

BASIC for the
APPLE
II

Programming and Applications



LARRY JOEL GOLDSTEIN
and MARTIN GOLDSTEIN

BASIC for THE APPLE II

PROGRAMMING
and APPLICATIONS

Larry Joel Goldstein

**University of Maryland
College Park, MD**

Martin Goldstein

**Goldstein Associates
West Palm Beach,
Florida**

Robert J. Brady Co.
*A Prentice-Hall Publishing and
Communications Company*
Bowie, Maryland 20715

Executive Editor: David T. Culverwell
Production Editor: Michael J. Rogers
Art Director: Bernard Vervin
Photography: George Dodson
Typesetting: Bi-Comp, Inc., York, PA
Typefaces: Aster (display) and Optima (text)
Printed by: R. R. Donnelley & Sons Company, Harrisonburg, VA
Text designer: Michael J. Rogers
Cover design: Don Sellers

Copyright © 1982 by Robert J. Brady Co.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage and retrieval system, without permission in writing from the publisher. For information, address Robert J. Brady Co., Bowie, Maryland 20715.

Library of Congress Cataloging in Publication Data

Goldstein, Larry Joel.

Basic for the Apple II.

Includes index.

1. Apple II (Computer)—Programming.

2. Basic (Computer program language)

I. Goldstein, Martin, 1919 Mar. 28— .

II. Title.

QA76.8.A662G64 1982 001.64'2 82-9418

ISBN 0-89303-190-9

ISBN 0-89303-189-5 (pbk.)

Prentice-Hall International, Inc., London
Prentice-Hall Canada, Inc., Scarborough, Ontario
Prentice-Hall of Australia, Pty., Ltd., Sydney
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Prentice-Hall of Southeast Asia Pte. Ltd., Singapore
Whitehall Books, Limited, Petone, New Zealand

Printed in the United States of America

82 83 84 85 86 87 88 89 90 91 92 10 9 8 7 6 5 4 3 2

CONTENTS

Preface v

1. A First Look at Computers	1
1.1 Introduction	1
1.2 What is a Computer?	4
1.3 Meet Your Apple II Computer	5
2. Getting Started in Applesoft BASIC	11
2.1 Computer Languages and Programs	11
2.2 Elementary Applesoft BASIC Programs	15
2.3 Giving Names to Numbers and Words	25
2.4 Doing Repetitive Operations	32
2.5 Some Applesoft BASIC Commands	42
2.6 Letting Your Computer Make Decisions	47
2.7 Some Programming Tips	58
3. More About Applesoft BASIC	63
3.1 Working With Tabular Data	63
3.2 Inputting Data	70
3.3 Advanced Printing	78
3.4 Gambling With Your Computer	83
3.5 Subroutines	89
4. Easing Programming Frustrations	97
4.1 Editing Program Lines	97
4.2 Flow Charting	101
4.3 Errors and Debugging	104
4.4 Appendix—Some Common Error Messages	107
5. Your Computer as a File Cabinet	109
5.1 What are Data Files?	109
5.2 Using An Apple II Computer Diskette File	111
5.3 An Introduction to DOS	116
5.4 Data Files for Disk Users	118
6. An Introduction to Computer Graphics	127
6.1 Low Resolution Graphics Principles	127
6.2 High Resolution Graphics	134
6.3 Computer Art	134

7. Word Processing	137	
7.1 What is Word Processing?	137	
7.2 Manipulating Strings	138	
7.3 Printer Controls and Form Letters	146	
7.4 Using Your Computer as a Word Processor		150
7.5 A Do-It-Yourself Word Processor	153	
8. Computer Games	157	
8.1 Telling Time With Your Computer	157	
8.2 Blind Target Shoot Game	162	
8.3 Tic Tac Toe	167	
9. Programming for Scientists	175	
9.1 Integer and Real Constants	175	
9.2 Variable Types	177	
9.3 Mathematical Functions in Applesoft BASIC		179
9.4 Defining Your Own Functions	184	
10. Computer-Generated Simulations	187	
10.1 Simulation	187	
10.2 Simulation of a Dry Cleaners	189	
11. Software You Can Buy	197	
11.1 VISICALC*	197	
11.2 Buying Software	205	
12. Some Other Applications of Your Computer		209
12.1 Computer Communications	209	
12.2 Information Storage and Retrieval		212
12.3 Advanced Graphics	212	
12.4 Connections to the Outside World		213
13. Where to Go From Here	215	
13.1 Assembly Language Programming	215	
13.2 Other Languages and Operating Systems		217
Answers to Selected Exercises	219	
Index	244	

* Trademark of Personal Software, Inc.

PREFACE

This book is designed to teach the computer novice how to use the Apple II computer. The development of the personal computer is one of the most exciting breakthroughs of our time. Indeed, the inexpensive personal computer promises to bring the computer revolution to tens of millions of people, and promises to alter the way we think, learn, work, and play. This book is designed to serve as an introduction to this revolution. Accordingly, the book has two purposes. First, it instructs the reader in the operation of the Apple II computer, and secondly, it illustrates some of the many ways it can be used.

We will guide the reader from the moment he or she turns on the computer for the first time, and discuss the rudiments of programming in the BASIC language. Since the book is designed as a tutorial, it includes an exercise set with each section. This book may also be used for self-study. Located in the text of each section are boxed questions labelled "Test Your Understanding." These questions test you on concepts introduced in each section and comprise a built-in study guide. The answers to the "Test Your Understanding" questions are found after the exercises of each section.

We have covered more topics than those normally found in most other books on elementary programming. This is due to our conviction that the beginner should gain, in addition to learning BASIC programming, an overview of as many real-life applications as possible. Therefore, we have included many applied discussions, including a brief look at word processing. These applications are designed to whet the reader's interest and may be used as preludes to further study.

Most enthusiastic personal computer users quickly upgrade their computers to include various equipment. We have included here an introduction to Apple DOS, and a brief discussion on printers and communications interfaces. This book closes with a discussion of possible topics for further study.

Any book owes its existence to the dedicated labors and inspirations of many people. In our case, we have been inspired by our wives, Sandy and Rose. Our children/grandchildren Melissa and Jonathan have enthusiastically joined us in applying our computers to a variety of tasks. Their enthusiasm and fresh viewpoints have given us a glimpse into the future of the computer revolution. Our sincere thanks to our reviewers, David Molz, Paul Tebbe, Anne Gygi, Gerald Mellentin, K. W. Miller, and Jim Morey for their careful scrutiny of the manuscript and their

many helpful suggestions. Thanks also go to Michael Rogers, production editor and designer, for the professional manner in which he managed the editing and production of this book. Finally, we would like to thank Harry Gaines, President of the Brady Company, and David Culverwell, Editor in Chief of the Brady Company, for their continued support. Their friendship has enhanced our excitement and pleasure in writing this book and can serve as a model for the ideal relationship between publishers and authors.

Larry Joel Goldstein
Silver Spring, Maryland

Martin Goldstein
West Palm Beach, Florida

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The author and publisher of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

1

A First Look At Computers

1.1 INTRODUCTION

The computer age is barely thirty years old, but it has already had a profound effect on our lives. Indeed, computers are now standard equipment in the office, the factory, and even the supermarket. In the last three or four years, the computer has even become commonplace in the home, as people have purchased millions of computer games and hundreds of thousands of personal computers. Computers are so common today that it is hard to imagine even a single day in which a computer will not somehow affect us.

In spite of the explosion of computer use in our society, most people know very little about them. They view a computer as an “electronic brain,” and do not know how a computer works, how it may be used, or how it may greatly simplify various everyday tasks. This does not reflect a lack of interest. Most people realize that computers are here to stay, and are interested in finding out how to use them. If you are so inclined, then this book is for you!

This book is an introduction to personal computing for the novice. You may be a student, teacher, homemaker, business person, or just a curious individual. We assume that you have had little or no previous exposure to computers and want to learn the fundamentals. We will guide you as you turn on your computer for the first time. (There’s really nothing to it!) From there, we will lead you through the fundamentals of talking with your computer in the Applesoft®* BASIC lan-

*Applesoft is a registered trademark of Apple Computer Corporation.

guage. Throughout, we will provide exercises for you to test your understanding of the material. We will show you the many ways *you can use your computer*. The exercises will suggest programs you can write. Many of the exercises will be designed to give you insight into how computers are used in business and industry. We will also suggest a number of computer applications for your home. For good measure, we will even build a few computer games!

WHAT IS PERSONAL COMPUTING?

In the early days of computing (the 1940's and 1950's), the typical computer was a huge mass of electronic parts which occupied several rooms. In those days, it was often necessary to reinforce the floor of a computer room, and to install special air conditioning so the computer could function properly. An early computer was likely to cost several million dollars. Over the years, the cost of computers has decreased dramatically and, thanks to micro-miniaturization, their size has shrunk even faster than their price.

The first "personal" computers appeared on the market a few years ago. These computers were reasonably inexpensive, and were designed to allow the average person to learn about the computer and how to use it to solve everyday problems. These personal computers were incredibly popular and several hundred thousand of them were sold in three years.

The personal computer is not a toy. It is a genuine computer which has most of the features of its big brothers, the so-called "main-frame" computers, which still cost several million dollars. A personal computer can be equipped with enough capacity to handle the accounting and inventory control tasks of most small businesses. It can also perform computations for engineers and scientists, and it can even be used to keep track of home finances and personal clerical chores. It would be quite impossible to give anything like a comprehensive list of the applications of personal computers. However, the following list suggests the range of possibilities:

For the business person

- Accounting
- Record keeping
- Clerical chores
- Inventory
- Cash management

- Payroll
- Graph and chart preparation

For the home

- Record keeping
- Budget management
- Investment analysis
- Correspondence
- Energy conservation
- Home security

For the student

- Computer literacy
- Term paper preparation
- Analysis of experiments
- Preparation of graphs and charts
- Project schedules
- Storage and organization of notes

For the professional

- Billing
- Analysis of data
- Report generation
- Correspondence

For recreation

- Computer games
- Computer graphics
- Computer art

As you can see, the list is quite comprehensive. If your interests are not listed, don't worry! There's plenty of room for those of you who are just plain curious about computers and wish to learn about them as a hobby.

THE APPLE®II COMPUTER*

This book will introduce you to personal computing using the Apple II Computer. This machine is an incredibly sophisticated device which uses many of the features of its main-frame big brothers. Before we begin discussing these particular features of the Apple®II Computer, let us discuss the features found in all computers.

