades

To illustrate the concept of a one-dimensional array, let's take an example that is near and dear to the hearts of most people—a set of examination grades. Suppose that we have the following results on an examination given to a class of fifteen students.

	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 to allow the class grades above to be typed in. The format should appear as follows:

```
HOW MANY STUDENTS ? 15
STUDENT          GRADE

1                ? 67
2                ? 82
3                ? 94
```

```
4                           ? 75
5                           ? 48
6                           ? 64
7                           ? 89
8                           ? 91
9                           ? 74
10                          ? 71
11                          ? 65
12                          ? 83
13                          ? 72
14                          ? 69
15                          ? 72
```

The program should compute the class average, the highest grade, and the lowest grade, and print this information out as follows:

```
CLASS AVERAGE IS 74.4
HIGHEST GRADE IS 94
LOWEST GRADE IS 48
```

As in past exercises, let's take this by steps. First, since we are going to store the student grades in subscripted form, we must include DIM and OPTION statements to save space for the array.

```
100 DIM G(50)
110 OPTION BASE 1
```

We are using the variable G to store grades and can insert up to fifty grades. Next we have a message, an input, and a space.

```
120 PRINT "HOW MANY STUDENTS
";
130 INPUT N
140 PRINT
```

Now we are ready to input the grades. First the heading for the table must be generated.

```
150 PRINT "STUDENT","GRADE"
160 PRINT
```

A loop using FOR NEXT statements is ideal to control the input of grades.

```
170 FOR I=1 TO N
180 PRINT I,
190 INPUT G(I)
200 NEXT I
```

The student number is printed out in line 180. In line 190, the student number (I) is used as a subscript for the grade. This 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 we compute 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 we will set both H and L equal to the first grade in the list which is G(1). We know that the same grade can't be the highest and lowest at the same time. Thus, we will 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:

```
350 NEXT I
360 PRINT "HIGHEST GRADE IS"
;H
370 PRINT "LOWEST GRADE IS";
L
380 END
100 DIM G(50)
110 OPTION BASE 1
120 PRINT "HOW MANY STUDENTS
";
130 INPUT N
140 PRINT
150 PRINT "STUDENT","GRADE"
160 PRINT
170 FOR I=1 TO N
180 PRINT I,
190 INPUT G(I)
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)
```

RUN this program on your computer using the DATA at the beginning of the discussion. 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

We can easily extend the ideas in Example 1 to a two-dimensional array. Now, suppose we have a class with 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	63	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

We will use FOR NEXT commands to READ the data from DATA statements. The computer is to compute 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 must start with a DIM statement although the DATA statements can go anywhere in the program.

```
100 DIM G(5,10)
110 OPTION BASE 1
```

This 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. The DATA statement can come next.

```
120 DATA 92,71,81,52,75,97,9
9,63,41,75
130 DATA 85,63,79,49,71,91,9
3,58,52,71
140 DATA 89,74,80,61,79,88,9
7,55,51,73
150 DATA 96,68,84,58,80,93,9
5,61,47,70
160 DATA 82,72,82,63,73,92,9
3,68,56,74
```

Now we must read the data into the program.

```
170 FOR R=1 TO 5
180 FOR C=1 TO 10
190 READ G(R,C)
200 NEXT C
210 NEXT R
```

This causes the numbers to be read into the array G by rows. Thus, the data in line 120 become row 1 of the array G, the data in line 130 become row 2 of the array, and so forth. Before doing anything else, we must print out the required headings.

```
220 PRINT "STUDENT","COURSE
AVE."
230 PRINT
```

Now we can compute the course average for each student.

```
240 FOR C=1 TO 10
```

Line 240 opens a loop that will look at each column in the array. For each value of C, we must compute 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, the C loop must be closed.

```
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 we have the END statement.

```
410 END
```

The complete program follows:

```
100 DIM G(5,10)
110 OPTION BASE 1
120 DATA 92,71,81,52,75,97,9
9,63,41,75
130 DATA 85,63,79,49,71,91,9
3,58,52,71
140 DATA 89,74,80,61,79,88,9
7,55,51,73
150 DATA 96,68,84,58,80,93,9
5,61,47,70
160 DATA 82,72,82,63,73,92,9
3,68,56,74
```

```
170 FOR R=1 TO 5
180 FOR C=1 TO 10
190 READ G(R,C)
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
```

Example 3 - Alphabetic Sort

As an example of how a string array might be used, let's design a program to call for the input of a list of words, sort the list into alphabetic order, and then print out the sorted list.

First, we will agree that no more than twenty words will be in the list. Of course, this could be any value we desire, but twenty seems like a good number. If we use A$ to name the string array, we can write the dimension and option statements.

```
100 DIM A$(20)
110 OPTION BASE 1
```

Next, let's call for the number of words in a specific list. Under the ground rules, this can be anything up to twenty. Then, we must input the words.

```
120 PRINT "HOW MANY WORDS";
130 INPUT N
140 FOR I=1 TO N
150 INPUT A$(I)
160 NEXT I
```

Now that the list of words is input, it can be sorted. The program segment below does this.

```
170 FOR I=1 TO N-1
180 IF A$(I+1)>=A$(I) THEN 2
30
190 LET B$=A$(I+1)
200 LET A$(I+1)=A$(I)
210 LET A$(I)=B$
220 GOTO 170
230 NEXT I
```

Study this program segment until you see how it works. If the condition in line 180 is true, the two words being compared are in alphabetical order and the comparison shifts up one place in the list. If not, the set of statements in lines 190 through 210 interchanges the two words. Then from line 220, the whole comparison starts again. This process keeps up until the assertion in line 180 is true for the whole list, at which time the list is in alphabetic order.

The sorted list is now output.

```
240 PRINT
250 FOR I=1 TO N
260 PRINT A$(I)
270 NEXT I
280 END
```

The complete program is

```
100 DIM A$(20)
110 OPTION BASE 1
120 PRINT "HOW MANY WORDS";
130 INPUT N
140 FOR I=1 TO N
150 INPUT A$(I)
160 NEXT I
170 FOR I=1 TO N-1
180 IF A$(I+1)>=A$(I) THEN 2
30
190 LET B$=A$(I+1)
200 LET A$(I+1)=A$(I)
210 LET A$(I)=B$
220 GOTO 170
230 NEXT I
240 PRINT
```

```
250 FOR I=1 TO N
260 PRINT A$(I)
270 NEXT I
280 END
```

Try this program out with a list of words of your choosing. Verify that the program does sort the list of words that you input into alphabetic order.

Example 4 - Business Records

As a final example suppose a small business needs a phone directory keyed to a customer identification number. The information is to be stored in the computer in a two dimensional string array A$. The information for each customer will be stored in a row as follows: column 1 - customer ID number, column 2 - last name, column 3 - first name, column 4 - telephone area code, and column 5 - phone number. We will store N (the number of customers in the directory) in element A$(0,0). All information will be stored in string form. Thus, numbers will have to be converted to strings before storage, and converted back to numbers when read from storage.

First, let's set up the array for a maximum number of customers. Since this example is intended to demonstrate the ideas involved, we will limit the maximum number of customers to twenty. Of course in a real world situation, this would be much bigger. At any rate, our problem is to write a program to call for the input information about N customers, load the information into the array A$, then record the information on a cassette tape. The program starts easily.

```
100 DIM A$(20,5)
110 OPTION BASE 0
```

Next we ask for the number of customers to be input.

```
120 PRINT "HOW MANY NAMES";
130 INPUT N
140 LET A$(0,0)=STR$(N)
```

We can use N in the program but have also converted it to a string to be stored in the array.

The input of the data and storage in the array follows without difficulty.

```
150 FOR I=1 TO N
160 LET J=1
```

```
170 INPUT "ID=":B$
180 LET A$(I,J)=B$
190 LET J=J+1
200 INPUT "LAST NAME=":B$
210 LET A$(I,J)=B$
220 LET J=J+1
230 INPUT "FIRST NAME=":B$
240 LET A$(I,J)=B$
250 LET J=J+1
260 INPUT "AREA CODE=":B$
270 LET A$(I,J)=B$
280 LET J=J+1
290 INPUT "PHONE #=":B$
300 LET A$(I,J)=B$
310 NEXT I
```

Now that the information is loaded, we will output it to the tape.

```
320 OPEN #1:"CS1",OUTPUT,FIX
ED 64
330 PRINT #1:A$(0,0)
340 FOR R=1 TO N
350 FOR C=1 TO 5
360 PRINT #1:A$(R,C)
370 NEXT C
380 NEXT R
390 CLOSE #1
400 END
```

The complete program follows.

```
100 DIM A$(20,5)
110 OPTION BASE 0
120 PRINT "HOW MANY NAMES";
130 INPUT N
140 LET A$(0,0)=STR$(N)
150 FOR I=1 TO N
160 LET J=1
170 INPUT "ID=":B$
180 LET A$(I,J)=B$
190 LET J=J+1
200 INPUT "LAST NAME=":B$
210 LET A$(I,J)=B$
220 LET J=J+1
230 INPUT "FIRST NAME=":B$
240 LET A$(I,J)=B$
```

```
250 LET J=J+1
260 INPUT "AREA CODE=":B$
270 LET A$(I,J)=B$
280 LET J=J+1
290 INPUT "PHONE #=":B$
300 LET A$(I,J)=B$
310 NEXT I
320 OPEN #1:"CS1",OUTPUT,FIX
ED 64
330 PRINT #1:A$(0,0)
340 FOR R=1 TO N
350 FOR C=1 TO 5
360 PRINT #1:A$(R,C)
370 NEXT C
380 NEXT R
390 CLOSE #1
400 END
```

You might try this program out with names and numbers of your choice. Once the data is recorded, we would like to reload the array A$ from the tape. The program below does this.

```
100 DIM A$(20,5)
110 OPTION BASE 0
120 OPEN #1:"CS1",INPUT,FIX
ED 64
130 INPUT #1:M$
140 LET A$(0,0)=M$
150 LET N=VAL(M$)
160 FOR R=1 TO N
170 FOR C=1 TO 5
180 INPUT #1:B$
190 LET A$(R,C)=B$
200 NEXT C
210 NEXT R
220 CLOSE #1
230 END
```

Of course, once the array A$ is reloaded, it could be modified, or sorted as desired, then recorded again on tape. However, the purpose of this example is to illustrate how an array can be loaded, stored on tape, and then recalled from tape.

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
```

which will read the size of a one-dimensional numeric array from the first DATA statement, then read the elements of the array from the second DATA statement, loading them into an array X. Then print out the array.

2. Write a BASIC program to read twenty-five numbers from DATA statements into a one-dimensional array named A. 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.

3. What will be output if the following program is RUN?

```
100 DIM Y(6)
110 OPTION BASE 1
120 FOR I=1 TO 6
130 READ Y(I)
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
```

4. What will be output if the following program is RUN?

```
100 DIM A(10)
110 OPTION BASE 1
120 FOR I=1 TO 10
130 READ A(I)
140 NEXT I
```

```
150 LET X=A(1)
160 FOR I=1 TO 9
170 LET A(I)=A(I+1)
180 NEXT I
190 LET A(10)=X
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
```

5. 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, sort the array into descending order, and finally print out the sorted array. (Hint: Look at the sort in Example 3.)

6. 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 numbers of 1s, number of 2s, etc., in the data and then print this out. (Hint: Use the data as they are read in as a subscript to increment an element of an array used to count the numbers.)

7. What will be printed out if the following program is RUN?

```
100 DIM Z(6,6)
110 OPTION BASE 1
120 FOR R=1 TO 6
130 FOR C=1 TO 6
140 LET Z(R,C)=0
150 NEXT C
160 NEXT R
170 FOR R=1 TO 5 STEP 2
180 FOR C=R TO 6
190 LET Z(R,C)=1
200 NEXT C
210 NEXT R
220 FOR R=1 TO 6
230 FOR C=1 TO 6
240 PRINT Z(R,C);
250 NEXT C
260 PRINT
270 PRINT
280 NEXT R
290 END
```

8. If the program below is executed, what will the computer print out?

```
100 DIM A(5,5)
110 OPTION BASE 1
120 FOR R=1 TO 5
130 FOR C=1 TO 5
140 LET A(R,C)=2
150 NEXT C
160 NEXT R
170 FOR C=5 TO 1 STEP -1
180 FOR R=1 TO C
190 LET A(R,C)=3
200 NEXT R
210 NEXT C
220 FOR R=1 TO 5
230 FOR C=1 TO 5
240 PRINT A(R,C);
250 NEXT C
260 PRINT
270 PRINT
280 NEXT R
290 END
```

9. Write a program to read the following array from DATA statements, then print out the array.

$$\begin{bmatrix} 2 & 1 & 0 & 5 & 1 \\ 3 & 2 & 1 & 3 & 1 \end{bmatrix}$$

10. Write a program to read the following array from DATA statements, then print out the array.

$$\begin{bmatrix} 5 & 3 \\ 2 & 0 \\ -1 & 1 \\ 4 & 2 \\ 2 & 6 \end{bmatrix}$$

11. Write a BASIC program that will call for the input of an M by N array. Then compute and print out the sum of the elements in each row and the product of the elements in each column.

12. Write a BASIC program that will read two arrays from DATA statements. Both the arrays are two by three. Then compute another 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.

13. The data below represent sales totals made by salespersons over a 1-week period.

		Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**
Salesperson	**1**	48	40	73	120	100	90
	2	75	130	90	140	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, and (c) the total weekly sales.

14. Write a BASIC program to input a list of N names and N grades into two different one-dimensional arrays. Assume that N will not be greater than twenty. Sort the arrays so that the names are in alphabetical order, and the grades are matched correctly with the names. Try out the program on data of your choice.

15. Repeat problem 14 except sort the grades so that they are listed in descending order with the names matched correctly with the grades.

16. Write a program to record ten numbers to be input from the keyboard on a data tape.

17. Write a program to input ten first names from a data tape. Sort the list into alphabetic order and then print it out.

8-6 PRACTICE TEST

Check yourself with the following practice test. The answers are given at the end of the book.

1. What is the purpose of the DIM and OPTION statements?

2. We have an array named X. What variable name does BASIC use to locate the element in row 3, column 4?

3. What will happen if the following program is RUN?

```
100 DIM A$(4),B(4)
110 OPTION BASE 1
120 FOR I=1 TO 4
130 READ A$(I),B(I)
140 NEXT I
150 PRINT A$(4),B(2)
160 DATA "HERB",165,"TOM",18
3
170 DATA "SAM",145,"BILL",19
2
180 END
```

4. Write a program to input a list of numbers, then find and print out the sum of the positive numbers in the list.

5. We have a string array named X. What variable name does BASIC use to locate the element in row 2, column 4?

6. Write a program using FOR NEXT statements to load a four by six array with 4s. Then print out the array.

7. What will be printed out if the program is RUN?

```
100 DIM A(5,5)
110 OPTION BASE 1
120 FOR I=1 TO 5
130 FOR J=1 TO 5
140 LET A(I,J)=0
150 NEXT J
160 NEXT I
170 FOR I=1 TO 5
180 LET A(I,I)=2
190 NEXT I
200 FOR I=1 TO 5
210 FOR J=1 TO 5
220 PRINT A(I,J);
230 NEXT J
240 PRINT
250 PRINT
260 NEXT J
270 END
```

8. 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))?

____

9. What is the purpose of the OPEN statement?

____

10. What is the purpose of the CLOSE statement?

____

CHAPTER

NINE

"DO-IT-YOURSELF" FUNCTIONS AND SUBROUTINES

9-1 OBJECTIVES

In this chapter we will learn how the computer can be programmed to perform suboperations. This can be done through either program segments or special on-line instructions. Specifically, we will look at the following things.

"Do-It-Yourself" Functions

We have previously seen functions that are built into BASIC. Now we will learn how to define our own functions involving either string or numeric variables to carry out any desired task.

Subroutines

When complicated operations are to be repeated, subroutines may be very useful. We will explore how subroutines can be set up and used in BASIC programs.

Program Applications

Sometimes it is difficult for the beginner to see the value of user-defined functions and subroutines. These ideas will be stressed in our continued attention to programming in BASIC.

9-2 DISCOVERY ACTIVITIES

1. Turn your computer on and enter the following program:

```
100 DEF FNA(X)=5*X+4
110 LET X=2
120 LET Y=5*X+4
130 PRINT Y,FNA(2)
140 END
```

RUN the program and record the output below.

2. Change line 130 to read

```
130 PRINT Y,FNA(X)
```

Display the program. What do you think will happen if we RUN this program?

RUN the program. What did happen?

3. Change line 110 to read

```
110 LET X=5
```

Display the program and study it. Now what will be output if we RUN the program?

See if you were right. RUN the program and record what happened.

4. Now change line 130 to read

```
130 PRINT Y,FNA(5)
```

Display the program. What do you think this program will do?

RUN the program and write down the output.

5. Notice that the expressions after the equal signs in lines 100 and 120 of your program are the same. In one of the versions of the program, we printed out Y and FNA(X) and saw that they were the same. Let's follow up on this information. Clear out the program in memory and enter the following program:

```
100 DEF FNA(X)=X^2
110 DEF FNB(X)=3*X
120 DEF FNC(X)=X+2
130 LET X=1
140 PRINT FNA(X),FNB(X),FNC(
X)
150 END
```

Study the program carefully. What do you think will be printed out if the program is executed?

Now RUN the program and write down what happened.

____

Substitute 1 for X in the expressions on the right side of lines 100, 110, and 120 in your program. Write down the numbers you obtain.

____

Now compare these numbers with those printed out by the computer.

6. Change line 130 to read

```
130 LET X=2
```

Display the program. What will be printed out by the program if it is RUN now?

____

See if you were right. RUN the program and record the results below.

____

7. OK, change line 130 to

```
130 LET X=3
```

Now what will happen if the program is RUN?

____

Verify your answer by executing the program and recording what happened.

---

8. Now on to some more ideas we can explore with this program. Type

```
130 LET X=1
140 PRINT FNC(X+4),FNA(X),FN
B(2)
```

Display the program. Write down what you think will be printed out if the program is RUN.

RUN the program and record the output.

---

9. Let's try a slightly different variation on the theme we have been exploring. Type

```
140 PRINT FNA(X),FNB(FNA(X))
```

Display the program and study it carefully. Try to figure out what will be printed out when the program is RUN. Record your answer below.

RUN the program and see if you were correct. Write down below what happened.

---

10. One more point on this matter. Type

```
130 LET X=4
140 PRINT FNA(X),FNC(X),FNA(
SQR(X))
```

Now what will happen in the program?

___

RUN the program and record what happened.

___

11. So far we have been working solely with numbers in DEF statements. We can also set up DEF statements that work on strings. Clear out the memory and enter the following program:

```
100 DEF SPACE$(A$)=SEG$(A$,1
,2)&CHR$(32)&SEG$(A$,3,LEN(A
$)-2)
110 INPUT N$
120 PRINT SPACE$(N$)
130 GOTO 110
140 END
```

The defined function in line 100 has the name SPACE$. The $ symbol at the end of the name indicates the defined function involves strings. Study the definition of SPACE$ briefly. RUN the program and at the input prompt type in CHARLES. What was printed out?

___

Now type in SARAH. What happened?

___

By this time you should see that SPACE$ inserts a space between the second and third characters in the string the function operates on. Whether or not there is a need for such a function is not the point. Our purpose here is to demonstrate how the DEF statements can involve strings. Jump the computer out of the INPUT loop.

12. Clear out the program in memory and enter the new program below.

```
100 DEF PI=3.141592654
110 INPUT "RADIUS=":RADIUS
120 LET CIRCUM=2*PI*RADIUS
130 PRINT "CIRCUMFERENCE=";C
IRCUM
140 GOTO 110
150 END
```

This program is simple and its purpose is obvious. Line number 100 illustrates still another type of DEF statement. RUN the program and try various numerical inputs. Then jump the computer out of the INPUT loop.

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

```
100 PRINT "A";
110 GOSUB 200
120 PRINT "B";
130 GOSUB 300
140 PRINT "C";
150 STOP
200 PRINT 1;
210 RETURN
300 PRINT 2;
310 RETURN
400 END
```

This program has three new statements that you haven't seen so far. These are GOSUB, RETURN, and STOP. The program itself is intended only to provide practice in tracing these new statements. Execute the program and record the output.

____

Compare what was printed out with the program lines that caused the printout.

14. The GOSUB statement in line 110 transfers the program to which statement? (Hint: Look at the printout in step 13.)

____

15. The RETURN statement in line 210 transfers the program to which statement? (Hint: Again, examine the printout in step 13.)

____

16. The line numbers below indicate 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	Transfer to line 400
400	End of program

Study this carefully and follow through with the program. Can you see the purpose of the GOSUB and RETURN statements yet? What about the STOP statement?

17. Clear out the program in your work space. Enter the following program:

```
100 REM SUBR. DEMO
110 DIM X(4)
```

```
120 READ X(1),X(2),X(3),X(4)
130 REM SORT
140 GOSUB 300
150 REM PRINT
160 GOSUB 400
170 LET X(3)=7
180 REM SORT
190 GOSUB 300
200 REM PRINT
210 GOSUB 400
220 STOP
300 REM SORT SUBR
310 FOR I=1 TO 3
320 IF X(I+1)>X(I) THEN 370
330 LET C=X(I+1)
340 LET X(I+1)=X(I)
350 LET X(I)=C
360 GOTO 310
370 NEXT I
380 RETURN
400 REM PRINT SUBR.
410 PRINT X(1),X(2),X(3),X(4
)
420 RETURN
500 DATA 2,1,5,6
600 END
```

Display the program and check that you have entered it correctly. This program furnishes an example of how a subroutine might be used. The subroutine in lines 300 through 380 sorts the array X into ascending order. The subroutine in lines 400 through 420 prints out the array. 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 the program. In line 140, the program jumps to the subroutine and a sort of the numbers is done. After the program returns to line 150, the sorted array is now

1 2 5 6

In line 170 we change the third element of the array, then branch to the subroutine for another sorting. After the return to line 200, the sorted array

```
1    2    6    7
```

is printed out. Finally, the STOP command in line 200 causes the program to jump to the END statement. Clearly we could sort the array X as often as desired by merely inserting a statement GOSUB 300. This is certainly more efficient than writing out the instructions for sorting each time it is desired.

18. This completes the discovery work for this chapter. Turn off your computer and go on to the next section.

9-3 DISCUSSION

Now we need to examine the ideas introduced in the computer work. Once you understand clearly how the computer handles these concepts, you will have powerful new skills to use in your programs.

"Do-It-Yourself" Functions

The DEF (an abbreviation for "define") statement permits us to have user-specified functions in BASIC in addition to those functions (SQR, INT, etc.) already built into the language. The DEF statements can be either numeric or string. The easiest way to learn about DEF statements is to look at typical examples.

```
100 DEF FNA(X)=X*4-1
110 DEF PI=3.141592654
120 DEF TAX(N)=(N-20)*.15
130 DEF ROTATE$(S$)=SEG$(S$,
2,LEN(S$)-1)&SEG$(S$,1,1)
```

By discussing how each of these sample statements works we can quickly see how DEF statements can be used to advantage in programs. The DEF statement in line 100 is easy to understand. If FNA(2) were to be used in a program, the computer would substitute 2 for X on the right side of the expression in the DEF statement. The result is that FNA(2) would be evaluated as seven. Likewise, if Y had the value six, FNA(Y) would be evaluated as twenty-three. We can even do things like FNA(SQR(Z)+1.5). The point is that the argument of the function (the thing that appears in parentheses after FNA) is converted to a number which is then substituted for X in the DEF statement.

The DEF statement in line 110 is very useful. Often constants are used in programs. In this example, the constant PI is defined to be 3.141592654. Later, we can use PI in the program rather than the numeric value. This capability is very useful where constants commonly go by their names rather then their numeric values. Of course we could also set up the constant using the LET statement if desired.

The purpose of the DEF statement in line 120 is to point out that we can use any name we want in the DEF statement. In this case, the tax is 15% of the amount by which N exceeds twenty. The DEF statement sets this up with the name TAX(N).

The final example of a DEF statement is in line 130. Here the DEF statement involves strings. The ROTATE$ rotates a character from the beginning of a string to the end. Thus HOUSE becomes OUSEH, BIRD becomes IRDB, and so on.

Define your own functions with a DEF statement.

The primary purpose of the user-specified functions that are set up with the DEF statements is to simplify programming by avoiding repeated use of complicated expressions. The DEF statements as implemented on the TI Home Computer are much more powerful than found in most versions of BASIC. The wise programmer should be alert for opportunities to save effort with the use of DEF statements.

Subroutines

One of the limitations of the DEF statements is that only a single variable may be involved and we are limited to a single line. More complicated situations in which we want to carry out the same process many times in a program are bound to come up. Here is where subroutines are very useful. The diagram below indicates how a subroutine might be used in a program.

Main program begins

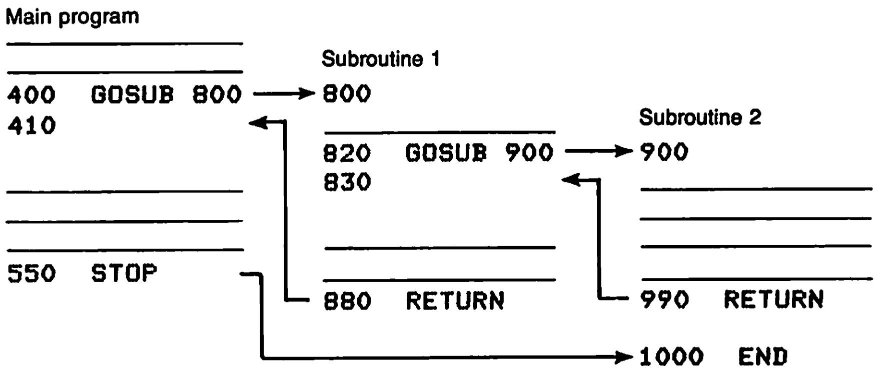
```
____________________________
____________________________
____________________________
200  GOSUB 1000
210
____________________________
____________________________
____________________________
350  GOSUB 1000
360
____________________________
```

	______________________
Main program ends	`430 STOP`
Subroutine begins	`1000 REM SUBROUTINE`
	______________________
	______________________
	______________________
End of subroutine	`1150 RETURN`
End of program	`1200 END`

If the typical program above were executed, when the computer reached the GOSUB in line 200, the program would jump to the beginning of the subroutine in line 1000. The subroutine would be executed, and when the RETURN was encountered in line 1150, control would be passed to the next higher line number after the GOSUB that put us in the subroutine. In this case the program would jump back to line 210. Then the computer would proceed through the main program to the GOSUB in line 350 which would again branch control to the subroutine in line 1000. This time the RETURN would jump back in the program to line 360.

Of course, we could have used GOSUB 1000 as many times as we wanted in the program or could have had as many subroutines as needed. Generally, the top part of the program is the main program and the subroutines are grouped together at the end. There is a good reason for this. We want to perform the subroutines only when called for by a GOSUB. Thus, after the main program is finished, we put a STOP statement in the program. This is precisely the same as a GOTO the END statement and jumps across all the subroutines grouped together at the end of the program. We can use the STOP statement anywhere there is a logical end to the program. This may occur several times in any given program.

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.



```
400   GOSUB 800
410
550   STOP
```

```
800
820   GOSUB 900
830
880   RETURN
```

```
900
990   RETURN
1000   END
```

Transfer to subroutines with a GOSUB statement.

Note that control passes from 400 to 800, on down to 820, to 900, and on down to the RETURN in line 990. Of course, the question here is, does the RETURN take us back to line 410 or line 830? The answer is determined by the rule that the RETURN takes us back to the next statement after the GOSUB that put us in the subroutine containing the RETURN. We are in subroutine 2 because of the GOSUB in line 820; hence the RETURN in line 990 branches us back to line 830. The same rule applies when we reach the RETURN in line 880. At that point we are in subroutine 1 and were put there by the GOSUB in line 400. Thus, the RETURN in line 880 carries us back to line 410. Finally, the STOP statement in line 550 jumps control to the END statement in line 1000.

Get back from subroutines with a RETURN statement.

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. It is enough at this time to point out that subroutines are extremely important and are considered to be one of the most powerful tools available to the programmer.

9-4 PROGRAM EXAMPLES

Several programs should assist you to master the ideas involved in both user-defined functions and subroutines.