*Apple II is a registered trademark of Apple Computer Corporation.

1.2 WHAT IS A COMPUTER?

At the heart of every computer is a **central processing unit** (or **CPU**) which performs the commands you specify. This unit carries out arithmetic, makes logical decisions and so forth. In essence, the CPU is the “brain” of the computer. The **memory** of a computer lets it “remember” numbers, words, and paragraphs, as well as the list of commands you wish the computer to perform. The **input unit** allows you to send information to the computer; the **output unit** allows the computer to send information to you. The relationship of these four basic components of a computer is shown in Figure 1-1.

In an Apple II Computer, the CPU is contained in a tiny electronic chip called a **6502 microprocessor**. As a computer novice, it is not necessary for you to know about the electronics of the CPU. For now, view the CPU as a magic device somewhere inside the case of your computer and don't give it another thought!

The input device of the Apple II Computer is the computer keyboard. We will discuss the special features of the keyboard in Section 1.3 (page 5). For now, think of the keyboard as a typewriter. By typing symbols on the keyboard, you are inputting them to the computer.

The Apple II Computer has a number of output devices. The most basic is the “TV screen” (sometimes called the video monitor or **video display**). You may also use a printer to provide output on paper. In computer jargon, paper output is called **hard copy**.

There are four types of memory in an Apple II Computer: **ROM**, **RAM**, **cassette**, and **diskette**. Each of these types of memory has its own advan-

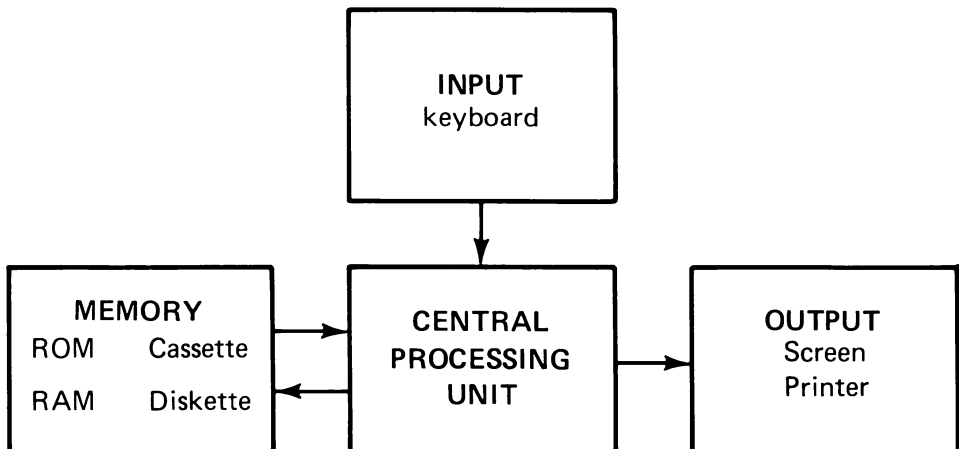


Figure 1-1. The main components of a computer.

tages and disadvantages. We will attempt to make the memory as versatile as possible by combining the good features of each.

ROM stands for "read only memory." This type of memory can be read by the computer (that is, the CPU), but you cannot record anything in it. ROM is usually reserved for the computer language which the CPU utilizes. This language will be discussed later. For now, just remember that ROM contains the information necessary for the computer to understand your commands. This information is pre-recorded in the factory and is permanently situated in ROM.

RAM stands for "random access memory." This is the memory which you can write into. If you type characters on the keyboard, they are stored in RAM. Similarly, the results of calculations are kept in RAM awaiting output to you. There is an extremely important feature of RAM which you should remember:

Important: If the computer is turned off, then RAM is erased.

Therefore, RAM may not be used to store data in permanent form. Nevertheless, it is used as the computer's main working storage because of its great speed. It takes about a millionth of a second to store or retrieve a piece of data from RAM.

To make permanent copies of programs and data, we may use either the cassette recorder or the diskette file. The cassette recorder is just a tape recorder which allows recording of information in a form which the computer can understand. The recording tape is the same sort you use for musical recordings.

A diskette drive records information on flexible diskettes which resemble phonograph records. These diskettes, often called "floppy disks," can store several hundred thousand characters each! (A double-spaced typed page contains about 3,000 characters.) A diskette file can provide access to information in much less time, on the average, than a cassette recorder. On the other hand, diskette files are more costly than cassette recorders.

The Apple II Computer comes in both diskette and nondiskette models. We will describe the operation of both versions.

1.3 MEET YOUR APPLE II COMPUTER

The best way to quickly master the operation of your computer is to read this book while sitting at the computer and verifying the various

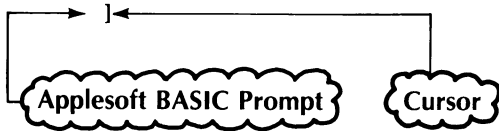


Figure 1-2. The Apple II Computer.

statements as they come up. So why not have a seat in front of your Apple II Computer? If your computer is not conveniently available, refer to Figures 1-2 and 1-3.

Begin by examining the keyboard. Note that it is similar to a typewriter keyboard, with a few important differences. Many typewriters use the same key for the number 1 and a lower case letter l. For the computer, however, spellings must not allow for any confusion; there are separate keys for these two symbols. Similarly, it is very easy to confuse the capital letter O ('oh') and the number 0 (zero) For this reason, a computer specialist usually writes zero with a slash through it: 0/. To prevent possible confusion, you should do likewise.

Note that the keyboard has a number of specialized keys which are not on a standard typewriter keyboard. We will discuss these keys one at a time, but first let's turn the computer on. Turn on the power to the video display and then turn on the power to the computer itself. Then push the **RESET** key. The computer will display a copyright notice and two symbols:



The computer is now awaiting your instructions! Strike a few keys to get the feel of the keyboard. Note that as you type, the corresponding characters will appear on the screen. Note, also, how the small white box travels along the typing line. This dash is called the **cursor**. It always sits at the location where the next typed character will appear. Note the character `]`. This is the **Applesoft BASIC prompt** which indicates that the computer language (called Applesoft BASIC) is ready to accept your instructions.

As you type, you should notice the similarities between the Apple II Computer keyboard and that of a typewriter. However, you should also note the differences. At the end of a typewriter line you return the carriage, either manually or, on an electric typewriter, with a carriage return key. Of course, your screen has no carriage to return. However, you still must tell the computer that you are ready to move on to the next line. This is accomplished by hitting the **RETURN** key. If you depress the **RETURN** key, the cursor will then return to the next line and position itself at the extreme left side of the screen. The **RETURN** key also has another function. **RETURN** signals the computer to accept the line just typed. Until you hit the **RETURN** key, as far as the computer is concerned, the line you just typed does not exist!

Keep typing until you are at the bottom of the screen. If you hit **RETURN**, the entire contents of the screen will move up by one line and

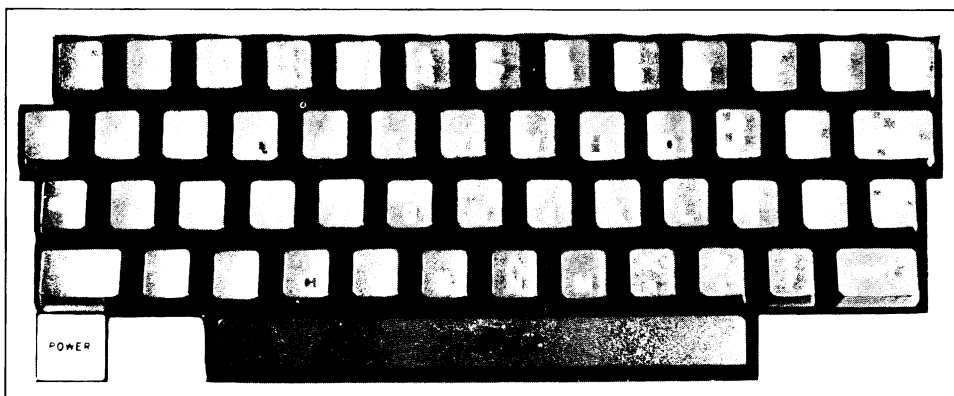


Figure 1-3. The Apple II Computer keyboard.

the line at the top of the screen will disappear. This movement of lines on and off the screen is called *scrolling*.

As you may have already noticed, the computer will respond to some of your typed lines with error messages. Don't worry about these now. The computer has been taught to respond only to certain typed commands. If it encounters a command that it doesn't recognize, it will announce this fact with an error message. It is extremely important for you to realize that these errors will not harm the computer. In fact, there is little you can do to hurt your computer (except by means of physical abuse, of course). Don't be intimidated by the occasional slaps on the wrist handed out by your computer. Whatever happens, don't let these "slaps" stop you from experimenting. The worst that can happen is that you might have to turn your computer off and start all over!

By this time, your screen should look pretty cluttered. To clear it, type **HOME** and then press **RETURN**. All characters on the screen will be erased and only the cursor will remain. The cursor is positioned in the upper left corner of the screen, its so-called "home" position.

TEST YOUR UNDERSTANDING 1* (answers on page 9)

- a. Type your name on the screen.
- b. Erase the screen.

Unless you are a superb typist (most of us are at the other extreme!), you will eventually make typing errors. Let's discover how to correct them. Type a few characters, but don't hit the **RETURN** key. Now hit the backspace key. (This is the key labelled ←.) Note that this key causes the cursor to backspace, one space at a time, erasing the characters it passes over. This is another difference between a typewriter and a computer keyboard. Note, however, that you may use the backspace to correct lines only if they have not been sent to the computer via the **RETURN** key.

There are other ways to correct typing errors, but for now let us be content with the methods discussed above.

*Answers to the **TEST YOUR UNDERSTANDING** questions follow the exercises of the section. See page 9.