Example 1 - Rounding Off Dollar Values to Cents

Business applications generally involve printing out the results of calculations in dollars and cents. Since your computer handles ten significant figures in calculations, we might get an amount like 23.15976431 typed out. This looks strange, and to solve the problem, we should round off the figure to the nearest cent, or 23.16.

This is an ideal application of a user-defined function. Let's write a program that will produce the following typical output when RUN:

```
LABEL PRICE ? 22.80
10% DISCOUNT IS 20.52
15% DISCOUNT IS 19.38
20% DISCOUNT IS 18.24
```

All dollar values typed out should be rounded off to the nearest cent.

First, we must define a function to do the rounding. Such a function is

```
100 DEF ROUND(X)=INT(X*100+.
5)/100
```

To see how this rule works, suppose X = 23.15976431. We can follow this value through the expression to see what happens.

X*100 = 2315.976431
X*100+0.5 = 2316.476431
INT (X*100+0.5) = 2316
INT(X*100+0.5)/100 = 23.16

Therefore 23.15976431 was correctly rounded up to 23.16.

As a second example, suppose that X = 23.15472563. Then

X*100 = 2315.472563
X*100+0.5 = 2315.972563
INT(X*100+0.5) = 2315
INT(X*100+0.5)/100 = 23.15

with the result that 23.15472563 was correctly rounded down to 23.15.

The next few lines of the program are self-explanatory.

```
110 PRINT "LABEL PRICE";
120 INPUT Z
130 PRINT "10% DISCOUNT IS";
ROUND(.9*Z)
140 PRINT "15% DISCOUNT IS";
ROUND(.85*Z)
150 PRINT "20% DISCOUNT IS";
ROUND(.8*Z)
```

If desired, we can loop back to the beginning with

```
160 GOTO 110
```

and then end the program.

```
170 END
```

The complete program is

```
100 DEF ROUND(X)=INT(X*100+.
5)/100
110 PRINT "LABEL PRICE";
120 INPUT Z
130 PRINT "10% DISCOUNT IS";
ROUND(.9*Z)
140 PRINT "15% DISCOUNT IS";
ROUND(.85*Z)
150 PRINT "20% DISCOUNT IS";
ROUND(.8*Z)
160 GOTO 110
170 END
```

In lines 130, 140, and 150 the defined function is used. For a 10 percent discount, the selling price is 90 percent of the original label price Z. Hence we print out ROUND(0.9*Z), which rounds off the value to the nearest cent as desired. Note the economy of using the defined function rather than writing out the expression in line 100 each time we want to print out a rounded dollar amount.

Example 2 - Carpet Estimating

We want to write a program that uses a subroutine to compute the price of installed carpet. Suppose that there are four grades of carpet and each is discounted as the quantity of carpet ordered increases. We will assume that the price structure is as follows:

		Price per square yard		
		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

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 RUN, the program should produce the following typical output:

```
HOW MANY ROOMS ? 4
FOR EACH ROOM TYPE IN
LENGTH AND WIDTH IN FEET
SEPARATED BY A COMMA

ROOM    DIMENSIONS

1       ? 10,12
2       ? 12,15
3       ? 12,8
4       ? 15,25
85.67 SQ YDS REQUIRED

CARPET GRADE    ORDER COST

A               674.83
B               910.25
C               1062.5
D               1197.17
```

Before getting involved in the program, we should think a bit about the output. Since the output is in dollars and cents, we may as well use the defined function from Example 1 to take care of rounding off the answers properly. We can also use the rounding function to round off the number of yards of carpet required to the nearest hundredth. So let's begin the program with that defined function.

```
100 DEF ROUND(X)=INT(X*100+.
5)/100
```

The next few lines follow without difficulty.

```
110 PRINT "HOW MANY ROOMS";
120 INPUT N
```

```
130 PRINT "FOR EACH ROOM, TY
PE IN"
140 PRINT "LENGTH AND WIDTH
IN FEET"
150 PRINT "SEPARATED BY A CO
MMA"
160 PRINT
170 PRINT "ROOM","DIMENSION"
180 PRINT
```

Now we are ready to call for the input of the room dimensions. We will use the variable AREA to keep track of the area of the rooms. Remember that the area of a room is its length times its width.

```
190 LET AREA=0
200 FOR I=1 TO N
210 PRINT I,
220 INPUT L,W
230 LET AREA=AREA+L*W
240 NEXT I
```

Since the total room area is now in square feet, we must divide this by 9 to convert to square yards, and then we will print out the quantity of carpet required rounded to two places past the decimal point.

```
250 LET YARDS=AREA/9
260 PRINT ROUND(YARDS);"SQ Y
ARDS REQUIRED"
```

At this point we may as well include the price table in the program in the form of DATA statements.

```
270 DATA 10,8.5,7.25
280 DATA 13.25,12,9.75
290 DATA 16,14,11.25
300 DATA 20,17.2,15.25
```

Next we can print out the heading required for the price printout.

```
310 PRINT
320 PRINT "CARPET GRADE","OR
DER COST"
330 PRINT
```

Now we come to the point in the program where the subroutine will be useful. Since we don't know precisely where the subroutine should begin, we will simply use a large line number and correct it later if needed.

```
340 REM COMPUTE PRICE FOR GRA
DE A
350 GOSUB 800
```

Let's write the subroutine now. First, for each of the grades of carpet we need the three prices. We can do this by reading them from the DATA statements.

```
800 REM SUBROUTINE TO COMPUT
E CARPET PRICE
810 READ C1,C2,C3
```

Next we check to see 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 YARDS>25 THEN 860
830 IF YARDS>15 THEN 880
840 LET P=C1*YARDS
850 GOTO 890
860 LET P=15*C1+10*C2+(YARDS
-25)*C3
870 GOTO 890
880 LET P=15*C1+(YARDS-15)*C
2
890 RETURN
```

Trace this program segment through to convince yourself that the price is being computed correctly. Now we can return to the main program and print out the first price.

```
360 PRINT "A",ROUND(P)
```

Once this pattern has been established, the rest of the main program follows easily.

```
370 REM COMPUTE PRICE FOR GR
ADE B
380 GOSUB 800
390 PRINT "B",ROUND(P)
400 REM COMPUTE PRICE FOR GR
ADE C
410 GOSUB 800
420 PRINT "C",ROUND(P)
430 REM COMPUTE PRICE FOR GR
ADE D
440 GOSUB 800
450 PRINT "D",ROUND(P)
460 STOP
```

The STOP statement in line 460 is needed to prevent the program from falling into the subroutine. The value of the subroutine becomes clear when we see that had it not been available, each of the four GOSUB statements would have had to be replaced with as many statements as in the subroutine.

The complete program is

```
100 DEF ROUND(X)=INT(X*100+.
5)/100
110 PRINT "HOW MANY ROOMS";
120 INPUT N
130 PRINT "FOR EACH ROOM, TY
PE IN"
140 PRINT "LENGTH AND WIDTH
IN FEET"
150 PRINT "SEPARATED BY A CO
MMA"
160 PRINT
170 PRINT "ROOM","DIMENSIONS
"
180 PRINT
190 LET AREA=0
200 FOR I=1 TO N
210 PRINT I,
220 INPUT L,W
230 LET AREA=AREA+L*W
240 NEXT I
250 LET YARDS=AREA/9
260 PRINT ROUND(YARDS);"SQ Y
ARDS REQUIRED"
270 DATA 10,8.5,7.25
280 DATA 13.25,12,9.75
290 DATA 16,14,11.25
300 DATA 20,17.2,15.25
```

```
310 PRINT
320 PRINT "CARPET GRADE","OR
DER COST"
330 PRINT
340 REM COMPUTE PRICE FOR GR
ADE A
350 GOSUB 800
360 PRINT "A",ROUND(P)
370 REM COMPUTE PRICE FOR GR
ADE B
380 GOSUB 800
390 PRINT "B",ROUND(P)
400 REM COMPUTE PRICE FOR GR
ADE C
410 GOSUB 800
420 PRINT "C",ROUND(P)
430 REM COMPUTE PRICE FOR GR
ADE D
440 GOSUB 800
450 PRINT "D",ROUND(P)
460 STOP
800 REM SUBROUTINE TO COMPUT
E CARPET PRICE
810 READ C1,C2,C3
820 IF YARDS>25 THEN 860
830 IF YARDS>15 THEN 880
840 LET P=C1*YARDS
850 GOTO 890
860 LET P=15*C1+10*C2+(YARDS
-25)*C3
870 GOTO 890
880 LET P=15*C1+(YARDS-15)
*C2
890 RETURN
900 END
```

Example 3 - Home Inventory

As a final example we will write a program to process information about items in your home and then write this information on a cassette tape. The information is that which would be necessary for an insurance claim in the event your home was damaged by fire.

The information will be written in a record (a block of characters) fifty-one characters long. Unused space in the record will be filled with blank spaces. Character 1 will be a space. Characters 2 through 16 will hold the room name. Characters 17 through 31 will contain the item name. In both these pieces of information, if the full fifteen characters are not used, trailing blank spaces will be appended.

Characters 32 and 33 will contain the year the item was purchased. Characters 34 through 42 will hold the purchase price of the item, and the current value will be stored in characters 43 though 51. If all nine characters are not used in these numbers, leading blanks will fill the unused space.

The program should call for input of the necessary information, check that the input is correct, convert all numeric quantities to strings, assemble the fifty-one character record that describes an item, and finally write that record to the tape cassette. Since you have had a great deal of experience with the computer by this time, we will depart from the usual practice of discussing examples in detail, and will instead give you the complete program. You should go through this program in detail until you understand exactly what is happening. As well as illustrating how subroutines can be used, this example is a good review of topics discussed earlier in the book.

```
100 OPEN #1:"CS1",OUTPUT,FIX
ED 51
110 LET A$= " "
115 INPUT "ROOM ":X$
120 GOSUB 700
130 INPUT "ITEM ":X$
140 GOSUB 700
150 INPUT "YEAR PURCHASED ":
X$
160 LET X$=SEG$(X$,LEN(X$)-1
,2)
170 LET A$=A$&X$
180 INPUT "PURCHASE PRICE ":
P
190 GOSUB 800
200 INPUT "CURRENT VALUE ":P
210 GOSUB 800
220 PRINT #1:A$
230 GOTO 110
500 REM SBR TO PAD WITH TRAI
LING BLANKS
505 LET X$=""
510 FOR I=1 TO N
520 LET X$=X$&CHR$(32)
530 NEXT I
540 RETURN
600 REM SBR TO PAD WITH LEAD
ING BLANKS
605 LET X$=""
610 FOR I=1 TO N
620 LET X$=CHR$(32)&X$
630 NEXT I
640 RETURN
700 REM CHECK STRING FOR LEN
GTH
710 IF LEN(X$)<=15 THEN 740
```

```
720 LET X$=SEG$(X$,1,15)
730 GOTO 760
740 LET N=15-LEN(X$)
750 GOSUB 500
760 LET A$=A$&X$
770 RETURN
800 REM CHECK FORMAT OF PRIC
E
810 LET X$=STR$(P)
820 IF SEG$(X$,LEN(X$)-2,1)
=CHR$(46) THEN 860
830 LET B$=".00"
840 LET X$=X$&B$
850 LET N=15-LEN(X$)
860 GOSUB 600
870 LET A$=A$&X$
880 RETURN
900 END
```

9-5 PROBLEMS

1. Trace the program below and write down what will be printed out if the program is executed.

```
100 DEF FNA(X)=2+X
110 DEF FNB(Y)=10*Y
120 DEF FNC(Z)=Z^2
130 LET R=2
140 LET S=3
150 LET T=5
160 PRINT FNC(T),FNA(S),FNB(
R)
170 LET R=S+T
180 PRINT FNA(R)+FNB(S)+FNC(
T)
190 END
```

2. What will be printed out if the program below is executed?

```
100 DEF FNX(A)=6*A
110 DEF FNY(B)=B+10
120 DEF FNZ(C)=C^3
130 READ P,Q,R
140 DATA 1,2,3
```

```
150 PRINT FNX(R),FNZ(P),FNY(
Q)
160 PRINT FNY(P+Q)+FNX(R)
170 END
```

3. What will be output by the following program if it is executed?

```
100 DIM A(5)
110 OPTION BASE 1
120 READ A(1),A(2),A(3),A(4
),A(5)
130 DATA 6,2,7,1,3
140 GOSUB 500
150 PRINT A(1);A(2);A(3);A(4
);A(5)
160 LET A(3)=10
170 GOSUB 500
180 PRINT A(1);A(2);A(3);A(4
);A(5)
190 LET A(5)=8
200 GOSUB 500
210 PRINT A(1);A(2);A(3);A(4
);A(5)
220 STOP
500 FOR I=1 TO 4
510 LET A(I)=A(I+1)
520 NEXT I
530 RETURN
600 END
```

4. What will be printed out if the program below is executed?

```
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 STOP
500 LET S=0
510 FOR Y=1 TO X
520 LET S=S+Y
530 NEXT Y
540 RETURN
600 END
```

5. Assume that a one-dimensional array Z contains the numbers to be added together. The first element of the array, Z(0), 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(0). 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.

6. X is a one-dimensional array. The first element of the array, X(0), gives the number of pieces of data that follow in the array. Write a subroutine beginning in line 500 to search through the 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.

7. Write a program to reverse the process described in Example 3. The program should input record blocks fifty-one characters long from a cassette tape. Assume that the first number on the tape contains the number of records that follow. After each record is input, decode and print the information on the screen.

8. 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 the standard deviation (S) of the numbers in the array which 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.

$$\text{Mean} = \text{Sum of values} / N$$

$$\text{Standard deviation} = \sqrt{\frac{N \times (\text{sum of squares of values}) - (\text{sum of values})^2}{N \times (N-1)}}$$

9-6 PRACTICE TEST

Check your progress with the following practice test. The answers are given at the end of the book.

1. If DEF FNA(X) = SQR(x)+3*X, Z = 2.5, and W = 10, what is

   a. FNA(1)

   b. FNA(4)

   c. FNA(9)

   d. FNA(Z*W)

2. What will be printed out if we execute the following program?

```
100 DEF FNR(X)=X*X
110 DEF FNS(X)=3*X
120 DEF FNT(Y)=Y+1
130 LET A=1
140 PRINT FNT(A),FNR(A),FNS(
A)
150 LET M=4
160 PRINT FNR(SQR(M))
170 END
```

3. With regard to subroutines

   a. How do you pass control from the main program to the subroutine?

   b. How do you pass control from the subroutine back to the main program?

   c. What is the purpose of the STOP statement?

4. What will be printed out if we 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 STOP
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
900 END
```

CHAPTER

TEN

RANDOM NUMBERS AND SIMULATIONS

10-1 OBJECTIVES

One of the most interesting applications of computers concerns simulation of events or processes that involve an element of chance. Examples might be using the computer to simulate gambling games or perhaps investigating 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. Our objectives are as follows.

Characteristics of Random-Number Generators

Computers have a random-number generator function that is the heart of all programs involving the element of chance, or randomness. We will learn how these random-number generators can be employed in BASIC programs.

Random Numbers with Special Characteristics

Generally, the random-number generator is used to produce sets of random numbers with characteristics specified by the programmer. We will see how this is done and how any desired set of numbers can be generated.

Programming and Simulations

The programming exercises and problems in this chapter will involve simulations and applications that involve the element of chance.

10-2 DISCOVERY ACTIVITIES

Setting Up the Random-Number Generator

Before beginning the computer work, we must discuss some important characteristics of random-number generators. By their very nature, these generators produce sequences of numbers that appear to have no pattern or relationship. For a random-number generator to be useful, each time we execute a program that utilizes it we should get a different sequence of numbers. However, this gives rise to an interesting question. Suppose a program that uses random numbers is not working correctly. If the problem is connected with the random numbers, it might be extremely difficult to correct since different random numbers are generated each time the program is executed. Consequently, provisions are always included so that a sequence of random numbers can be repeated each time the program is executed. Remember that this feature of BASIC should be used only when you are troubleshooting a program.

On the TI Home Computer we control the type of random-number sequence by the presence or absence of the RANDOMIZE statement. If the program contains a RANDOMIZE statement, a different sequence of numbers is generated each time the program is RUN. Otherwise, the same sequence of random numbers is generated.

Now, let's go on to the discovery work.

1. Turn your computer on. Unless otherwise specified, we will use a RANDOMIZE statement in all programs to generate different sequences of random numbers.

2. Enter the following program:

```
100 RANDOMIZE
110 FOR I=1 TO 10
120 PRINT RND
130 NEXT I
140 END
```

RUN the program and 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. Clear out the program in memory and enter the following program:

```
100 RANDOMIZE
110 LET L=.5
120 LET S=.5
130 FOR I=1 TO 100
140 LET X=RND
150 IF X>L THEN 180
160 IF X<S THEN 200
170 GOTO 210
180 LET L=X
190 GOTO 210
200 LET S=X
210 NEXT I
220 PRINT "LARGEST = ";L
230 PRINT "SMALLEST = ";S
240 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.

5. Change line 120 to read

```
120 FOR I=1 TO 1000
```

Now the program will generate 1000 random numbers. RUN the program and record what was printed out.

____

Based upon what you have seen thus far, what do you believe is the largest number that will be generated by the RND function?

____

What about the smallest?

____

6. Now let's go on to some other ideas associated with random numbers. Clear out the program in memory and enter the following program:

```
100 RANDOMIZE
110 FOR I=1 TO 10
120 PRINT INT(2*RND)
130 NEXT I
140 END
```

Execute the program and record the output.

____

What were the only two numbers in the printout?

____

7. Change line 120 to read as follows:

```
120 PRINT INT(3*RND)
```

Display the program. If this program is executed, what numbers do you think will be typed out?

RUN the program and write down the output. Can you predict anything about the sequence or pattern in which the numbers will be typed out?

8. Now change line 120 to read

```
120 PRINT INT(2*RND+1)
```

What do you think the program will do now?

Execute the program and record the output.

9. Modify line 120 as follows:

```
120 PRINT INT(4*RND+4)
```

If the program is executed, what do you think will be printed out?

RUN the program and describe the output.

Any pattern to the output?

---

10. OK, change line 120 as follows:

```
120 PRINT INT(30*RND)/10
```

Display the program and study it carefully. What do you think this program will print out?

Execute the program and describe the printout.

---

11. Finally, change line 120 to read

```
120 PRINT INT(200*RND)/100
```

Display the program in your work space. What do you think will happen if this program is executed?

See if you were right. Execute the program and record the output below.

---

12. Turn your computer off. This terminates the computer work for now.

10-3 DISCUSSION

Now that you have seen some of the characteristics of the random-number generator on the computer, we can profitably proceed to a complete discussion of the matter.

Random-Number Generators

We will not become involved with the details of how random numbers are generated. It is enough to say that there are several mathematical methods to produce these numbers. The random-number generator is called on with the RND function. This function is used like the other built-in functions in BASIC that were studied previously, but differs in two important respects. 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. However, the RND function has no argument.

In the introductory material, it was pointed out that depending on the RANDOMIZE statement we can get two different types of sequences. This bears repeating here. First, if the program contains a RANDOMIZE statement, we will get a different sequence of random numbers each time the program is run. If there is no RANDOMIZE statement, we will get the same sequence of numbers each time the program is used. This is the first major difference in the RND function compared to the others we have studied.

The second major difference is that there seems to be no pattern or rule used in generating numbers with the RND function. Of course, this is precisely the point of the function. RND stands for "random." The function generates numbers between 0 and 1 at random. All the numbers in the inverval have an equal chance of showing up. Actually, the range of numbers generated is from 0.0000000000 to 0.9999999999. Zero can show up very rarely, but the number 1 never occurs.

RND generates random numbers in the range 0.0000000000 to 0.9999999999.

A good way to visualize how the random-number generator works is to imagine the following situation. We have 10 billion chips numbered 0.0000000000, 0.0000000001, 0.0000000002, and so on up to 0.9999999998, and 0.9999999999. The chips are all placed in a large container and mixed thoroughly. If we want a random number, we reach into the container and withdraw a single chip, read the number, return the chip to the container, and then mix all the chips again very thoroughly. The RND function works exactly the same way and can be used in BASIC programs anytime we want a random number.

Designing Sets of Random Numbers

Most often we do not want random numbers in the range produced by the RND function, that is, from zero to one. We might want random integers (whole numbers) over a certain range or a set of random numbers with a particular set of characteristics. Therefore, we must give some thought to how to generate sets of random numbers with characteristics we can specify.

Let's begin with the characteristics of random numbers. RND delivers numbers from 0 to just less than one. If we multiply RND by N, we multiply the range of the function by N. Thus N*RND will produce random numbers from zero to just less than N. If desired, we could shift the numbers (keeping the same range) by adding a number. N*RND+A would produce random numbers from A to just less than (A+N). Finally, if desired, we could take the integer part of an expression, using the INT function, to produce random integers. The examples below indicate how the RND function might be used.

BASIC Expression	Result
5*RND + 10	Random numbers in the range 10 to 15
INT(5*RND + 10)	Random integers 10,11,12,13,14
INT(2*RND + 1)	Random integers 1,2
100*RND	Random numbers in the range 0 to 100

You may have encountered the notion of mean and standard deviation (see problem 8 in Chapter 9). We 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. This is an application in which a subroutine would be very useful. As defined above, the values of X will appear to be coming from a collection of numbers with mean M and standard deviation S. The values of X can be used to simulate a process following the "bell curve" that is often referred to.

Troubleshooting Programs That Use Random Numbers

We have already pointed out that BASIC provides a way to execute a program several times and repeat the sequence of numbers that are generated by the RND function. It is usually wise to write programs initially so that they do generate the same sequence of numbers each time they are executed. Once you are sure that the program is working correctly, you can insert a RANDOMIZE statement to produce the randomness that is the central idea in the RND function.

10-4 PROGRAM EXAMPLES

Now we will go through several examples to 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. We want to write a program that when executed will produce the following typical 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 contains the RANDOMIZE statement, and generates the heading and the space indicated in the printout above.

```
100 RANDOMIZE
110 PRINT "TOSS","OUTCOME"
120 PRINT
```

Now we must open the loop to generate the ten tosses of the coin.

```
130 FOR I=1 TO 10
```

The next step is to generate 0s and 1s randomly. We will assume that the occurrence of a 0 means a "head" and the occurrence of a 1 means a "tail." You should be able to convince yourself that the following statement will produce 0s and 1s randomly.

```
140 LET X=INT(2*RND)
```

Now we analyze X to see whether a head (0) or a tail (1) has occurred.

```
150 IF X=0 THEN 180
160 PRINT I,"T"
170 GOTO 190
180 PRINT I,"H"
190 NEXT I
```

All that remains now is the END statement.

```
200 END
```

The complete program is listed below.

```
100 RANDOMIZE
110 PRINT "TOSS","OUTCOME"
120 PRINT
130 FOR I=1 TO 10
140 LET X=INT(2*RND)
150 IF X=0 THEN 180
160 PRINT I,"T"
170 GOTO 190
180 PRINT I,"H"
190 NEXT I
200 END
```

This is a good program for demonstrating how the computer can be instructed to produce either different sequences of random numbers or identical sequences each time the program is executed. Remove the RANDOMIZE statement to see identical sequences produced.

Example 2 - Random Integers

The next problem is to write a BASIC program to generate and print out fifty random integers (whole numbers) over the range 10 to 15. The only part of the program that will require much thought is the statement to generate the random integers, so we will concentrate on this one statement.

Remember that the RND function generates numbers over the range from zero to slightly less than one. By using the integer function we can convert from random numbers to random integers. INT(6*RND) will produce the integers 0, 1, 2, 3, 4, 5 randomly. Now it is clear that to get the desired numbers, we must add 10. Thus, the expression INT(6*RND)+10 will produce the numbers we want.

Once we have this one line figured out, the program follows easily.

```
100 RANDOMIZE
110 FOR I=1 TO 50
120 LET Y=INT(6*RND)+10
130 PRINT Y;
140 NEXT I
150 END
```

Example 3 - Birthday Pairs in a Crowd

Suppose that fifty strangers get together in a room. What is the probability that two of the people have the same birthday? We consider only the day of the year, not the year of birth. This problem is a famous one in probability theory and has surprising results. We can attack the problem with the following strategy. By generating random integers over the range 1 to 365, we can simulate a birthday for each of the strangers. If we 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), we check to see if it matches any of the remaining ones. Then we do the same thing for B(2), and so on.

For this example, we will depart from the usual method and will look at the complete program, then go back and explain what is taking place in each line.

```
100 RANDOMIZE
110 DIM B(50)
120 FOR I=1 TO 50
130 LET B(I)=INT(365*RND)+1
140 NEXT I
150 LET F=0
160 FOR I=1 TO 49
170 FOR J=I+1 TO 50
180 IF B(I)<>B(J)THEN 200
190 LET F=F+1
200 NEXT J
```

```
210 NEXT I
220 PRINT "NUMBER OF BIRTHDA
Y"
230 PRINT "PAIRS FOUND IS";F
240 END
```

Of course, line 110 merely dimensions an array for fifty elements. Lines 120 through 140 load the array with random integers selected over the range 1 to 365 inclusive. In line 150, we set the variable F equal to zero. We will use this variable to keep track of the number of birthday pairs we find. Line 160 opens a loop to identify the birthday that will be compared with the rest in the list. Since we have to have at least one birthday left in the list to compare with, the value of I stops at 49. In line 170, 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 180. If no match is found, we jump to the next value of J. If a match is found, the pair counter is increased by 1 in line 190. The results are printed out in line 220. 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 an extremely interesting program to experiment with. The number of people in the crowd can be modified with simple changes in the program. The program can be executed many times to see how many birthday pairs on the average will be found in a crowd of a specified size.

Example 4 - Word Generator

We can use the random-number generator to make up words. Suppose you are given the job to come up with new names for laundry products. You decide that the names should be five characters long. The first, third, and fifth characters will be consonants. The second and fourth characters will be vowels. Random numbers will be used to pick the vowels from the list "AEIOU", and the consonants from the list "BCDFGHJKLMNPQRSTVWXYZ."

We will write a BASIC program to enable the computer to generate a block of twenty words as described above. First we define the string variables that contain the vowels and consonants.

```
100 RANDOMIZE
110 LET A$="AEIOU"
120 LET B$="BCDFGHJKLMNPQRST
VWXYZ"
```

We will need random integers (whole numbers) over the range 1-5 to select a vowel, and integers over the range 1-21 to select a consonant. This is an ideal application for DEF statements. We will use X as the argument of the DEF statements and will set it equal to 1.

```
130 LET X=1
140 DEF FNV(X)=INT(5*RND+1)
150 DEF FNC(X)=INT(21*RND+1)
```

Now we open the loop to generate the words.

```
160 FOR I=1 TO 20
```

We can use the DEF functions to generate integers, which can in turn be used in the SEG$ function to pick out the desired letters from the strings A$ and B$.

```
170 LET C$=SEG$(B$,FNC(X),1)
180 LET C$=C$&SEG$(A$,FNV(X)
,1)
190 LET C$=C$&SEG$(B$,FNC(X)
,1)
200 LET C$=C$&SEG$(A$,FNV(X)
,1)
210 LET C$=C$&SEG$(B$,FNC(X)
,1)
```

In line 170 the first consonant is generated. A vowel, a consonant, and a vowel are added in lines 180, 190, and 200. Finally the last consonant is appended in line 210.

The balance of the program follows without difficulty.

```
220 PRINT C$,
230 NEXT I
240 END
```

The complete program follows.

```
100 RANDOMIZE
110 LET A$="AEIOU"
120 LET B$="BCDFGHJKLMNPQRST
VWXYZ"
```

```
130 LET X=1
140 DEF FNV(X)=INT(5*RND+1)
150 DEF FNC(X)=INT(21*RND+1)
160 FOR I=1 TO 20
170 LET C$=SEG$(B$,FNC(X),1)
180 LET C$=C$&SEG$(A$,FNV(X)
,1)
190 LET C$=C$&SEG$(B$,FNC(X)
,1)
200 LET C$=C$&SEG$(A$,FNV(X)
,1)
210 LET C$=C$&SEG$(B$,FNC(X)
,1)
220 PRINT C$,
230 NEXT I
240 END
```

RUN the program a few times and see if your favorite brand names turn up!

10-5 PROBLEMS

1. Write a program to generate and print out twenty-five random numbers of the form X.Y where X and Y are digits selected randomly from the set 0, 1, 2, ..., 9.

2. Write a program to generate and print out fifty integers selected at random from the range 13 to 25.

3. What will be printed out if the following program is executed?