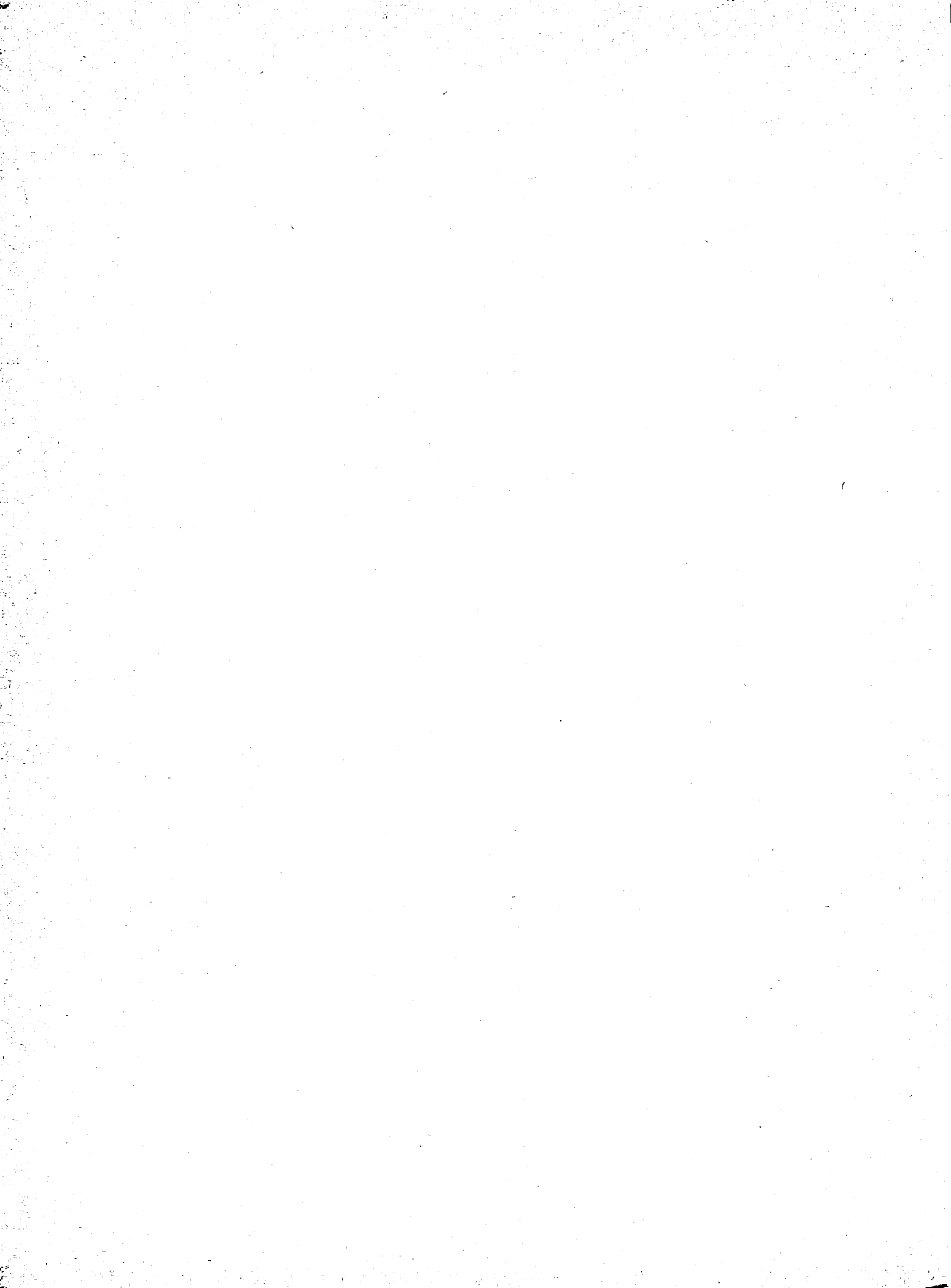
EXERCISES

Type the following expressions on the screen. After each numbered exercise, clear the screen.

- | | |
|---------------------------------|----------------------|
| 1. 10 HELLO! I'M YOUR NEW OWNER | 2. 10 ARITHMETIC |
| 3. 10 PRINT 3+7 | 4. 20 LET A=3-5 |
| 5. 20 5% OF 68 | 6. 10 IF 38>-5 |
| 7. 10 X=5: PRINT X | 8. 20 IF X>0 THEN 50 |
| 9. 10 LET X=10 | 10. 200 Y=X*2-5 |
| 20 LET Y=50.35 | 300 PRINT Y,"Y" |

ANSWERS TO TEST YOUR UNDERSTANDING 1

- Type your name, ending the line with RETURN.
- Type HOME followed by RETURN.



2

Getting Started in Applesoft BASIC

2.1 COMPUTER LANGUAGES AND PROGRAMS

In Chapter One we learned how to manipulate the keyboard and display screen of the Apple II Computer. Let's now learn how to communicate instructions to the computer.

Just as humans use languages to communicate with one another, computers use languages to communicate with other electronic devices (such as printers), human operators, and other computers. There are hundreds of computer languages in use today. However, the most common one for microcomputers is called **BASIC** (**B**eginners **A**ll **P**urpose **S**ymbolic **I**nstructional **C**ode). This is the best computer language to learn first since it is the most elementary computer language used by your Apple II Computer. The first version of BASIC (not Applesoft) was developed especially for computer novices by John Kemeny and Thomas Kurtz at Dartmouth College. In the next few chapters, we will concentrate on learning the fundamentals of Applesoft BASIC. In the process, we will learn a great deal about the way in which a computer may be used to solve problems.

Many people think of a computer as an "electronic brain" which somehow has the power of human thought. This is very far from the truth. The electronics of the computer and the rules of the Applesoft BASIC language allow it to recognize a very limited vocabulary, and to take various actions based on the data which is given to it. It is very

important to recognize that the computer does not have “common sense.” The computer will attempt to interpret whatever data you input. If what you input is a recognizable command, the computer will perform it. It does not matter that the command makes no sense in a particular context. The computer has no way to make such judgments. It can only do what you instruct it to do. Because of the computer’s inflexibility in interpreting commands, you must tell the computer **exactly** what you want it to do. Do not worry about confusing the computer. If you communicate a command in an incorrect form, you won’t damage the machine in any way! In order to make the computer do our bidding, it is necessary to carefully learn its language. That is our goal.

Let’s begin to learn something about Applesoft BASIC. Assume that you have followed the start-up instructions of the last chapter and the computer shows that it is ready to accept further instructions by displaying the Applesoft BASIC prompt:

```
] 
```

From this point on, a typical session with your computer might go like this:

1. Type in a series of instructions in Applesoft BASIC. Such a series of instructions is called a **program**.
2. Locate and correct any errors in the program.
3. Tell the computer to carry out the series of instructions in the program. This step is called **running the program**.
4. Obtain the output requested by the program.
5. Either: (a) run the program again; or (b) repeat steps 1–4 for a new program; or (c) end the programming session (turn off the computer and go have lunch).

To fully understand what is involved in these five steps, let us consider a particular example. Suppose that you want the computer to add 5 and 7. First, you would type the following instructions:

```
10 PRINT 5+7  
20 END
```

This sequence of two instructions constitutes a program to calculate 5 + 7. Note that as you type the program the computer records your instructions, **but does not carry them out**. As you are typing a program, the computer provides you with an opportunity to change, delete, and correct instruction lines. (More on how to do this later.) Once you are

content with your program, tell the computer to run it (that is, to execute the instructions) by typing the command*:

RUN

The computer will run the program and display the desired answer:

12

If you wish the computer to run the program a second time, type **RUN** again.

Running a program does not erase it from RAM. Therefore, if you wish to add instructions to the program or change the program, you may continue typing just as if the **RUN** command had not intervened. For example, if you wish to include in your program the problem of calculating $5 - 7$, type the additional line

15 PRINT 5-7

To see the program currently in memory, type **LIST** (no line number), then hit the **RETURN** key. The program consists of the following three lines, now displayed on the screen:

10 PRINT 5+7

15 PRINT 5-7

20 END

Note how the computer puts line 15 in proper sequence. If we now type **RUN** again, the computer will display the two answers

12

-2

In the event that you now wish to go on to another program, type the command

NEW

This erases the previous program from RAM and prepares the computer to accept a new program. You should always remember the following important fact

RAM can contain only one program at a time.

*Don't forget to end the line by pressing the **RETURN** key.

TEST YOUR UNDERSTANDING 1 (answers on page 14)

- a. Write and type in a BASIC program to calculate $12.1 + 98 + 5.32$.
- b. Run the program of a.
- c. Erase the program of a. from RAM.
- d. Write a program to calculate $48.75 - 1.674$.
- e. Type in and run the program of d.

Applesoft BASIC on the Apple II Computer operates in two distinct modes. In **command mode**, the computer accepts typed program lines and commands (like **RUN** and **NEW**) used to manipulate programs. The computer identifies a program line by its line number. Program lines are not immediately executed. Rather, they are stored in RAM until you tell the computer what to do with them. On the other hand, commands are executed as soon as they are given.

In the **execute mode**, the computer runs a program. In this mode, the screen is under control of the program.

When you turn the computer on it is automatically in command mode. The command mode is indicated by the presence of the] prompt on the screen. The **RUN** command puts the computer into execute mode. After you run the program the computer redisplay the] prompt indicating that it is back in command mode.

The computer is a stern taskmaster! It has a very limited vocabulary (Applesoft BASIC) and this vocabulary must be used according to very specific rules concerning the order of words, punctuation, and so forth. However, Applesoft BASIC allows for some freedom of expression. For example, any extra spaces are ignored. Thus, Applesoft BASIC will interpret all of the following instructions as the same:

```
10 PRINT A
10 PRINT  A
10  PRINT A
```

ANSWERS TO TEST YOUR UNDERSTANDING 1

- a. 10 PRINT 12.1+98+5.32
20 END
- b. Type RUN

- c. Type NEW
- d. 10 PRINT 48.75-1.674
20 END
- e. Type in the program followed by RUN

2.2 ELEMENTARY APPLESOFT BASIC PROGRAMS

In learning to use a language, you must first learn the alphabet of the language. Next, you must learn the vocabulary of the language. Finally, you must study the way in which words are put together into sentences. In learning the Applesoft BASIC language, we will follow the progression just described. In Chapter One, we learned about the characters of the Apple II keyboard. These characters are the alphabet of Applesoft BASIC. Let us now learn some basic vocabulary. The simplest "words" are the so-called constants.

APPLESOFT BASIC CONSTANTS

Applesoft BASIC allows us to manipulate numbers and text. The rules for manipulating numerical data differ from those for handling text, however. In Applesoft BASIC we distinguish between these two types of data as follows: a **numeric constant** is a number and a **string constant** is a sequence of keyboard characters, which may include letters, numbers, or any other keyboard symbols. The following are examples of numeric constants:

5, -2, 3.145, 23456, 456.78345676543987, 27134566543

The following are examples of string constants:

"John", "Accounts Receivable", "\$234.45 Due", "Dec. 4,1981"

Note that string constants are always enclosed in quotation marks. In order to avoid vagueness, quotation marks may not appear as part of a string constant. (In practice, an apostrophe ' should be used as a substitute for " within a string constant.) Although numbers may appear within a string constant, you cannot use such numbers in arithmetic. Only numbers not enclosed by quotation marks may be used for arithmetic.

For certain applications, you may wish to specify your numeric constants in *exponential format*. This will be especially helpful in the case of very large and very small numbers. Consider the number 15,300,000,000. It is very inconvenient to type all the zeros. This large number can be written in handy shorthand as 1.53E10. The 1.53 indicates the first three digits of the number. The E10 means that you move the decimal point in the 1.53 to the right 10 places. Similarly, the number -237,000 may be written in the exponential format as -2.37E5. The exponential format may also be used for very small numbers. For example, the number .0000000054 may be written in exponential format as 5.4E-10. The -10 indicates that the decimal point in 5.4 is to be moved 10 places to the *left*.

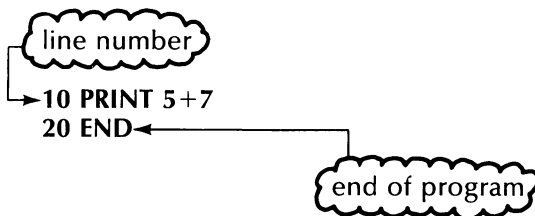
TEST YOUR UNDERSTANDING 1 (answers on page 24)

- a. Write these numbers in exponential format: .00048, -1374.5
- b. Write these numbers in decimal format: -9.7E3, 9.7E-3, -9.7E-3

We will say more about constants later. For example, we will describe the number of digits of accuracy you can get, how to round off numbers, and so forth. Right now, you know more than enough to get started. So instead of concentrating on the fine points now, let's learn enough to make our computer **do something**.

APPLESOFT BASIC PROGRAMS

Let us look again at the Applesoft BASIC program in Section 2.1 (page 12), namely:



This program illustrates two very important features common to all Applesoft BASIC programs:

1. The instructions of a program must be numbered. Each line must start with a line number. The computer executes instructions in order of increasing line number.
2. The **END** instruction identifies the end of the program. On encountering this instruction, the computer stops running the program and displays].

Note that line numbers need not be consecutive. For example, it is perfectly acceptable to have a program whose line numbers are 10, 23, 47, 55, or 100. Also note that it is not necessary to type instructions in their numerical order. You could type line 20 and then go back and type line 10. The computer will sort out the lines and rearrange them according to increasing number. This feature is especially helpful in case you accidentally omit a line while typing your program.

Here is another important fact about line numbering. If you type two lines having the same line number, the computer erases the first version and remembers the second version. This feature is very useful for correcting errors: if a line has an error, just retype it!

Your Apple II Computer will perform all the standard calculations that can be done with a calculator. Since most people are familiar with the operation of a calculator, let us start by writing programs to solve various arithmetic problems.

Most arithmetic operations are written in customary fashion. For example, addition and subtraction are written for the computer in the usual way:

5+4, 9-8

Multiplication, however, is typed using the symbol *, which shares the ":" key. Therefore, the product of 5 and 3 is typed as:

5*3

Division is typed using /. Therefore, 8.2 divided by 15 is typed as:

8.2/15

Example 1. Write an Applesoft BASIC program to calculate the sum of 54.75, 78.83, and 548.

Solution. The sum is indicated by typing

54.75+78.83+548

The Applesoft BASIC instruction for printing data on the screen is **PRINT**. So we write our program as follows:

```
10 PRINT 54.75+78.83+548
20 END
```

Applesoft BASIC carries out arithmetic operations in a special order. It scans an expression and carries out all multiplication and division operations first, *proceeding in left-to-right order*. It then returns to the left side of the expression and performs addition and subtraction, also in a left-to-right order. If parentheses occur, these are evaluated first following the same rules stated above. If parentheses occur within parentheses, the innermost parentheses are evaluated first.

Example 2. What are the numerical values which Applesoft BASIC will calculate from these expressions?

(a) $(5 + 7)/2$

(b) $5 + 7/2$

(c) $5 + 7*3/2$

(d) $(5 + 7*3)/2$

Solution. (a) The computer first applies its rules for the order of calculation to determine the value in the parentheses, namely 12. It then divides 12 by 2 to obtain 6.

(b) The computer scans the expression from left to right performing all multiplication and division in the order encountered. First it divides 7 by 2 to obtain 3.5. It then rescans the line and performs all additions and subtractions in order. This gives

$$5 + 3.5 = 8.5$$

(c) The computer first performs all multiplication and division in order:

$$5 + 10.5$$

Now it performs addition and subtraction to obtain 15.5.

(d) The computer calculates the value of all parentheses first. In this case, it computes $5 + 7*3 = 26$. (Note that it does the multiplication first!) Next it rescans the line which now looks like

$$26/2$$

It performs the division to obtain 13.

TEST YOUR UNDERSTANDING 2 (answer on page 24)

Calculate $5+3/2+2$ and $(5+3)/(2+2)$.

Example 3. Write an Applesoft BASIC program to calculate the quantity

$$\frac{22 \times 18 + 34 \times 11 - 12.5 \times 8}{27.8}$$

Solution. Here is the program:

```
10 PRINT (22*18+34*11-12.5*8)/27.8
20 END
```

Note that we used parentheses in line 10. They tell the computer that the entire quantity in parentheses is to be divided by 27.8. If we had omitted the parentheses, the computer would divide $-12.5*8$ by 27.8 and add $22*18$ and $34*11$ to the result.

TEST YOUR UNDERSTANDING 3 (answers on page 24)

Write Applesoft BASIC programs to calculate:

- $((4 \times 3 + 5 \times 8 + 7 \times 9)/(7 \times 9 + 4 \times 3 + 8 \times 7)) \times 48.7$
- 27.8 % of $(112 + 38 + 42)$
- The average of the numbers 88, 78, 84, 49, 63

PRINTING WORDS

So far, we have used the **PRINT** instruction only to display the answers to numerical problems. However, this instruction is very versatile. It also allows us to display string constants. For example, consider this instruction:

```
10 PRINT "PATIENT HISTORY"
```

During program execution, this statement will create the following display:

```
PATIENT HISTORY
```

In order to display several string constants on the same line, separate them by commas in a single **PRINT** statement. For example, consider the instruction

```
10 PRINT "AGE", "BIRTHPLACE"
```

It will cause two words to be printed as follows:

```
AGE      BIRTHPLACE
```

Both numeric constants and string constants may be included in a single **PRINT** statement, for example

```
100 PRINT "AGE", 65.43
```

Here is how the computer determines the spacing on a line. Each line is divided into **print zones**. Each print zone consists of 16 spaces. By placing a comma in a **PRINT** statement, you are telling the computer to start the next string of text at the beginning of the next print zone. Thus, for example, the two words above, *AGE* and *BIRTHPLACE*, begin in columns 1, and 17, respectively. (See Figure 2-1.) Note that the third print zone (positions 33–40) is available only if nothing is printed in print positions 24–32. Moreover, to use print zone 2, you must leave print position 32 blank.

TEST YOUR UNDERSTANDING 4 (answer on page 24)

Write a program to print the following display.

	NAME	
LAST	FIRST	MIDDLE
SMITH	JOHN	DAVID

Example 4. Suppose that a distributor of office supplies sells 50 chairs and five desks. The chairs cost \$59.70 each and are subject to a 30 percent discount. The desks cost \$247.90 each and are also subject to a 30 percent discount. Prepare a bill for the shipment.

Solution. Let us insert two headings on our bill: *Item* and *Cost*. We then print two lines, corresponding to the two types of items shipped. Finally, we calculate the total due as shown here.

```
10 PRINT "ITEM","COST"
20 PRINT
30 PRINT "CHAIR",50*(59.70-.3*59.70)
40 PRINT "DESK", 5*(247.90-.3*247.90)
50 PRINT
60 PRINT "TOTAL DUE",50*(59.70-.3*59.70)+5*(247.90-.3*247.90)
```

Note the **PRINT** statements in lines 20 and 50. They specify that a blank

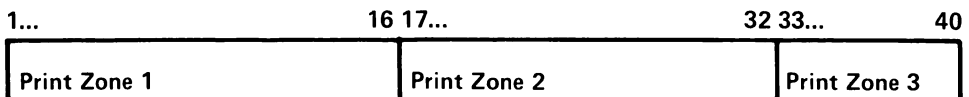


Figure 2-1. Print zones.

line is to be printed. If we now type **RUN** (followed by **RETURN**), the screen will look like this:

```

10 PRINT "ITEM","COST"
20 PRINT
30 PRINT "CHAIR",50*(59.70-.3*59.70)
40 PRINT "DESK", 5*(247.90-.3*247.90)
50 PRINT
60 PRINT "TOTAL DUE",50*(59.70-.3*59.70)+5*(247.90-.3*247.90)
70 END

```

RUN

ITEM	COST
CHAIR	2089.50
DESK	867.65
TOTAL DUE	2957.15

You may think that the above invoice is somewhat sloppy because the columns of figures are not properly aligned. Patience! We will learn to align the columns after we have learned a bit more programming.

TEST YOUR UNDERSTANDING 5 (answer on page 24)

Write a computer program which creates the following display.

BUDGET-APRIL	
FOOD	387.50
CAR	123.71
GAS	100.00
UTILITIES	146.00
ENTERTAINMENT	100.00

TOTAL	(Calculate total)

EXPONENTIATION

Suppose that A is a number and N is a positive whole number (this means that N is one of the numbers 1, 2, 3, 4, . . .). Then A *raised to the N th power* is the product of A times itself N times. This quantity is

usually denoted in mathematics texts as A^N , and the process of calculating it is called **exponentiation**. For example,

$$2^3 = 2*2*2 = 8, 5^7 = 5*5*5*5*5*5*5 = 78125$$

It is possible to calculate A^N by repeated multiplication. However, if N is large, this can be tiresome to type. Applesoft BASIC provides a shortcut for typing this function. Exponentiation is denoted by the symbol \wedge , which is produced by hitting the key with the upward-pointing arrow (this symbol shares the "N" key at the top of the keyboard). For example, 2^3 is denoted $2\wedge 3$. The operation of exponentiation takes precedence over multiplication and division. This is illustrated in the following example.