```
100 RANDOMIZE
110 FOR N=1 TO 20
120 PRINT INT(20*RND+1)/100
130 NEXT N
140 END
```

4. If the following program is executed, what will be printed out?

```
100 RANDOMIZE
110 FOR I=1 TO 10
120 PRINT INT(100*RND)/10
130 NEXT I
140 END
```

5. Write a program that will simulate tossing a coin 10, 50, 100, 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. For each toss, print out the dice faces that are uppermost.

7. Write a program to generate and print out the average of 1000 random numbers selected from the range 0 to 1. What should this average be?

8. Modify the program of Example 3 and execute it as many times as needed to find the size of crowd such that there is a 50% 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 and 2 P.M. They further agree that they will wait 10 minutes after arriving (but not after 2 P.M.), and if the other person has not arrived, will leave. Write a BASIC program to compute the probability that John and Bill will meet one another. Do a simulation of the problem using the random-number generator.

10. Suppose a bucket 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. The printout should be the colors of the balls drawn in sequence.

11. Use the rule given in the discussion section in this chapter to generate and print out twenty-five 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.

10-6 PRACTICE TEST

Take the following test to see how you are progressing. The answers are given at the end of the book.

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 we execute the following program?

```
100 RANDOMIZE
110 FOR I=1 TO 10
120 LET N=INT(2*RND+1)
130 IF N=1 THEN 160
140 PRINT "WHITE"
150 GOTO 170
160 PRINT "RED"
170 NEXT I
180 END
```

4. What will be printed out if we execute the following program?

```
100 RANDOMIZE
110 FOR J=1 TO 5
120 PRINT INT(1000*RND)/100
130 NEXT J
140 END
```

CHAPTER

ELEVEN

SUBPROGRAMS

11-1 OBJECTIVES

The TI Home Computer has been designed with the capacity to utilize subprograms. These subprograms are not written in BASIC but can be called from BASIC programs as well as in the immediate mode we studied in Chapter 2. The subprograms are either in the computer itself or contained in plug-in cartridges. In this chapter we shall deal with the subprograms that are part of the computer. However, should you ever want to use plug-in subprograms, they are handled in the same way.

Character Manipulation

The computer contains five subprograms dealing with characters. With these subprograms characters can be drawn horizontally and vertically on the screen, the computer can read which character is at a position on the screen, and new characters can be designed.

Sound Generation

With this feature, up to three tones can be generated at a time. Under control of a BASIC program the computer can produce a wide range of audio effects.

Color Control

Through the color subprogram, the computer has access to a "palette" containing sixteen colors. Rich screen displays can be produced using these colors.

Keyboard Interrogation

Often it is useful for the computer to detect what has happened on the keyboard. This is particularly important in teaching or tutorial programs. A subprogram has been provided to carry out keyboard interrogation.

11-2 DISCOVERY ACTIVITIES

In this material we will refer to the ASCII character set. While going through the discovery material you should have available the complete ASCII description contained in the TI Home Computer reference manual.

Now let's go on to the discovery material.

1. Turn your computer on, enter BASIC, and type in the following program:

```
100 INPUT "R = ":R
110 INPUT "C = ":C
120 INPUT "N = ":N
130 INPUT "M = ":M
140 CALL CLEAR
150 CALL HCHAR(R,C,N,M)
160 END
```

RUN this program and enter 10, 10, 72, and 5 for R, C, N, and M respectively. What happened?

2. RUN the program three times, setting R equal to 10, 15, and 20. Keep the other inputs the same as in step 1. Which side of the display does R seem to be measured from?

3. All right, now let R stay at 10, but RUN the program with C equal to 10, 15, and 20. Keep the values of N and M equal to 72 and 5. Which side of the display is C measured from?

4. Now that we have seen how R and C are handled in the CALL HCHAR, let's turn our attention to the part played by N. Keep R, C, and M equal to 10, 10, and 5, but RUN the program using values of N in the range 48 to 90. What does N control?

____

5. Now set R, C, and N to 10, 10, and 72 respectively. RUN the program with M equal to 10, 20, and 50. What does M control?

____

What does the H in CALL HCHAR refer to?

____

6. Now that we have explored the HCHAR subprogram, we will turn to a new function. Change line 150 to read

```
150 CALL VCHAR(R,C,N,M)
```

The VCHAR subprogram should be much easier to understand now that you have had experience with HCHAR. RUN the program several times changing the values of R, C, N, and M. What does R control?

____

What does C control?

____

Changing N changes what?

What is the purpose of M in the VCHAR expression?

What does the V in VCHAR signify?

7. Clear the program from memory and type in the following:

```
100 CALL CLEAR
110 CALL HCHAR(5,5,65)
120 CALL HCHAR(6,6,66)
130 CALL HCHAR(7,7,67)
140 CALL HCHAR(8,8,68)
150 INPUT "R = ":R
160 INPUT "C = ":C
170 LET C = C-2
180 CALL GCHAR(R,C,N)
190 PRINT "CHARACTER AT THAT
DISPLAY"
200 PRINT "LOCATION IS ";CHR
$(N)
210 END
```

Study this program a few moments. The new idea is in the GCHAR subprogram in line 180. RUN the program and enter 6 for both R and C. What happened?

8. OK, if you RUN the program and enter 8 for R and C, what will happen?

____

Try it and record below what took place.

____

9. Now RUN the program and enter 10 for R and 15 for C. What happened?

____

What is on the screen at R = 10, and C = 15?

____

10. By now you should see what GCHAR does. In particular, in the expression GCHAR(R,C,N), what do R and C refer to?

____

What part does N play?

____

What does the G refer to?

____

If you could answer the questions above, fine. If not, don't worry as we will go back over the concepts in the discussion material.

11. Clear out the program from memory and let's go on to a new subprogram. Enter the following program:

```
100 CALL CLEAR
110 INPUT "TYPE IN STRING ":
A$
120 CALL CHAR(96,A$)
130 CALL HCHAR(15,15,96)
140 END
```

The CALL CLEAR in line 100 is familiar since we have been using the command throughout the book. The string input in line 110 is used in the new subprogram CHAR in line 120. Also, the 96 appearing in the CHAR statement is used in the HCHAR subprogram in line 130. RUN the program and at the input prompt, type in the string "FF83858991A1C1FF" and press the ENTER key. What happened?

Is the character at the center of the screen part of the ASCII set?

12. All right, RUN the program again, and this time type in the sixteen characters 30468991523C1010 (the 2nd, 14th, and 16th characters are zeros), and press ENTER. What happened?

You should see a greek letter (not one of the ASCII character set) on the display. What does the CHAR subprogram do?

13. Just one more time. RUN the program and this time use the string 0F03050810204080. In this string, the character used most often is a zero, not the letter O. What happened?

14. Now list the program and study it briefly. Clearly the string A$ which we type in controls the new characters in the CHAR function. What part is played by the 96 used in the CHAR and HCHAR statements?

For now you must be content to see that we can generate new characters. Later, all the loose ends will be tied up and you will learn how to design any characters you want.

15. Now on to a new topic. Clear out the program in memory and type in the one below:

```
100 CALL CLEAR
110 INPUT "DURATION ":D
120 INPUT "TONE ":T
130 INPUT "LOUDNESS ":L
140 CALL SOUND(D,T,L)
150 END
```

It should be clear that this subprogram produces sound. Make sure the volume control on your TV display is turned up before going further. Now RUN the program. Set DURATION equal to 1000. Set the TONE equal to 264 and LOUDNESS equal to zero. What happened?

16. Now RUN the program several times leaving DURATION and TONE at 1000 and 264, but change LOUDNESS to 5, 10, 15, and 20. As the loudness number increases, what happens to the loudness?

This may be a bit confusing to you now. The issue will be cleared up later, so relax!

17. Let's RUN the program again several times. This time, let DURATION be 1000, LOUDNESS be zero, but set TONE to the values; 264, 297, 330, 352, 396, 440, 495, and 528. What does TONE control?

As the number assigned to TONE increases, what happens to the pitch of the sound?

18. Now let TONE and LOUDNESS remain at 264 and 0 respectively but change DURATION. RUN the program with DURATION equal to 4000, 2000, 1000, 500, and 250. What does DURATION control?

As the number assigned to DURATION gets smaller, what happens to the sound?

19. We will try one more wrinkle before leaving the SOUND program. Clear out the program in memory. We will do this work in the immediate mode rather than using a program. Type the following:

```
CALL SOUND(100,264,0)
```

What happened?

There should be no surprises here since we have just been exploring similar issues.

20. Type in the following command:

```
CALL SOUND(1000,264,0,330,0)
```

What did the computer do?

---

21. Try each of the following CALL statements. You should be able to predict what is going to happen for each statement.

```
CALL SOUND(100,264,0,330,0,3
96,0)
CALL SOUND(1000,264,0,352,0,
440,0)
CALL SOUND(1000,264,0,528,0)
```

22. Now we will go on to a new subprogram. Type in the program below:

```
100 CALL CLEAR
110 LET A$="ABCDEFGHIJKLMNOP
QRSTUVWXYZ"
120 PRINT A$
130 LET C=16
140 FOR SET=5 TO 6
150 FOR HUE=2 TO 15
160 CALL COLOR(SET,HUE,C)
170 FOR DELAY=1 TO 100
180 LET X=1
190 NEXT DELAY
200 NEXT HUE
210 NEXT SET
220 END
```

Don't be detracted by the DELAY loop in lines 170, 180, and 190. Its purpose is to provide a delay in the program. What happened?

---

23. Now change line 160 to read

```
160 CALL COLOR(SET,HUE,HUE)
```

RUN the program and describe below what happened.

____

At this point we must let it go with the fact that the color can be changed on the screen. Since the full explanation of how the COLOR subprogram works is rather involved, we won't attempt to explore it any more here. It will be covered in detail in the discussion and examples to follow.

24. This concludes the discovery material. Turn off the computer and go on to the next section.

11-3 DISCUSSION

After seeing what can be done with subprograms, you should have a newfound respect for your TI Home Computer! Now it is important to go back over all the subprograms and concentrate on the details.

Character Manipulation

Before getting started in a discussion of the character subprograms, we should review the ASCII character set briefly. The complete set is described in your computer reference manual. We need to examine only certain parts. First, there are 128 characters in the set. We can use the CHR$(N) function to convert from the character number (N) to the character itself. Many of the characters in the ASCII set have no importance to our discussion. Our specific interest is in the character numbers from 32 through 95.

Character 32 is the space. This is important since when you type CALL CLEAR, it instructs the computer to fill the screen with character number 32, i.e., to fill the screen with blank space. We will come back to this point later.

Characters in the range 33-47, 58-64, and 91-95 are used in punctuation and mathematical notation. You should check these out in the ASCII table in the computer reference manual. The numerals 0-9 have character numbers 48 through 57 in the ASCII set. The upper-case letters A-Z are characters 65 through 90. The lower case letters a-z have ASCII numbers 97 through 123. However, these lower case letters are not available on the TI Home Computer. The character numbers 96 and above have been reserved for a different purpose. Now let's see how new characters or symbols can be defined. All characters on the screen are formed using a dot pattern having eight rows and eight columns. Each position in each row is either turned on (a dot) or left blank. The resultant pattern of dots generates the character.

In the normal character display, only the center six by six array of dots is used to form a character. The blank set of dots around the outside of the array provides horizontal and vertical separation between the characters. There is color involved in the generation of characters on the display. We will see how this is controlled later.

An example of character definition by a dot pattern is shown below. The Xs represent dots and the Os represent spaces.

```
0 0 0 X X 0 0 0
0 0 0 X X 0 0 0
0 0 0 X X 0 0 0
X X X X X X X X
X X X X X X X X
0 0 0 X X 0 0 0
0 0 0 X X 0 0 0
0 0 0 X X 0 0 0
```

This pattern of Xs and Os uses the whole eight by eight array and defines a plus sign, but we could clearly draw anything desired using the total of 64 positions. The dot pattern is communicated to the computer four spaces at a time, two groups of four spaces to a row. Thus in row 1 above, the two 4 space groups are 000X and X000.

A single character is used to describe a block of four spaces or dots. This is done with the following code:

CHARACTER	DOTS	CHARACTER	DOTS
0	OOOO	8	XOOO
1	OOOX	9	XOOX
2	OOXO	A	XOXO
3	OOXX	B	XOXX
4	OXOO	C	XXOO
5	OXOX	D	XXOX
6	OXXO	E	XXXO
7	OXXX	F	XXXX

Thus our dot pattern 000X and X000 would be represented by the characters 8 and 1. Since two characters define the dot pattern in a row of the dot array and there are eight rows, sixteen characters define a complete dot pattern. The plus sign above is defined by the string 818181FFFF818181.

To define your own symbols it is wise to use graph paper with squares already ruled. Set up an 8 by 8 array of squares and then darken squares as desired to generate the new character. Once this has been done, write down the characters that define the dot pattern at the left and right ends of each row in the pattern. Now the necessary string of sixteen characters that define the new symbol can be read easily.

As an example, let's design a symbol to represent the lower case Greek letter lambda. The dot pattern and characters to define the letter are shown below.