Example 5. Determine the value which Applesoft BASIC assigns to this expression:

$$20*3 - 5*2\wedge 3$$

Solution. The exponentiation is performed first to yield:

$$\begin{aligned} 20*3 - 5*8 &= 60 - 40 \\ &= 20 \end{aligned}$$

TEST YOUR UNDERSTANDING 6 (answer on page 24)

Evaluate the following first manually and then by an Apple II Computer program.

a. $2^4 \times 3^3$

b. $2^2 \times 3^3 - 12^2/3^2 \times 2$

EXERCISES (answers on page 219)

Write Applesoft BASIC programs to calculate the following quantities.

- $57 + 23 + 48$
- $57.83 \times (48.27 - 12.54)$
- $127.86/38$
- $365/.005 + 1.02^5$
- Make a table of the first, second, and third powers of the numbers 2, 3, 4, 5, and 6. Put all first powers in a column, all second powers in another column, and so forth.

6. Mrs. Anita Smith visited her doctor regarding a broken leg. Her bill consists of \$45 for removal of the cast, \$35 for therapy, and \$5 for drugs. Her major medical policy will pay 80 percent directly to the doctor. Use the computer to prepare an invoice for Mrs. Smith.
7. A school board election is held to elect a representative for a district consisting of Wards 1, 2, 3, and 4. There are three candidates, Mr. Thacker, Ms. Hoving, and Mrs. Weatherby. The tallies by candidate and ward are as follows.

	Ward 1	Ward 2	Ward 3	Ward 4
Thacker	698	732	129	487
Hoving	148	928	246	201
Weatherby	379	1087	148	641

Write an Applesoft BASIC computer program to calculate the total number of votes achieved by each candidate, as well as the total number of votes cast.

Describe the output from each of these programs.

8.

```
10 PRINT 8*2-3*(2^4-10)
20 END
```
9.

```
10 PRINT "SILVER","GOLD","COPPER"
20 PRINT 327,448,1052
30 END
```
10.

```
10 PRINT , "GROCERIES","MEATS"
20 PRINT "MON", "1,245","2,348"
30 PRINT "TUE", " 248","3,459"
40 END
```

Convert the following numbers to exponential format.

11. 23,000,000
12. 175.25
13. -200,000,000
14. .00014
15. -.000000000275
16. 53,420,000,000,000,000

Convert the following numbers in exponential format to standard format.

17. 1.59E5
18. -20.3456E6
19. -7.456E-12
20. 2.39456E-18

ANSWERS TO TEST YOUR UNDERSTANDING 1,2,3,4,5, and 6

- 1: a. 4.8E-4, -1.3745E3
b. -9700, .0097, -.0097
- 2: 8.5, 2
- 3: a. 10 PRINT ((4*3+5*8+7*9)/(7*9+4*3+8*7))*48.7
20 END
b. 10 PRINT .278*(112+38+42)
20 END
c. 10 PRINT (88+78+84+49+63)/5
20 END
- 4: 10 PRINT , "NAME"
20 PRINT
30 PRINT "LAST", "MIDDLE", "FIRST"
40 PRINT
50 PRINT "SMITH", "JOHN", "DAVID"
60 END
- 5: 10 PRINT , " BUDGET-APRIL"
20 PRINT "FOOD", 387.50
30 PRINT "CAR", 123.71
40 PRINT "GAS", 100.00
50 PRINT "UTILITIES", 146.00
60 PRINT "ENTERTAINMENT", 100.00
70 PRINT , " _____"
80 PRINT "TOTAL", 387.50+123.71+146.00+100.00
90 END
- 6: a. 432
b. 76

2.3 GIVING NAMES TO NUMBERS AND WORDS

In the examples and exercises of the preceding section, you probably noticed that you were wasting considerable time retyping certain numbers over and over. Not only does this retyping waste time, it is also a likely source for errors. Fortunately, such retyping is unnecessary if we use variables.

A **variable** is a letter used to represent a number. Any letter of the alphabet may be used as a variable. (There are other possible names for variables. See below.) Possible variables are A, B, C, X, Y, or Z. At any given moment, a variable has a particular value. For example, the variable A might have the value 5 while B might have the value -2.137845 . One method for changing the value of a variable is through use of the **LET** statement. The statement

10 LET A=7

sets the value of A equal to 7. Any previous value of A is erased. The **LET** command may be used to set the values of a number of variables simultaneously. For instance, the instruction

100 LET C=18: D=23: E=2.718

assigns C the value 18, D the value 23, and E the value 2.718.

Once the value of a variable has been set, the variable may be used throughout the program. The computer will insert the appropriate value wherever the variable occurs. For instance, if A has the value 7 then the expression

$A + 5$

is evaluated as $7 + 5$ or 12. The expression

$3*A - 10$

is evaluated $3*7 - 10 = 21 - 10 = 11$. The expression $2*A^2$ is evaluated

$2*7^2 = 2*49 = 98$

TEST YOUR UNDERSTANDING 1 (answer on page 31)

Suppose that A has the value 4 and B has the value 3. What is the value of the expression $A^{2/2}*B^2$?

Note the following important fact:

If you do not specify a value for a variable, Applesoft BASIC will assign it the value 0.

Variables may also be used in **PRINT** statements. For example, the statement

```
10 PRINT A
```

will cause the computer to print the current value of A (in the first print zone, of course!). The statement

```
20 PRINT A,B,C
```

will result in printing the current values of A, B, and C in print zones 1, 2, and 3, respectively.

TEST YOUR UNDERSTANDING 2 (answer on page 31)

Suppose that A has the value 5. What will be the result of the instruction:

```
10 PRINT A,A^2,2*A^2
```

Example 1. Consider the three numbers 5.71, 3.23, 4.05. Calculate their sum, their product, and the sum of their squares (that is, the sum of their second powers; such a sum is often used in statistics.).

Solution. Introduce variables A, B, and C and set them equal, respectively, to the three numbers. Then compute the desired quantities.

```
10 LET A=5.71: B=3.23: C=4.05
20 PRINT "THE SUM IS", A+B+C
30 PRINT "THE PRODUCT IS", A*B*C
40 PRINT "THE SUM OF SQUARES IS", A^2+B^2+C^2
50 END
```

TEST YOUR UNDERSTANDING 3 (answer on page 31)

Consider the numbers 101, 102, 103, 104, 105, and 106. Write a program which calculates the product of the first two, the first three, the first four, the first five, and all six numbers.

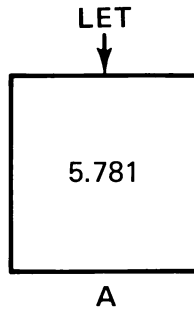


Figure 2-2. The variable A.

The following mental imagery is often helpful in understanding how Applesoft BASIC handles variables. When Applesoft BASIC first encounters a variable, let's say A, it sets up a box (actually a memory location) which it labels "A". (See Figure 2-2.) In this box it stores the current value of A. When you request a change in the value of A, the computer throws out the current contents of the box and inserts the new value.

Note that the value of a variable need not remain the same throughout a program. At any point in the program, you may change the value of a variable (with a LET statement, for example). If a program is called on to evaluate an expression involving a variable, it will always use the **current** value of the variable, ignoring any previous values the variable may have had at earlier points in the program.

TEST YOUR UNDERSTANDING 4 (answer on page 31)

Suppose that a loan for \$5,000 has an interest rate of 1.5 percent on the unpaid balance at the end of each month. Write a program to calculate the interest at the end of the first month. Suppose that at the end of the first month, you make a payment of \$150 (after the interest is added). Design your program to calculate the balance after the payment. (Begin by letting B = the loan balance, I = the interest, and P = the payment. After the payment, the new balance is $B + I - P$.)

Example 2. What will be the output of the following computer program?

```
10 LET A=10: B=20
20 LET A=5
30 PRINT A+B+C, A*B*C
40 END
```

Solution. Note that no value for C is specified, so $C = 0$. Also note that the value of A is initially set to 10. However, in line 20, this value is changed to 5. So in line 30, A, B, and C have the respective values 5, 20, and 0. Therefore, the output will be:

```
25      0
```

To the computer, the statement

```
LET A=
```

means that the current value of A is to be *replaced* with whatever appears to the right of the equal sign. Therefore, if we write

```
LET A=A+1
```

then we are asking the computer to replace the current value of A with $A + 1$. So, if the current value of A is 4, the value of A after performing the instruction is $4 + 1$, or 5.

TEST YOUR UNDERSTANDING 5 (answer on page 31)

What is the output of the following program?

```
10 LET A=5.3
20 LET A=A+1
30 LET A=2*A
40 LET A=A+B
50 PRINT A
60 END
```

LEGAL VARIABLE NAMES

As we mentioned previously, you may use any letter of the alphabet as a variable name. The Apple II Computer is quite flexible concerning variable names. Any sequence of up to 238 characters which begins with a letter is a legal variable name. (For an exception, see below.) Therefore, you may use variables named PAYROLL, TAX, REFUND, and BALANCE. **However, Applesoft BASIC will use only the first two characters to distinguish one name from another.** Actually, not every sequence of characters is a legal variable name. You must avoid any sequences of characters which are reserved by Applesoft BASIC for special meanings. Examples of such words are:

IF, ON, OR, TO

Once you become familiar with Applesoft BASIC, it will be second nature to avoid these and the other reserved words as variable names.

Note: A variable name must always start with a letter.

A variable name **cannot** begin with a number. For example, 1A is **not** a legal variable name.

So far, all of the variables we have discussed have represented numerical values. However, Applesoft BASIC also allows variables to assume string constants as values. The variables for doing this are called *string variables*. These are denoted by a variable name followed by a dollar sign. Thus, A\$, B1\$, and ZZ\$ are all valid names of string variables. To assign a value to a string variable, we use the LET statement with the desired value inserted in quotation marks after the equal sign. To set A\$ equal to the string "BALANCE SHEET", we use the statement

```
LET A$="BALANCE SHEET"
```

We may print the value of a string variable just as we print the value of a numeric variable. For example, if A\$ has the value just assigned, the statement

```
PRINT A$
```

will result in the following screen output:

```
BALANCE SHEET
```

Example 3. What will be the output of the following program?

```
10 LET A$="RECEIPTS":B$="EXPENSES"
20 LET A=20373.10: B=17584.31
30 PRINT A$,B$
40 PRINT A,B
50 END
```

Solution.

RECEIPTS	EXPENSES
20373.10	17584.31

Note that we have used the variables A and A\$ (as well as B and B\$) in the same program. The variables A and A\$ are considered *different* by the computer.

REMARKS IN PROGRAMS

It is very convenient to explain programs using remarks. For one thing, remarks make programs easier to read. Remarks also assist in finding errors and making modifications in a program. To insert a remark in a program, we may use the **REM** statement. For example, consider the line

```
520 REM X DENOTES THE COST BASIS
```

Since the line starts with **REM**, it will be ignored during program execution.

MULTIPLE STATEMENTS ON A SINGLE LINE

It is possible to put several Applesoft BASIC statements on a single line. Just separate them by a colon. For example, instead of the two statements:

```
10 LET A=5.784: B=3.571
20 PRINT A^2+B^2
```

we may use the single statement:

```
10 LET A=5.784: B=3.571 : PRINT A^2+B^2
```

To insert a remark on the same line as a program statement, use a colon followed by a **REM**, as in this example:

```
10 LET A=PI*R^2 :REM A IS THE AREA,R IS THE RADIUS
```

The **REM** statement causes the remainder of the line to be ignored by the computer. In what follows, we will sprinkle comments liberally throughout our programs so that they will be easier to understand.

TEST YOUR UNDERSTANDING 6 (answer on page 31)

What is the result of the following line?

```
10 LET A=7:B$="COST":C$="TOTAL"
20 PRINT C$,B$
30 PRINT "=",A
```

EXERCISES (answers on page 220)

In Exercises 1–6, determine the output of the given program.

- | | |
|---|--|
| <p>1. 10 LET A=5:B=5
20 PRINT A+B
30 END</p> <p>3. 10 LET A1=5
20 PRINT A1^2+5*A1
30 END</p> <p>5. 10 LET A\$="JOHN JONES"
20 LET B\$="AGE": C=38
30 PRINT A\$, B\$, C
40 END</p> | <p>2. 10 LET AA=5
20 PRINT AA*B
30 END</p> <p>4. 10 LET A=2: B=7: C=9
20 PRINT A+B, A-C, A*C
30 END</p> <p>6. 10 LET X=11, Y=19
20 PRINT 2*X
30 PRINT 3*Y
40 END</p> |
|---|--|

What is wrong with the following Applesoft BASIC statements?

- | | |
|--|---|
| <p>7. 10 LET A="YOUTH"</p> <p>9. 10 LET A\$=57</p> <p>11. 250 LET AAA=-9</p> | <p>8. 10 LET AA=-12</p> <p>10. LET ZZ\$= Address</p> <p>12. 10000 LET 1A=-2.34567</p> |
|--|---|
13. Consider the numbers 2.3758, 4.58321, 58.11. Write a program which computes their sum, product, and the sum of their squares.
14. A company has three divisions: Office Supplies, Computers, and Newsletters. The revenues of these three divisions for the preceding quarter were, respectively, \$346,712, \$459,321, and \$376,872. The expenses for the quarter were \$176,894, \$584,837, and, \$402,195, respectively. Write a program which displays this data on the screen, with appropriate explanatory headings. Your program should also compute and display the net profit (loss) from each division and the net profit (loss) for the company as a whole.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, 3, 4, 5, and 6

- 1: 72
- 2: It prints the display:
5 25 50
- 3: 10 LET A=101:B=102:C=103:D=104:E=105:F=106
20 PRINT A*B
30 PRINT A*B*C

```

40 PRINT A*B*C*D
50 PRINT A*B*C*D*E
60 PRINT A*B*C*D*E*F
70 END

4: 10 LET B=5000: I=.015: P=150.00
20 IN=I*B
30 PRINT "INTEREST EQUALS", IN
40 B=B+IN
50 PRINT " BALANCE WITH INTEREST EQUALS", B
60 B=B-P
70 PRINT "BALANCE AFTER PAYMENT EQUALS", B
80 END

5: 12.6

6: It creates the display:

TOTAL    COST
=         7

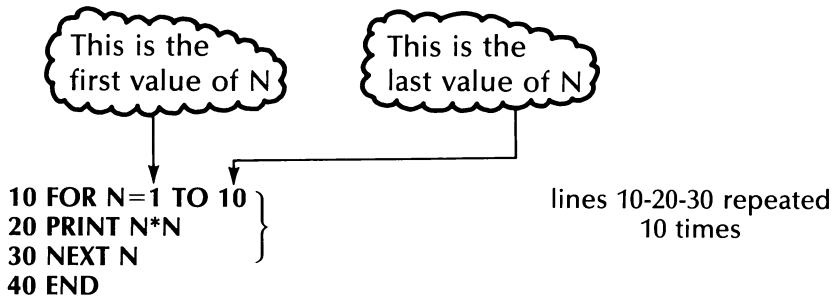
```

2.4 DOING REPETITIVE OPERATIONS

Suppose that we wish to solve 50 similar multiplication problems. It is certainly possible to type in the 50 problems one at a time and let the computer solve them. However, this is a very clumsy way to proceed. Suppose that instead of 50 problems there were 500, or even 5000. Typing the problems one at a time would not be practical. If, however, we can describe to the computer the entire class of problems we want solved, then we can instruct the computer to solve them using only a few Applesoft BASIC statements. Let us consider a concrete problem. Suppose that we wish to calculate the quantities

$$1^2, 2^2, 3^2, \dots, 10^2$$

That is, we wish to calculate a table of squares of integers from 1 to 10. This calculation can be described to the computer as calculating N^2 , where the variable N is allowed to assume, one at a time, each of the values 1, 2, 3, . . . , 10. Here is a sequence of Applesoft BASIC statements which accomplishes the calculations:



The sequence of statements 10, 20, and 30 is called a **loop**. When the computer encounters the FOR statement, it sets N equal to 1 and continues executing the statements. Statement 20 calls for printing N^2 . Since N is equal to 1, we have $N^2 = 1^2 = 1$. So the computer will print a 1. Next comes statement 30, which calls for the next N. This instructs the computer to return to the FOR statement in 10, increase N to 2, and to repeat instructions 20 and 30. This time, $N^2 = 2^2 = 4$. Line 20 then prints a 4. Line 30 says to go back to line 10 and increase N to 3 and so forth. Lines 10, 20, and 30 are repeated 10 times! After the computer executes lines 10, 20, and 30 with $N = 10$, it will leave the loop and execute line 40.

Type in the above program and give the **RUN** command. The output will look like this:

```

1
4
9
16
25
36
49
64
81
100
]

```

TEST YOUR UNDERSTANDING 1 (answers on page 41)

- Devise a loop allowing N to assume the values 3 to 77.
- Write a program which calculates N^2 for $N = 3$ to 77.

Let's modify the above program to include on each line of output not only N^2 , but also the value of N . To make the table easier to read, let's also add two column headings. The new program reads:

```
10 PRINT "N", "N^2"
20 FOR N=1 TO 10
30 PRINT N,N*N
40 NEXT N
50 END
```

The output now looks like this:

N	N^2
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

TEST YOUR UNDERSTANDING 2 (answer on page 42)

What would happen if we change the number of line 10 to 25?

Let us now illustrate some of the many uses loops have by means of some examples.

Example 1. Write an Applesoft BASIC program to calculate $1 + 2 + 3 + \dots + 100$.

Solution. Let us use a variable S (for sum) to contain the sum. Let us start S at 0 and use a loop to successively add to S the numbers 1, 2, 3, . . . , 100. Here is the program.

```
10 LET S=0
20 FOR N=1 TO 100 }
30 LET S=S+N      }   These instructions
40 NEXT N         }   repeated 100 times
50 PRINT S
60 END
```

When we enter the loop the first time, $S = 0$ and $N = 1$. Line 30 then replaces S by $S + N$, or $0 + 1$. Line 40 sends us back to line 20, where the value of N is now set equal to 2. In line 30, S (which is now $0 + 1$) is replaced by $S + N$, or $0 + 1 + 2$. Line 40 now sends us back to line 20,

where N is now set equal to 3. Line 30 then sets S equal to $0 + 1 + 2 + 3$. Finally, on the 100th time through the loop, S is replaced by $0 + 1 + 2 + \dots + 100$, the desired sum. If we run the program, we derive the output

```
5050
]
```

TEST YOUR UNDERSTANDING 3 (answer on page 42)

Write an Applesoft BASIC program to calculate $101 + 102 + \dots + 110$.

TEST YOUR UNDERSTANDING 4 (answer on page 42)

Write an Applesoft BASIC program to calculate and display the numbers $2, 2^2, 2^3, \dots, 2^{20}$.

Example 2. Write a program to calculate the sum:

$$1 \times 2 + 2 \times 3 + 3 \times 4 + \dots + 49 \times 50$$

Solution. We let the sum be contained in the variable S, like we did in Example 1. The quantities to be added are just the numbers $N*(N + 1)$ for $N = 1, 2, 3, \dots, 49$. So here is our program:

```
10 LET S=0
20 FOR N=1 TO 49
30 LET S=S+N*(N+1)
40 NEXT N
50 PRINT S
60 END
```

Example 3. You borrow \$7,000 to buy a car. You finance the balance for 36 months at an interest rate of one percent per month. Your monthly payments are \$232.50. Write a program which computes the amount of interest each month, the amount of the loan which is repaid, and the balance owed.