```
E   X X X 0 0 0 0 0   0
3   0 0 X X 0 0 0 0   0
1   0 0 0 X X 0 0 0   8
0   0 0 0 0 X X 0 0   C
1   0 0 0 X X X X 0   E
3   0 0 X X 0 0 X X   3
6   0 X X 0 0 0 0 X   1
C   X X 0 0 0 0 0 X   1
```

The character string to define lambda is E030180C1E3361C1.

We have to let the computer know about our new symbol. If we decided to assign our new symbol (lambda) to character number 96, it would be done with CALL CHAR(96,A$) where A$ = "E030180C1E3361C1". Thereafter if we refer to character number 96, we will get our new character. We could go on to define new characters numbered 97, 98, 99, and so on.

Define new characters with the CALL CHAR subprogram.

Before leaving this subprogram several important points need to be made. It is possible, though probably not wise, to redefine the characters 95 and below. Suppose you redefined character 32 to some new symbol and subsequently encountered a CALL CLEAR statement in the same program. Of course, since CALL CLEAR fills the screen with character number 32, your screen is going to be filled with the new symbol rather than being cleared as expected. Moreover, after program termination or break, all character definitions for characters 95 and below revert to their original specification. Consequently if after seeing a screen full of "wierd things," you break the program and type CALL CLEAR, this time you will get the expected clear screen.

The point is that redefinition of the character set below number 96 can produce strange and unpredictable results. Accordingly, it will be prudent to define new characters with numbers 96 and above. There are two reasons for this. First, you don't disturb the character set that is normally used. Second, character definition for characters 96 and above is not lost on program break or termination. They are lost when the computer is turned off or whenever NEW is typed. If desired, whole new character sets can be designed using the CALL CHAR subprogram.

Three subprograms having to do with character handling remain to be discussed. The first of these, the CALL GCHAR subprogram can be disposed of easily. The purpose of this subprogram is to identify a character on the screen. An example is

```
CALL GCHAR(10,12,N)
```

When executed, this causes the ASCII number of the character located 10 rows from the top of the screen and 12 columns over from the left to be assigned to the variable N. If the letter A were at this position, N would have the value 65 (the ASCII character number of A). You may use any numeric name you want in the argument of the CALL GCHAR statement. Thus,

```
150 CALL GCHAR(Y,X,C)

160 CALL GCHAR(ROW,COL,M)
```

represent valid statements. Remember that the three numeric variables used in the argument of the GCHAR subprogram have specific meanings. The number stored in the first of these numerical variables gives the number of rows to move from the top of the screen. The second variable defines the number of columns to move over from the left of the screen. These two numbers (coordinates) locate a position on the screen. The ASCII number of the character at that position is assigned to the third numeric variable in the argument of the GCHAR function.

Read the screen with the CALL GCHAR subprogram.

Sometimes, TV displays "clip" columns from the left and right edges of the screen. Accordingly, columns 1, 2, 31, and 32 may not show on the screen.

Horizontal lines of characters can be drawn on the display with the CALL HCHAR subprogram. A typical statement is

```
100 CALL HCHAR(Y,X,N,R)
```

This statement tells the computer to start a horizontal line Y rows down from the top of the display and X columns over from the left of the display. N defines the ASCII number of the character to form the line. Note that this can also include the special characters defined with the CALL CHAR subprogram that have numbers 96 and above. Whatever character is used, it is repeated R times starting at the X and Y position. If R is so large that the line runs off the right side of the display, it is finished coming in from the left side of the display on the line below. It is not necessary to include R in the argument. For example,

```
100 CALL HCHAR(Y,X,N)
```

is OK. Since there is no repetition number specified, the computer prints a single character (defined by N) at display position X and Y.

Vertical lines are handled exactly the same way. The statement

```
100 CALL VCHAR(Y,X,N,R)
```

causes a vertical line of characters defined by the ASCII equivalent of N to start at display position X and Y. However, this line is drawn down the screen with the character repeated R times. If the line runs off the botton of the screen, it is completed from the top of the display, one line to the right.

Draw horizontal and vertical lines of characters on the display with the CALL HCHAR and CALL VCHAR subprograms.

Sound Generation

Tones (heard over the TV display) can be generated with the CALL SOUND subprogram. A typical statement is

```
100 CALL SOUND(D,P,A)
```

The D in the argument of this function defines the duration of the sound in milliseconds. Since there are 1000 milliseconds in a second, setting D equal to 1000 would specify a tone duration of one second. D can be as large as 4275 (a little over four and a quarter seconds) or as small as 1. If you use CALL SOUND in a program with D = 1 (1 thousandth of a second) the sound is easily heard, and is obviously longer than one thousandth of a second. This is because the duration specification of from 1 to 4275 refers to the sound chip in the computer that generates tones. However program characteristics, audio circuits, and speakers all combine to produce audible tones somewhat different in duration than that specified. Below a duration of D = 100, the effect is particularly pronounced.

Once a CALL SOUND statement is met in a program, the computer turns on the sound generator and then goes on to other statements in the program. Suppose another CALL SOUND statement is encountered while the original sound is still being generated? What happens depends on the algebraic sign of the duration term in the second CALL SOUND statement. If the duration is positive, the computer waits until the first sound is finished, then generates the second sound. If the argument is negative, the first sound is terminated immediately and the second sound is turned on.

Again, returning to the argument of the SOUND subprogram, the second number gives the pitch (or frequency) of the sound in cycles per second. P can be set as low as 110 and goes well past the audible limit (about 20,000 cycles per second) on the high end. To give you some feeling for reasonable values for frequencies, middle C on the piano has a frequency of 264 cycles per second. C an octave above is 528 cycles per second, and C an octave below middle C has a frequency of 132 cycles per second. Thus, frequencies in the range 132 to 528 are in the middle part of the piano keyboard.

The last number in the argument of the SOUND function controls the loudness of the sound. Actually, this number gives the attenuation in db (decibels) that the computer is to apply to the tone. If A is 0, there is no attenuation, and the result is the loudest sound. An A value of 30 corresponds to an attenuation of 30 db which results in the quietest sound. Any attenuation between 0 and 30 db can be specified.

Up to three tones can be handled at the same time. The format is

```
100 CALL SOUND(D,P1,A1,P2,A2
,P3,A3)
```

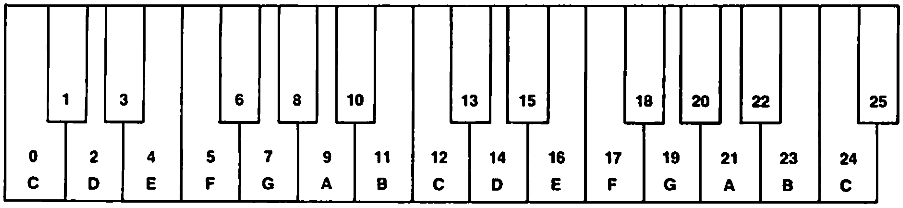
The same duration D applies to all three tones. The first tone has pitch P1 and attenuation A1. The second tone is described by P2 and A2, the third by P3 and A3. If only two tones were desired, P3 and A3 would be deleted from the SOUND specification.

Generate tones on the display audio system with the CALL SOUND subprogram.

Special noise effects are generated by setting the frequency (pitch) to a negative number between -1 and -8. See your computer reference manual for details.

Before leaving the CALL SOUND subprogram, a bit of musical theory is in order if you are to be able to generate music. The musical scale commonly used in western civilizations has twelve notes. The development of this scale took a long time and generated a great deal of controversy which was more or less resolved by Johann Sebastian Bach in his "The Well-Tempered Clavier." In these twenty-four pieces, Bach demonstrated the value of a "tempered" scale, i.e., the twelve note scale we use today.

At any rate, it is possible to generate this scale mathematically. We will do this using the section of a piano keyboard shown below.



$$\text{freq} = f_0(2^{n/12}) \quad \text{read n from above.}$$
$$= f_0(2^{1/12})^n$$
$$= f_0(1.059463094)^n$$

In this diagram, everything is referenced to the left side which is middle C having a frequency of 264 cycles per second. The keys are numbered starting with 0 (middle C), number 12 (C an octave above), and so on. The major scale (white notes on the piano) would be keys numbered 0, 2, 4, 5, 7, 9, 11, and 12. This is the familiar do, re, mi, fa, sol, la, ti, do' scale. The chromatic scale includes black and white keys and would be keys 0, 1, ..., 11, and 12.

The point of this discussion is that we can specify tones by key number and let the computer worry about generating the right frequency. For key N, the correct frequency is

```
NOTE=(FREQ)*(1.059463094)^N
```

Normally we would set the base frequency to middle C (264 cycles per second). This is not necessary, however, and a piece of music played on the computer can be shifted to a different key by merely changing the base frequency. The number 1.059463094 is the twelfth root of 2. So, if we want the frequency of note 12, the twelfth root of two raised to the twelfth power is two, and two times the base frequency gives the frequency of the note an octave above the fundamental (or base) note.

If N takes on the key values of the major scale, we can compute the following information:

Key	Name	N	Freq
C	do	0	264.00
D	re	2	296.33
E	me	4	333.62
F	fa	5	352.40
G	sol	7	395.55
A	la	9	443.99
B	ti	11	499.37
C	do'	12	528.00

It is not necessary to use this information. It is presented here only to demonstrate how the formula works that defines the frequencies of the major scale. On the computer we will use piano keyboard numbers and let the computer worry about generating the right frequencies. This will be demonstrated in the examples.

In passing, it should be pointed out that there are many musical scales including an oriental scale with four notes, an eight-note middle-east scale, and variations on the western twelve-note scale. The TI Home Computer provides a powerful and flexible way to explore music from different cultures. Texts on music theory contain the information needed to set up different scales on the computer.

Color Control

Color is controlled with the COLOR subprogram. A sample call statement is

```
100 CALL COLOR(A,B,C)
```

The arguments of the COLOR function can be named using any numeric name desired. Here we have chosen the names A, B, and C. The first argument (A) defines the character set that is to be controlled. We will go into this later. B specifies the color of the dots which form the character, and C specifies the color of the background of the character.

The computer can generate sixteen different colors which are called out by number. The colors available and their numbers are listed in the following table:

COLOR	NUMBER	COLOR	NUMBER
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

The character set in the CALL COLOR is a subset of the complete ASCII character set plus additional defined characters. Each of these subsets is identified by number. The table below shows how this is done.

SUBSET	N	SUBSET	N
1	32-39	9	96-103
2	40-47	10	104-111
3	48-55	11	112-119
4	56-63	12	120-127
5	64-71	13	128-135
6	72-79	14	136-143
7	80-87	15	144-151

Before discussing how to change the colors of subsets, let's review what takes place on any TV display. The entire picture is refreshed (or redrawn if you will) thirty times each second. If the letter Z is on the screen, for example, it must be redrawn thirty times each second. The information about how to draw the Z must come from somewhere. For a normal TV broadcast, such information comes from the TV station via radio waves. However, the computer display is done differently. Stored in memory are the "dot plans" to construct each character. While studying the CALL CHAR subprogram we saw that the dot plan for a symbol or character is a string sixteen characters long. Additional information is required to tell the computer what the color is to be. All this information is read from memory thirty times each second to maintain the screen display. If you change the instructions to generate characters then immediately the display appearance of all the characters involved will change.

Notice that each of the subsets contains eight characters. You can refer to the ASCII table in the computer reference manual to see exactly which characters are in each of the subsets. We will point out here where some of the characters lie. The upper case letters A–G are in subset 5, H–O are in subset 6, P–W are in number 7, and X–Z are in subset 8. The numerals 0–9 are in subsets 3 and 4.

Now we can see how the COLOR subprogram works. The statement

```
100 CALL COLOR(5,9,12)
```

changes the instructions for generating the letters A, B, C, D, E, F, and G (subset number 5). The dots generating the letters will switch to medium red (color number 9), and the background for each letter will switch to light yellow (color number 12). When this statement is executed by the computer, all the characters A–G on the screen will immediately switch to the new colors.

If we wanted to change the colors of the complete alphabet and the numerals, we would have to write CALL COLOR statements for character subsets 3 through 8. Note that subset 9 begins with character number 96 where we can define new characters with the CALL CHAR subprogram.

Control character color with the CALL COLOR subprogram.
Control the screen color with the CALL SCREEN subprogram.

One final point about the color of the display needs to be made. As pointed out above, the character colors are controlled by setting the foreground (the character) and the background (the part of the eight by eight array not involved in the character) colors. However, both these "layers" of color are over a third layer, the "bottom" color of the screen. The point is that this bottom color can be controlled with the CALL SCREEN subprogram. An example might be

```
280 CALL SCREEN(N)
```

where N contains a number between 1 and 16 inclusive. Of course, these sixteen numbers are the color numbers already referred to. Very interesting displays can be generated. For example, if the bottom screen color is set to light blue (color 6), the background of the character set to transparent (color 1), the character color set to white (color 16), then the result would be white characters on a light blue screen. Using the CALL CHAR, CALL COLOR, and CALL SCREEN subprograms, many interesting and colorful screen displays can be produced.

Keyboard Interrogation

This topic is rather specialized and was not covered in the discovery material. However, there are cases where one needs to interrogate the keyboard under program control. This is done with the CALL KEY subprogram. A typical statement is

```
100 CALL KEY(0,N,S)
```

The first argument is set equal to zero. This means that the keyboard is to be interrogated. Other codes are used for other devices. See the computer reference manual for details.

The last argument (S) reflects the status of the keyboard. If a new key has been pressed since the last time the CALL KEY statement was executed, S has the value +1. If the same key is down as the last time the CALL statement was executed, S is -1. Finally, if no key is down, S has the value 0. Thus, by looking at the third argument (any numeric name can be used) we can determine what is happening on the keyboard.

If the status indicator is either +1 or -1, the second argument (N) contains the character number of the key down at the instant the CALL statement is executed. This can be translated to the character with the CHR$(N) function.

The CALL KEY subprogram enables you to see if a key is down or not, and if so, which one is down. Since this can be done within a BASIC program, a new dimension has been added to the capabilities of the computer.

11-4 PROGRAM EXAMPLES

Now we will look at some programs which take advantage of the powerful subprograms already discussed.

Example 1 - "Frere Jacques"

To show off the musical ability of the TI Home Computer we will write a program to play a three part round. Since the melody to "Frere Jacques" is familiar to many, we will select this tune.

We will use arrays to keep track of the scale and the notes to be played. So, we must declare an option base, and dimension the arrays.

```
100 OPTION BASE 0
110 DIM S(26),K(65,3)
```

Our strategy will be to set up the scale in the array S which will contain twenty-six frequencies corresponding to the keyboard diagram in the discussion section. The array K will contain which keys are supposed to be "down" at any given time. There are 65 rows in K but we will ignore row zero. Each row in K corresponds to an eighth note in the music. If we want a quarter note, we must repeat an eighth note twice. A half note is obtained by four eighth notes, and so on. The song "Frere Jacques" requires 64 eighth notes to generate the half, quarter, and eighth notes in the music.

First we generate the scale.

```
120 LET FREQ=264
130 FOR N=0 TO 25
140 LET S(N)=FREQ*1.05946309
4^N
150 NEXT N
```

All the frequencies are computed with respect to the base frequency of 264 cycles per second (middle C). We will use the subscript on array S to correspond to the key numbers on the keyboard diagram. Key 0 will refer to S(0) which contains 264. Key 12 points to S(12) which contains 528, and so forth.

Now we set up a loop to call for the input of the three key numbers for each of the 64 eighth notes which make up the music.