Solution. Let B denote the balance owed. Initially we have B equal to 7,000 dollars. At the end of each month let us compute the interest (I) owed for that month, namely $.01*B$. For example, at the end of the first month, the interest owed is $.01*7000.00 = \$70.00$. Let $P = 232.50$ to denote the monthly payment, and let R denote the amount repaid out

of the current payment. Then $R = P - I$. For example, at the end of the first month, the amount of the loan repaid is $232.50 - 70.00 = 162.50$. The balance owed may then be calculated as $B - R$. At the end of the first month, the balance owed is $7000.00 - 162.50 = 6837.50$. Here is a program which performs these calculations:

```

10 PRINT "INTEREST","BALANCE"
20 LET B=7000
25 LET P=232.50
30 FOR M=1 TO 36:REM M IS MONTH NUMBER
40 LET I=.01*B :REM CALCULATE INTEREST FOR MONTH
50 LET R=P-I :REM CALCULATE REPAYMENT
60 LET B=B-R :REM CALCULATE NEW BALANCE
70 PRINT I,B
80 NEXT M
90 END

```

You should attempt to run this program. Notice that it runs, but it is pretty useless because the screen will not contain all of the output. Most of the output goes flying by before you can read it. One method for remedying this situation is to press* **CTRL** and **C** as the output scrolls by on the screen. This will pause execution of the program and freeze the contents of the screen. To resume execution and unfreeze the screen, type **CONT** and hit the **RETURN** key. The output will begin to scroll again. To use this technique requires some manual dexterity. Moreover, it is not possible to guarantee where the scrolling will stop.

TEST YOUR UNDERSTANDING 5

RUN the program of Example 3 and practice freezing the output on the screen. It may take several runs before you are comfortable with the procedure.

Let us now describe another method of adapting the output to our screen size by printing only 12 months of data at one time. This amount of data will fit since the screen contains 24 lines. We will use a second loop to keep track of 12 month periods. The variable for the new loop will be *Y* (for "years"), and *Y* will go from 0 to 2. The month variable will be *M* as before, but now *M* will go only from 1 to 12. The month number

*The **CTRL** key should be depressed first and then held down while the **C** is depressed.

will now be $12*Y + M$ (the number of years plus the number of months). Here is the revised program.

```

10 LET B=7000
15 LET P=232.50
20 FOR Y=0 TO 2
30 PRINT "INTEREST","BALANCE"
40 FOR M=1 TO 12
50 LET I=.01*B
60 LET R=P-I: REM ONE 12 MONTH PERIOD
70 LET B=B-R
80 PRINT I,B
90 NEXT M
100 STOP :REM HALTS EXECUTION
110 HOME :REM CLEAR SCREEN
120 NEXT Y :REM GOES TO NEXT 12 MONTHS
130 END

```

This program utilizes several new statements. In line 100, we use the **STOP** statement. This causes the computer to stop execution of the program. The computer remembers where it stops, however, and all values of the variables are preserved. The **STOP** statement also leaves unchanged the contents of the screen. You can take as long as you wish to examine the data on the screen. When you are ready for the program to continue, type **CONT**. The computer will resume where it left off. The first instruction it encounters is in line 110. **HOME** clears the screen. So, after being told to continue, the computer clears the screen and goes on to the next value of Y—the next 12 months of data. Here is a copy of the output. The underlined statements are those you type.

]

RUN

INTEREST	BALANCE
70	6837.5
<u>68.375</u>	<u>6673.375</u>
66.73375	65076.0875
65.0760875	6340.18484
63.4018484	6171.08669
61.7108669	6000.29755
60.0029755	5827.80053
58.2780053	5653.57854
56.5357854	5477.61432
54.7761432	5299.89047
52.9989047	5120.38937
51.2038937	4939.09326

BREAK IN 100

]

CONT

INTEREST	BALANCE
49.3909326	4755.98419
47.5598419	4571.04404
45.7104404	4384.25448
43.8425448	4195.59702
41.9559702	4005.05299
40.0505299	3812.60352
38.1260352	3618.22955
36.1822955	3421.91185
34.2191185	3223.63097
32.2363097	3023.36728
30.2336728	2821.10095
28.2110095	2616.81196

BREAK IN 100

]

CONT

INTEREST	BALANCE
26.1681196	2410.48008
24.1048008	2202.08488
22.0208488	1991.60573
19.9160573	1779.02179
17.7902179	1564.31201
15.6431201	1347.45513
13.4745513	1128.42968
11.2842968	907.213974
9.07213974	683.7856113
6.83786113	458.123975
4.58126	230.205214
2.30205215	7.26658106E-03

BREAK IN 100

]

Note that the data in the output is carried out to ten figures, even though the problem deals with dollars and cents. We will look at the problem of rounding numbers later. Also note the balance listed at the end of month 36. It is in scientific notation. The -03 indicates that the decimal point is to be moved three places to the left. The number listed is .00726658106 or about .72 cents (less than one cent)! The computer shifted to scientific notation since the usual notation (.00726658106) requires more than ten digits. The computer made the choice of which form of the number to display.

USING LOOPS TO CREATE DELAYS

By using a loop we can create a delay inside the computer. Consider the following sequence of instructions:

```
10 FOR N=1 TO 3000
20 NEXT N
```

This loop doesn't do anything! However, the computer repeats instructions 10 and 20 three thousand times! This may seem like a lot of work. But not for a computer. To obtain a feel for the speed at which the computer works, you should time this sequence of instructions. Such a loop may be used as a delay. For example, when you wish to keep some data on the screen without stopping the program, just build in a delay. Here is a program which prints two screens of text. A delay is imposed to give a person time to read the first screen.

```
10 PRINT "THIS IS A PROGRAM TO DISPLAY SALES"
20 PRINT "FOR THE YEAR TO DATE"
30 FOR N=1 TO 5000
40 NEXT N: REM DELAY LOOP
50 HOME
60 PRINT "YOU MUST SUPPLY THE PARAMETERS:"
70 PRINT "PRODUCT, TERRITORY, VOLUME"
80 END
```

Example 4. Use a loop to produce a blinking display for a security system.

Solution. Suppose that your security system is tied in with your computer and the system detects that an intruder is in your warehouse. Let us print out the message:

```
SECURITY SYSTEM DETECTS INTRUDER-ZONE 2
```

For attention, let us blink this message on and off by alternately printing the message and clearing the screen.

```
10 FOR N=1 TO 2000
20 PRINT:PRINT:PRINT
30 PRINT "SECURITY SYSTEM DETECTS INTRUDER-ZONE 2"
40 FOR K=1 TO 150
50 NEXT K
60 HOME
70 FOR K=1 TO 150
80 NEXT K
90 NEXT N
100 END
```

The loop in 40-50 is a delay loop to keep the message on the screen a moment. Line 60 turns the message off, but the PRINT statement in line

30 almost immediately turns it back on. The message will blink 2000 times.

TEST YOUR UNDERSTANDING 6 (answer on page 42)

Write a program which blinks your name on the screen 500 times, leaving your name on the screen for a loop of length 50 each time.

In all of our loop examples, the loop variable increased by one with each repetition of the loop. However, it is possible to have the loop variable change by any amount. For example, the instructions

```
10 FOR N=1 TO 5000 STEP 2
:
:
:
1000 NEXT N
```

define a loop in which N jumps by 2 for each repetition, so N will assume the values:

1, 3, 5, 7, 9, . . . , 4999

Similarly, use of STEP .5 in the above loop will cause N to advance by .5 and assume the values:

1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, . . . , 5000

It is even possible to have a negative step. In this case, the loop variable will run backwards. For example, the instructions

```
10 FOR N=100 TO 1 STEP -1
:
:
:
100 NEXT N
```

will "count down" from N = 100 to N = 1 one unit at a time. We will give some applications of such instructions in the Exercises.

TEST YOUR UNDERSTANDING 7 (answers on page 42)

Write instructions to allow N to assume the following sequences of values:

a. 95, 96.7, 98.4, . . . , 112

b. 200, 199.5, 199, . . . , 100

EXERCISES (answers on page 221)

Write Applesoft BASIC programs to compute the following quantities.

1. $1^2 + 2^2 + 3^2 + \dots + 25^2$
2. $1 + (1/2) + (1/2)^2 \dots + (1/2)^{10}$
3. $1^3 + 2^3 + 3^3 + \dots + 10^3$
4. $1 + (1/2) + (1/3) + \dots + (1/100)$
5. Write a program to compute N^2 , N^3 , N^4 for $N=1, \dots, 12$. The format of your output should be as follows:

N	N^2	N^3
1		
2		
3		
.		
.		
.		
12		

6. Suppose that you have a car loan whose current balance is \$4,000.00. The monthly payment is \$125.33 and the interest is one percent per month on the unpaid balance. Make a table of the interest payments and balances for the next 12 months.
7. Suppose you deposit \$1,000 on January 1 of each year into a savings account paying 10 percent interest. Suppose that the interest is computed on January 1 of each year, based on the balance for the preceding year. Calculate the balances in the account for each of the next 15 years.
8. A stock market analyst predicts that Tyro Computers, Inc. will achieve a 20 percent growth in sales in each of the next three years, but profits will grow at a 30 percent annual rate. Last year's sales were \$35 million and last year's profits were \$5.54 million. Project the sales and profits for the next three years, based on the analyst's prediction.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, 3, 4, 6, and 7

- | | |
|---|---|
| <p>1: a. 10 FOR N=3 TO 77
 .
 .
 100 NEXT N</p> | <p>b. 10 FOR N=3 TO 77
 20 PRINT N^2
 30 NEXT N
 40 END</p> |
|---|---|

2: The heading

N N²

would be printed before each entry of the table.

3: 10 S=0

20 FOR N=101 TO 110

30 S=S+N

40 NEXT N

50 PRINT S

60 END

4: 10 FOR N=1 TO 20

20 PRINT 2^N

30 NEXT N

40 END

6: 10 FOR N=1 TO 500

20 PRINT "(YOUR NAME)"

30 FOR K=1 TO 50

40 NEXT K

50 HOME

60 FOR K=1 TO 50:NEXT K

70 NEXT N

80 END

7: a. 10 FOR N=95 TO 112 STEP 1.7

b. 20 FOR N=200 TO 100 STEP -.5

2.5 SOME APPLESOFT BASIC COMMANDS

Thus far, most of our attention has been focused on learning statements to insert inside programs. Let us now learn a few of the commands available for manipulating programs and the computer. The **NEW** command, previously discussed, is in this category. Remember the following facts about Applesoft BASIC commands:

1. Commands are typed *without* using a line number.
2. You must hit the **RETURN** key after typing a command.
3. A command may be given whenever the computer is in command mode. (Recall that when the computer first enters the

command mode, it displays the] message. The computer remains in the command mode until a **RUN** command is given.)

4. The computer executes commands as soon as they are received.

LISTING A PROGRAM

To obtain a list of all statements of the current program in RAM, you may type the command

LIST

For example, suppose that RAM contains the following program.

```
10 PRINT 5+7, 5-7
20 PRINT 5*7,5/7
30 END
```

(This program may or may not be currently displayed on the screen.) If you type **LIST**, then the above three instruction lines will be displayed, followed by the] message.