```
160 FOR R=1 TO 64
170 PRINT R,
180 INPUT K(R,1)K(R,2),K(R,3
)
190 NEXT R
```

We have printed out the row number R to indicate which three numbers are to be typed in. The data to enter will be given later.

Now we can play the music.

```
200 LET D=250
210 FOR R=1 TO 64
```

```
220 CALL SOUND(D,S(K(R,1)),0
,S(K(R,2)),0,S(K(R,3)),0)
230 NEXT R
240 GOTO 210
250 END
```

In line 200 we set the duration of the eighth notes to 250 milliseconds (a quarter of a second). Then we loop through the key number array K picking up three key numbers at a time which are then used in the scale array S to get the needed frequencies for the CALL SOUND statement.

The complete program follows:

```
100 OPTION BASE 0
110 DIM S(25),K(65,3)
120 LET FREQ=264
130 FOR N=0 TO 25
140 LET S(N)=FREQ*1.05946309
4^N
150 NEXT N
160 FOR R=1 TO 64
170 PRINT R
180 INPUT K(R,1),K(R,2),K(R,
3)
190 NEXT R
200 LET D=250
210 FOR R=1 TO 64
220 CALL SOUND(D,S(K(R,1)),0
,S(K(R,2)),0,S(K(R,3)),0)
230 NEXT R
240 GOTO 210
250 END
```

In this round, we have started all three "voices" at the same time, each at its proper place in the melody. It may take some time to type in the key array, but the results are worth it!

We still need the key numbers to define the music. The table below gives this information. When you RUN the program, type in the information in this table, three numbers at a time.

R	Keys	R	Keys
1	12,12,19	33	19,16,12
2	12,12,21	34	21,16,12
3	14, 7,19	35	19,17,14
4	14, 7,17	36	17,17,14

```
5        16,12,16       37       16,19,16
6        16,12,16       38       16,19,16
7        12,12,12       39       12,19,12
8        12,12,12       40       12,19,12
9        12,12,19       41       19,16,12
10       12,12,21       42       21,16,12
11       14, 7,19       43       19,17,14
12       14, 7,17       44       17,17,14
13       16,12,16       45       16,19,16
14       16,12,16       46       16,19,16
15       12,12,12       47       12,19,12
16       12,12,12       48       12,19,12
17       16,12,12       49       12,19,16
18       16,12,12       50       12,21,16
19       17,14, 7       51        7,19,17
20       17,14, 7       52        7,17,17
21       19,16,12       53       12,16,19
22       19,16,12       54       12,16,19
23       19,12,12       55       12,12,19
24       19,12,12       56       12,12,19
25       16,12,12       57       12,19,16
26       16,12,12       58       12,21,16
27       17,14, 7       59        7,19,17
28       17,14, 7       60        7,17,17
29       19,16,12       61       12,16,19
30       19,16,12       62       12,16,19
31       19,12,12       63       12,12,19
32       19,12,12       64       12,12,19
```

Example 2 - Colored Character Sets

In this example we shall simply present a program that displays subsets of the ASCII character set in various colors. Type any key and press ENTER to get out of this program. The program is

```
100 CALL CLEAR
110 FOR C=1 TO 24
120 CALL HCHAR(C,3,64+C,28)
130 NEXT C
140 LET HUE=10
150 FOR SET=5 TO 8
160 CALL COLOR(SET,HUE,16)
170 LET HUE=HUE + 1
180 NEXT SET
190 INPUT A$
200 END
```

Example 3 - Graphic Characters

As the final example, we will present a program to draw a grid on the screen with solid colored lines. Run the program to see what happens. Then analyze the program in detail to see what each statement does.

```
100 CALL CLEAR
110 LET A$ = "FFFFFFFFFFFFFFFFF"
120 CALL CHAR(96,A$)
130 LET Y=6
140 FOR X=6 TO 22 STEF 4
150 CALL VCHAR(Y,X,96,16)
160 NEXT X
170 LET X=6
180 FOR Y=6 TO 22 STEF 4
190 CALL HCHAR(Y,X,96,17)
200 NEXT Y
210 FOR HUE=1 TO 16
220 CALL COLOR(9,HUE,HUE)
230 FOR DELAY=1 TO 100
240 REM DO NOTHING FOR DELAY
250 NEXT DELAY
260 NEXT HUE
270 END
```

11-5 PROBLEMS

1. Write a program to play the major scale beginning at middle C.

2. Write a program to play a song of your choice.

3. What will happen if the following program is RUN?

```
100 LET A$="8080B0C88484C8B0"
110 CALL  CHAR(96,A$)
120 CALL CLEAR
130 CALL HCHAR(5,12,96,10)
140 END
```

4. Write a program to print "RED LETTERS" on the screen using red dots on a white background.

5. Design the lower case letters a, b, c, d, e, and f. Then use CALL CHAR to load the designs into the computer. Write a program to display these characters on the screen.

6. Design a character with every other dot turned on in blue with a transparent background over a white bottom color. Use this character to fill every other row on the screen.

7. Write a program to fill the screen with green H characters.

11-6 PRACTICE TEST

See how well you have learned the material in this chapter by taking this practice test. The answers are given at the end of the book.

1. What does CALL SCREEN(11) DO?

____

2. In the statement CALL HCHAR(Y,X,N,R), explain what each of the arguments does.

____

3. What is the purpose of the CALL GCHAR subprogram?

____

4. Explain what CALL COLOR(6,11,16) will do.

____

5. What is the purpose of the CALL KEY subprogram?

____

6. Explain precisely what CALL CLEAR accomplishes.

7. Explain what each of the arguments in CALL SOUND(L,F,X) controls.

PRACTICE TEST SOLUTIONS

Chapter 2

1. Press the ENTER key.
2. Press the shift-Q key. Or, you can turn the computer off, then back on.
3. Multiplication is indicated with the * symbol.
4. Type CALL CLEAR and press the ENTER key.
5. The symbol / indicates division.
6. The computer will display the number 2 on the screen.
7. The characters "25/5+2" will be displayed on the screen.
8. Press the shift-S key seven times to move the cursor back over the G. Then type T and press the ENTER key.

Chapter 3

1. Press the ENTER key.
2. Press the shift-C key.
3. Press the shift-C key.
4. The numeral 1 will be displayed on the screen.
5. Up to 15 characters can be used for numeric variables, and up to 14 characters for string variables (the $ must be appended).
6. Type the line number and press the ENTER key.
7. Simply type it in using a line number not already in the program.
8. Just type it again in the form desired.
9. Type LIST and press the ENTER key.

10. Type CALL CLEAR and press the ENTER key.

11. Type NEW and press the ENTER key.

12. Type RUN and press the ENTER key.

13. A numeric variable names a number. A character-string variable names a collection of characters.

Chapter 4

1. The operators are –, *, +, ^ ,and /.

2. First priority is exponentiation. Next is multiplication and division. Finally, the computer does addition and subtraction.

3. Left to right.

4. 100 LET A = (4+3*B/D)^2

5. The number 4.

6. a. 5.673E+14 b. 3.814275168E-06

7. a. 7258000. b. 0.001437

8. /, +, ^ .

9. Type SAVE CS1 and then follow the instructions.

10. Type OLD CS1 and then follow the instructions.

Chapter 5

1. The sequence of numbers below will be displayed on the screen.

```
1         2
3         4
5         6
7         8
   etc.
```

2. a. Assignment with the LET statement, b. INPUT, and c. READ DATA.

3. A string.

4. To insert explanatory remarks into a program.

5. A DATA statement.

6. Y = 3 will be displayed on the screen.

7. Two columns per line.

8. As many as needed.

9. To obtain precise, variable spacing on a line.

10. The following number pattern will be displayed:

```
1          3
1 3
```

11. The computer will detect extra input since it is expecting two numbers and three were typed in. The computer will prompt you to enter the data again.

12.

```
100 PRINT "HOW MANY MILES";
110 INPUT M
120 LET K=1.609*M
130 PRINT M;" MILES IS THE"
140 PRINT "SAME AS ";K;" KM
."
150 END
```

Chapter 6

1. The sequence of numbers 6, 10, 14, and 18 will be displayed on the screen.

2. The messages below will be displayed.

```
BEST

BETTER
BEST
```

```
GOOD
BETTER
BEST

DATA ERROR IN 100
```

3.

```
100 PRINT "HOW MANY WIDGETS"
;
110 INPUT N
120 IF N<=20 THEN 160
130 IF N<=50 THEN 180
140 LET P=1.50
150 GOTO 190
160 LET P=2.00
170 GOTO 190
180 LET P=1.80
190 PRINT "PRICE PER WIDGET
IS ";P
200 LET C=N*P
210 PRINT "TOTAL COST OF ORD
ER IS ";C
220 GOTO 100
230 END
```

4.

```
100 LET NUMBER=0
110 PRINT NUMBER,
120 LET NUMBER=NUMBER+5
130 IF NUMBER<=115 THEN 110
140 END
```

5.

```
100 PRINT "SPEED LIMIT";
110 INPUT LIMIT
120 PRINT "SPEED ARRESTED AT
";
130 INPUT ARRESTED
140 LET X=ARRESTED-LIMIT
150 IF X<=10 THEN 210
160 IF X<=20 THEN 230
170 IF X<=30 THEN 250
180 IF X<=40 THEN 270
190 LET F=80
200 GOTO 280
210 LET F=5
```

```
220 GOTO 280
230 LET F=10
240 GOTO 280
250 LET F=20
260 GOTO 280
270 LET F=40
280 PRINT "FINE IS ";F;" DOL
LARS"
290 END
```

Chapter 7

1.

```
20          18
16          14
12          10
8           6
4           2
```

2. The numbers 1, 2, 3, 2, 4, 6, 3, 6, 9, 4, 8, and 12 will be displayed in a vertical line on the screen.

3. a. 6, b. 7, c. 22.8, and d. -1.

4. The I and J loops are crossed.

5.

```
100 PRINT "MILES","KILOMETER
S"
110 PRINT
120 FOR MILES=10 TO 100 STEP
 5
130 LET KM=1.609*MILES
140 PRINT MILES,KM
150 NEXT MILES
160 END
```

6.

```
100 READ N
110 LET SUM=0
120 FOR COUNT =1 TO N
130 READ X
140 LET SUM=SUM+X
150 NEXT COUNT
160 PRINT SUM/N
```

```
170 DATA 10
180 DATA 25,21,24,21,26,27,2
5,24,23,24
190 END
```

7. a. ABS(X) computes the absolute magnitude of X.

   b. SGN(X) computes the algebraic sign of X. If X is positive, SGN(X) is +1. If X is negative, SGN(X) is -1, and if X is 0, SGN(X) is 0.

   c. INT(X) is the first integer less than X.

   d. SQR(X) computes the square root of X. X cannot be negative.

   e. SEG$ is used to pick out a segment of a string.

   f. VAL is used to convert a string representation of a number to the numeric form.

Chapter 8

1. The DIM statement is used to reserve space for either numeric or string arrays. The OPTION statement establishes the first subscript of arrays as either 0 or 1.

2. X(3,4)

3. The word "BILL" and the number "183" will be displayed on the same line.

4.

```
100 OPTION BASE 0
110 DIM X(100)
120 PRINT "HOW MANY NUMBERS"
;
130 INPUT N
140 PRINT
150 PRINT " ","NUMBER"
160 PRINT
170 FOR I=1 TO N
190 PRINT I,
200 INPUT X(I)
210 NEXT I
220 LET S=0
230 FOR I=1 TO N
240 IF X(I)<0 THEN 260
250 LET S=S+X(I)
260 NEXT I
270 PRINT "SUM OF POSITIVE"
280 PRINT "NUMBERS IS ";S
290 END
```

5. X$(2,4)

6.

```
 90 OPTION BASE 1
100 FOR ROW=1 TO 4
110 FOR COL=1 TO 6
120 LET X(ROW,COL)=4
130 NEXT COL
140 NEXT ROW
150 FOR ROW=1 TO 4
160 FOR COL=1 TO 6
170 PRINT X(ROW,COL);
180 NEXT COL
190 PRINT
200 NEXT ROW
210 END
```

7.

```
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
```

8. a. DIM A(2,3), b. 4, c. 3, and d. 3.

9. OPEN sets up a communication path between the computer and an external device such as a tape cassette.

10. CLOSE severs the communication path established by an OPEN statement.

Chapter 9

1. a. 4, b. 14, c. 30, and d. 80.

2.

```
2                   1
3
4
```

3. a. GOSUB.

   b. RETURN.

   c. The STOP statement is equivalent to GOTO the END statement.

4.

```
WHITE
RED
BLUE
```

Chapter 10

1.

```
100 RANDOMIZE
110 FOR COUNT=1 TO 100
120 PRINT INT(4*RND+1),
130 NEXT COUNT
140 END
```

2.

```
100 RANDOMIZE
110 FOR COUNT=1 TO 100
120 PRINT 25*RND+25
130 NEXT COUNT
140 END
```

3. The words WHITE and RED will be selected at random and printed ten times.

4. Five random numbers over the range 0.00 to 9.99.

Chapter 11

1. This command will fill the base color of the screen with dark yellow.

2. Y defines the row number measured from the top of the screen. X defines the column number measured from the left of the screen. N defines the character number (from the ASCII set) to be printed. R is the repetition factor.

3. The purpose of the GCHAR subprogram is to read the ASCII number of the character at the row and column number specified on the screen.

4. CALL COLOR(6,11,16) sets the color of character subset number 6. The dots forming the characters will be dark yellow, and the background will be white.

5. CALL KEY is a method whereby the keyboard can be interrogated to see if a key is down, or a key has been depressed since the last execution of a CALL KEY statement.

6. CALL CLEAR fills the screen with ASCII character number 32 (the blank space). This clears the screen.

7. In CALL SOUND(L,F,X), L is the duration of the tone in milliseconds, F is the pitch of the tone in cycles per second, and X is the attenuation of the tone in db.

SOLUTIONS TO ODD-NUMBERED PROBLEMS

Chapter 5

1.

```
100 READ A,B,C,D
110 DATA 10,9,1,2
120 LET S=A+B
130 LET P=C*D
140 PRINT S,P
150 END
```

3. The program will display 17 and 25 on the same line.

5.

```
100 PRINT "TIME OF FALL (SEC
)";
110 INPUT TIME
120 LET DISTANCE=16*T^2
130 PRINT "OBJECT FALLS ";DI
STANCE;" FEET"
140 END
```

7.

```
100 INPUT A,B
110 LET T=B
120 LET B=A
130 LET A=T
140 PRINT A,B
150 END
```

9.

```
100 PRINT "PRINCIPAL";
110 INPUT P
120 PRINT "INT. RATE (%)";
130 INPUT I
140 PRINT "TERM (YEARS)";
150 INPUT N
160 LET T=P*(1+I/100)^N
170 PRINT "TOTAL VALUE IS"
180 PRINT T
190 END
```

Chapter 6

1.

```
100 INPUT A,B
110 IF A>B THEN 140
120 PRINT B
130 GOTO 150
140 PRINT A
150 END
```

3.

```
100 LET SUM=0
110 LET NUMBER=0
120 LET SUM=SUM+NUMBER
130 LET NUMBER=NUMBER+1
140 IF NUMBER<=100 THEN 120
150 PRINT SUM
160 END
```

5. If the program is RUN it will use up all the numbers (including 1111) and after printing DATA ERROR IN 120, will stop. The reason is that the program is looking for a "flag variable" 9999 to mark the end of the data and it isn't present. So, the program runs out of data and stops.

7.

```
100 DATA 4,18,-3,-28,36,8
110 DATA 1,-6,12,9999
120 LET SUM=0
130 READ NUMBER
140 IF NUMBER=9999 THEN 190
150 IF NUMBER<-10 THEN 130
160 IF NUMBER>10 THEN 130
170 LET SUM=SUM+NUMBER
180 GOTO 130
190 PRINT SUM
200 END
```

9.

```
100 INPUT A,B
110 IF A<10 THEN 170
120 IF B<10 THEN 150
130 PRINT A+B
140 GOTO 210
150 PRINT A-B
160 GOTO 210
170 IFB>=10 THEN 200
180 PRINT A*B
190 GOTO 210
200 PRINT B-A
210 END
```

11.

```
100 PRINT "GROWTH RATE (%)";
110 INPUT R
120 LET N=0
130 LET Q=1
140 LET Q=Q*(1+R/100)
150 LET N=N+1
160 IF Q<2 THEN 140
170 PRINT "NUMBER OF GROWTH
PERIODS"
180 PRINT "TO DOUBLE IS ";N
190 END
```

Chapter 7

1.

```
100 PRINT "N","SQR(N)"
110 PRINT
120 FOR N=2 TO 4 STEP .1
130 PRINT N,SQR(N)
140 NEXT N
150 END
```

3.

```
100 INPUT N
110 FOR X=2 TO N STEP 2
120 PRINT X
130 NEXT X
140 END
```

5. The numbers 0, -1, 8, 0, and 0 will be displayed in a vertical column on the screen.

7.

```
100 LET P=1
110 INPUT "FACTORIAL OF ";F
120 FOR LOOP=1 TO F
130 LET P=P*LOOP
140 NEXT LOOP
150 PRINT "THE FACTORIAL OF
";F
160 PRINT "IS ";P
170 END
```

9. The X and Z loops are crossed.

11.

```
100 PRINT "ANNUAL INVESTMENT
";
110 INPUT I
120 PRINT "INTEREST RATE (%)
";
130 INPUT R
140 PRINT "HOW MANY YEARS ";
150 INPUT N
160 LET P1=0
170 FOR COUNT=1 TO N
180 LET P2=(P1+I)*(1+R/100)
190 LET P1=P2
200 NEXT COUNT
210 PRINT "AT THE END OF THE
"
220 PRINT "LAST YEAR, THE VA
LUE"
230 PRINT "OF THE INVESTMENT
"
240 PRINT "WILL BE ";P1
250 END
```

13.

```
100 PRINT "ID","AVE. GRADE"
110 PRINT
120 READ N
130 FOR COUNT=1 TO N
140 READ ID,G1,G2,G3
150 LET AVE=.25*G1+.25*G2+.5
0*G3
160 PRINT ID,AVE
170 NEXT COUNT
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
300 END
```

15.

```
100 FOR X=1 TO 127
110 PRINT CHR$(X);
120 NEXT X
130 END
```

Chapter 8

1.

```
100 DIM X(25)
110 OPTION BASE 1
120 INPUT N
130 FOR I=1 TO N
140 READ X(I)
150 NEXT I
160 FOR I=1 TO N
170 PRINT X(I);
180 NEXT I
200 DATA 12
210 DATA 2,1,4,3,2,4,5,6,3,5
,4,1
220 END
```

3. The number 10 will be displayed on the screen.

5.

```
100 DIM X(100)
110 OPTION BASE 1
120 INPUT N
130 FOR I=1 TO N
140 PRINT I;
150 INPUT X(I)
160 NEXT I
170 FOR I=1 TO N-1
180 IF X(I+1)<=X(I) THEN 230
190 LET TEMP=X(I)
200 LET X(I)=X(I+1)
210 LET X(I+1)=TEMP
220 GOTO 170
230 NEXT I
240 FOR I=1 TO N
250 PRINT X(I)
260 NEXT I
270 END
```

7.

```
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
```

9.

```
100 DIM X(2,5)
110 OPTION BASE 1
120 FOR ROW=1 TO 2
130 FOR COL=1 TO 5
140 READ X(ROW,COL)
150 NEXT COL
160 NEXT ROW
170 DATA 2,1,0,5,1
180 DATA 3,2,1,3,1
190 FOR ROW=1 TO 2
200 FOR COL=1 TO 5
210 PRINT X(ROW,COL);
220 NEXT COL
230 PRINT
240 PRINT
250 NEXT ROW
260 END
```

11.

```
100 DIM A(30,30)
110 OPTION BASE 1
120 PRINT "HOW MANY ROWS ";
130 INPUT R
140 PRINT "HOW MANY COLUMN
S ";
150 INPUT C
160 FOR R1=1 TO R
170 FOR C1=1 TO C
180 PRINT "ROW ";R1;" COL ";
C1;
190 INPUT A(R1,C1)
200 NEXT C1
210 NEXT R1
220 FOR R1=1 TO R
230 LET S=0
240 FOR C1=1 TO C
250 LET S=S+A(R1,C1)
260 NEXT C1
270 PRINT "SUM OF ROW ";R1;"
 IS ";S
280 NEXT R1
290 FOR C1=1 TO C
300 LET P=1
310 FOR R1=1 TO R
320 LET P=P*A(R1,C1)
330 NEXT R1
340 PRINT "PRODUCT OF COLUMN
 ";C1;" IS ";P
350 NEXT C1
360 END
```

13.

```
100 DIM SALES(4,6),DAILYTOTA
L(4),WEEKLYTOTAL(6)
110 OPTION BASE 1
120 FOR SALESPERSON=1 TO 4
130 PRINT "DAILY TOTALS FOR"
140 PRINT "SALESPERSON ";SAL
ESPERSON
150 FOR DAY=1 TO 6
160 PRINT "DAY ";DAY
170 INPUT SALES(SALESPERSON,
DAY)
180 NEXT DAY
190 NEXT SALESPERSON
200 LET TOTAL=0
210 FOR SALESPERSON=1 TO 4
220 LET DAILYTOTAL(SALESPERS
ON)=0
230 FOR DAY=1 TO 6
240 LET DAILYTOTAL(SALESPERS
ON)=DAILYTOTAL(SALESPERSON)+
SALES(SALESPERSON,DAY)
250 LET WEEKLYTOTAL(DAY)=WEE
LYTOTAL(DAY)+SALES(SALESPERS
ON,DAY)
260 LET TOTAL=TOTAL+SALES(SA
LESPERSON,DAY)
270 NEXT DAY
280 NEXT SALESPERSON
290 PRINT
300 PRINT "SALESPERSON ","WE
EKLY TOTAL"
310 FOR SALESPERSON=1 TO 4
320 PRINT SALESPERSON,WEEKLY
TOTAL(SALESPERSON)
330 NEXT SALESPERSON
340 PRINT
350 PRINT "DAY","DAILY TOTAL
"
360 FOR DAY=1TO 6
370 PRINT DAY,DAILYTOTAL(DAY
)
380 NEXT DAY
390 PRINT
400 PRINT "TOTAL SALES FOR W
EEK IS ";TOTAL
410 END
```

15.

```
100 DIM NAMES$(20),GRADE(20)
110 OPTION BASE 1
120 PRINT "HOW MANY NAMES ";
130 INPUT N
140 PRINT
150 PRINT "TYPE IN NAMES AND
GRADES SEPARATED BY A COMMA"
160 FOR I=1 TO N
170 INPUT NAMES$(I),GRADE(I)
180 NEXT I
190 PRINT
200 FOR I=1 TO N-1
210 IF GRADE(I+1)<GRADE(I) T
HEN 290
220 LET TEMP=GRADE(I)
230 LET TEMP$=NAME$(I)
240 LET GRADE(I)=GRADE(I+1)
250 LET GRADE(I+1)=TEMP
260 LET NAME$(I)=NAME$(I+1)
270 LET NAME$(I+1)=TEMP$
280 GOTO 200
290 NEXT I
300 PRINT "GRADE","NAME"
310 PRINT
320 FOR I=1 TO N
330 PRINT GRADE(I),NAME$(I)
340 NEXT I
350 END
```

17.

```
100 DIM NAME$(10)
110 OPTION BASE 1
120 OPEN #1:"CS1",INPUT,FIXED
 64
130 FOR I=1 TO 10
140 INPUT #1:NAME$(I)
150 NEXT I
160 FOR I=1 TO 9
170 IF NAME$(I)<NAME$(I+1) T
HEN 220
180 LET TEMP$=NAME$(I)
190 LET NAME$(I)=NAME$(I+1)
200 LET NAME$(I+1)=TEMP$
210 GOTO 160
220 NEXT I
230 FOR I=1 TO 10
240 PRINT NAME$(I)
250 NEXT I
260 CLOSE #1
270 END
```

Chapter 9

1.

```
25                5
20
65
```

3.

```
2    7    1    3    3
7    10   3    3    3
10   3    3    8    8
```

5.

```
100 REM SUBROUTINE
110 LET T=0
120 FOR I=1 TO Z(0)
130 LET T=T+Z(I)
140 NEXT I
150 RETURN
```

7.

```
100 OPEN #1:"CS1",INPUT,FIXED
 51
110 INPUT #1:TOTAL
120 FOR COUNT=1 TO TOTAL
130 INPUT #1:A$
140 PRINT "ROOM: ";SEG$(A$,2
,15)
150 PRINT "ITEM: ";SEG$(A$,1
7,15)
160 PRINT "BOUGHT: 19";SEG$(
A$,32,2)
170 PRINT "PURCHASED FOR: $"
;SEG$(A$,31,9)
180 PRINT "CURRENT VALUE: $
";SEG$(A$,43,9)
190 PRINT
200 NEXT COUNT
210 END
```

Chapter 10

1.

```
100 RANDOMIZE
110 FOR COUNT=1 TO 25
120 PRINT INT(100*RND)/10
130 NEXT COUNT
140 END
```

3. Twenty numbers selected at random over the range 0.01 to 0.20 will be displayed on the screen.

5.

```
100 RANDOMIZE
110 FOR I=1 TO 5
120 READ N
130 LET HEADS=0
140 LET TAILS=0
150 FOR COUNT=1 TO N
160 LET X=INT(2*RND+1)
170 IF X=1 THEN 200
180 LET TAILS=TAILS+1
190 GOTO 210
200 LET HEADS=HEADS+1
210 NEXT COUNT
220 PRINT
230 PRINT "FOR ";N;" TOSSES
THERE WERE"
240 PRINT HEADS;" HEADS"
250 PRINT TAILS;" TAILS"
260 NEXT I
270 DATA 10,50,100,500,1000
280 END
```

7.

```
100 RANDOMIZE
110 LET SUM=0
120 FOR COUNT=1 TO 1000
130 LET SUM=SUM+RND
140 NEXT COUNT
150 LET AVERAGE=SUM/1000
160 PRINT AVERAGE
170 END
```

9.

```
100 RANDOMIZE
110 LET MEETS=0
120 FOR COUNT=1 TO 1000
130 LET JOHN=60*RND
140 LET BILL=60*RND
150 IF ABS(JOHN-BILL)>10 THE
N 170
160 LET MEETS=MEETS+1
170 NEXT COUNT
180 PRINT "PROB. OF A MEET I
S ";MEETS/1000
190 END
```

11.

```
100 RANDOMIZE
110 FOR LOOP=1TO 25
120 LET SUM=0
130 FOR COUNT=1 TO 12
140 LET SUM=SUM+RND
150 NEXT COUNT
160 LET R=10+2*(SUM-6)
170 PRINT INT(100*R+.5)/100
180 NEXT LOOP
190 END
```

Chapter 11

1.

```
100 DATA 264,296,334,352
110 DATA 396,444,499,528
120 FOR I=1 TO 8
130 READ FREQ
140 CALL SOUND(1000,FREQ,0)
150 NEXT I
160 END
```

3. A string of 10 lower case bs will be printed horizontally beginning 5 rows down from the top and 12 columns over from the left of the screen.

5.

```
100 DATA "00001A264242261A"
110 DATA "4040586442426458"
120 DATA "00001C224040221C"
130 DATA "02021A264242261A"
140 DATA "00001C227E40221C"
150 DATA "0008142470202020"
160 LET WORDS$=""
170 FOR COUNT=96 TO 101
180 READ A$
190 CALL CHAR(COUNT,A$)
200 LET WORD$=WORD$&CHR$(COUNT)
210 NEXT COUNT
220 CALL CLEAR
230 PRINT WORD$
240 END
```

7.

```
100 CALL CLEAR
110 CALL SCREEN(16)
120 CALL COLOR(6,3,1)
130 CALL HCHAR(1,1,72,768)
140 FOR DELAY=1 TO 5000
150 REM DO NOTHING
160 NEXT DELAY
170 END
```

INDEX

Herbert D. Peckham is professor of Natural Science at Gavilan College, Gilroy, California, where he has taught courses in physics, mathematics and computers for the past 16 years. He is active in the American Association of Physics Teachers, having served as chairman of the committee on physics in 2-year colleges and as a member of the executive board of that organization. During his tenure at Gavilan College, he has been a consultant to the Aerospace Electroexplosive Industry. He helped found and was the first president of the Northern California Community College Computing Consortium. Mr. Peckham is the author of numerous books and monographs on computers in education.

LCB-4190

0-07-049156-9