In developing a program, you will undoubtedly find it necessary to input lines in non-consecutive order and to correct lines already input. If this happens, the screen will usually not indicate the current version of the program. Typing **LIST** every so often will assist in keeping track of what has been changed. **LISTing** is particularly helpful in checking a program or determining why a program won't run.

Note that the Apple II Computer screen can display up to 24 lines of text. This means you can display, at most, 24 program statements at one time. Therefore, you must list long programs a section at a time. For example, to **LIST** only those statements with line numbers from 100 to 240, we use the command:

LIST 100-240

In a similar fashion, we may list any collection of consecutive program lines.

There are several other variations of the **LIST** command. To list the program lines from the beginning of the program to line 75, use the command

LIST -75

Similarly, to list the program lines from 100 to the end of the program, use the command

LIST 100-

To list line 100, use the command

LIST 100

TEST YOUR UNDERSTANDING 1 (answers on page 46)

Write a command to:

- a. List line 200
- b. List lines 300–330
- c. List lines 300 to the end

Test out these commands with a program.

DELETING PROGRAM LINES

When typing a program or revising an existing program, it is often necessary to delete lines which are already part of the program. One simple way is to type the line number followed by **RETURN**. For example,

275

(followed by hitting the **RETURN** key) will delete line 275. The **DEL** command may also be used for the same purpose. For example, we may delete line 275, using the command

DEL 275,275

Similarly, to delete the lines 500 to 700 inclusive, use the command

DEL 500,700

If the program does not have a line 700, the computer will determine the last line before 700 and will delete from 500 to that line, inclusive.

TEST YOUR UNDERSTANDING 2 (answers on page 46)

What is wrong with the following commands?

- a. DEL 450,
- b. LIST 450-
- c. DEL 300,200

SAVING A PROGRAM

Once you have typed a program into RAM, you may save a copy on cassette or diskette. At any future time, you may read the cassette or diskette copy back into RAM. At that point, you may re-execute the program, modify it, or add to it. For the sake of concreteness, suppose that the following program is in RAM:

```
10 PRINT 5+7  
20 END
```

Saving a Program on Diskette. We must first assign the program a name, which may consist of any string of up to 30 letters or numbers. Suppose that we choose the name RETAIN for our program. We may save this program on the diskette in either disk drive. To save RETAIN on drive 1, for example, we would use the command:

```
SAVE RETAIN, D1
```

When the computer finishes writing a copy of the program onto the designated diskette, it will display the] prompt. Saving a program does not alter the copy of the program in RAM.

To read a program from diskette into RAM, we use the **LOAD** command. For example, to read RETAIN from the diskette in drive 2, we use the command:

```
LOAD RETAIN, D2
```

Important Note. In order to **SAVE** or **LOAD** programs on diskette, it is first necessary to initialize the Disk Operating System (DOS). To see how this is done, consult Chapter 4.

Saving a Program on Cassette. To save RETAIN on cassette, we position the tape on a blank segment, and push the **PLAY** and **RECORD** buttons of the cassette recorder simultaneously. Next, type the command

```
SAVE
```

The program will then be written onto the cassette. You should use the tape counter on the cassette recorder to identify the beginning of the program on the cassette.

To read the program back into RAM, rewind the tape and position it at the start of the program (use the tape counter to find the right spot), push the **PLAY** button on the cassette recorder, and type the command

```
LOAD
```

Note that the cassette recorder does not allow you to identify a program by a program name.

You should try the above sequence of commands using the given program. After saving the program, erase the program from RAM (by typing **NEW**). Then load the program. Just to check that the program has indeed been retrieved, you should now **LIST** it.

EXERCISES

Exercises 1–7 refer to the following program.

```
10 LET A=19.1: B=17.5
20 PRINT A+B,A*B
30 END
```

1. Type the above program into RAM and RUN it.
2. Erase the screen without erasing RAM. LIST the program.
3. Save the program and erase RAM.
4. Recall the program and LIST it. RUN the program again.
5. Add the following line to the program:

```
25 PRINT A^2+B^2
```

Do not retype the entire program! LIST and RUN the new program.

6. Save the new program without destroying the old one.
7. Recall the new program. Delete line 20 and RUN the resulting program.

ANSWERS TO TEST YOUR UNDERSTANDING 1 and 2

- 1: a. LIST 200
b. LIST 300-330
c. LIST 300-
- 2: a. The line number of the last line to be deleted must be specified. It should read:
DEL 405,450
b. Nothing wrong.
c. The lower line number must come first. The command should read:
DEL 200,300

2.6 LETTING YOUR COMPUTER MAKE DECISIONS

One of the principal features which makes computers useful as problem-solving tools is their ability to make decisions. The vehicle which Applesoft BASIC uses to make decisions is the **IF . . . THEN** statement. The **IF** part of such a statement allows us to ask a question. If the answer is YES, then the computer carries out the **THEN** part of the statement. If the answer is NO, then the computer goes on to the next statement in numerical order. For example, consider the statement

```
500 IF N=0 THEN PRINT "CALCULATION DONE"
```

First, the computer determines if N is equal to zero. If so, it prints "CALCULATION DONE" and proceeds with the next instruction after line 500. However, if N is not zero, then the computer goes directly to the next instruction after line 500 and continues program execution from that instruction.

A variation of the **IF . . . THEN** statement allows you to insert a line number after **THEN**. For example, the instruction

```
600 IF N>0 THEN GOTO 500
```

will determine whether N is greater than 0. If so the program will go to line 500. Otherwise, the program will go to the next line in numerical sequence. Notice the **GOTO** after **THEN**. This may be omitted. In any case, putting the **GOTO** in will never hurt anything.

After **IF**, you may insert any expression which the computer may test for truth or falsity. Here are some examples:

```
N = 0
N > 5 (N is greater than 5)
N < 12.9 (N is less than 12.9)
N >= 0 (N is greater than or equal to 0)
N <= -1 (N is less than or equal to -1)
N >< 0 (N is not equal to 0)
A + B <> C (A + B is not equal to C)
A^2 + B^2 <= C^2 (A^2 + B^2 is less than or equal to C^2)
```

NOTE: Any statement which follows on the same line as an **IF . . . THEN** statement will be executed only if the **THEN** part is executed. For example, in the statement

```
10 IF A<B THEN C=D: GOTO 300
20 C=E
```

the computer will do the following: If A is less than B, the computer will set C equal to D. It will then go to 300. On the other hand if A is not less than B, the computer will go to the statement on line 20.

TEST YOUR UNDERSTANDING 1 (answers on page 57)

Write instructions which do the following:

- a. If A is less than B, then print the value of A plus B; if not, then go to the end.
- b. If $A^2 + D$ is at least 5000 then go to line 300; if not, go to line 500.
- c. If N is larger than the sum of I and K, then set N equal to the sum of I and K; otherwise, let N equal K.

The **IF . . . THEN** statement may be used to interrupt the normal sequence of program line execution, based upon the truth or falsity of some condition. In many applications, however, we will want to perform instructions out of the normal sequence, independent of any conditions. For such applications, we may use the **GOTO** instruction. This instruction has the form:

GOTO <line number>

For example, the instruction

1000 GOTO 300

will send the computer back to line 300 for its next instruction.

The next examples illustrate some of the uses of the **IF . . . THEN** and **GOTO** statements.

Example 1. A lumber supply house has a policy that a credit invoice may not exceed \$1,000, including a 10 percent processing fee and 5 percent sales tax. A customer orders 150 2×4 studs at \$1.99 each, 30 sheets of plywood at \$14.00 each, 300 pounds of nails at \$1.14 per pound, and two double hung insulated windows at \$187.95 each. Write a program which prepares an invoice and decides whether the order is over the credit limit.

Solution. Let's use the variables A1, A2, A3, and A4 to denote, respectively, the numbers of studs, sheets of plywood, pounds of nails, and windows. Let's use the variables B1, B2, B3, and B4 to denote the unit costs of these four items. The cost of the order is then computed as:

$$A1*B1+A2*B2+A3*B3+A4*B4.$$

We add 10 percent of this amount to cover processing and form the sum to obtain the total order. Next, we compute 5 percent of the last amount as tax and add it to the total to obtain the total amount due. Finally, we determine if the total amount due is more than \$1,000. If it is, we print out the message: ORDER EXCEEDS \$1,000. CREDIT SALE NOT PERMITTED. Here is our program.

```

10 LET A1=150:A2=30:A3=300:A4=2
20 LET B1=1.99:B2=14:B3=1.14:B4=189.75
30 LET T=A1*B1+A2*B2+A3*B3+A4*B4
40 PRINT "TOTAL ORDER",T
50 LET P=.1*T
60 PRINT "PROCESSING FEE",P
70 LET TX=.05*(P+T)
80 PRINT "SALES TAX",TX
90 DU=T+P+TX
100 PRINT "AMOUNT DUE", DU
110 IF DU>1000 THEN GOTO 200
120 GOTO 300
200 PRINT "ORDER EXCEEDS $1,000"
210 PRINT "CREDIT SALE NOT PERMITTED"
220 GOTO 400
300 PRINT "CREDIT SALE OK"
400 END

```

Note the decision in line 110. If the amount due exceeds \$1,000 then the computer goes to line 200 where it prints out a message denying credit. In line 220, the computer is sent to line 400 which is the END of the program. On the other hand, if the amount due is less than \$1,000, the computer is sent to line 300, in which credit is approved.

TEST YOUR UNDERSTANDING 2 (answers on page 57)

Suppose that a credit card charges 1.5 percent per month on any unpaid balance up to \$500 and 1 percent per month on any excess over \$500.

- Write a program which computes the service charge and the new balance.
- Test your program on the unpaid balances of \$1,300 and \$275.

TEST YOUR UNDERSTANDING 3 (answers on page 57)

Consider the following sequence of instructions.

```

100 IF A>=5 THEN GOTO 200
110 IF A>=4 THEN GOTO 300

```

```
120 IF A>=3 THEN GOTO 400
130 IF A>=2 THEN GOTO 500
```

Suppose that the current value of A is 3. List the sequence of line numbers which will be executed.

Example 2. At \$20 per square yard, a family can afford up to 500 square feet of carpet for their dining room. They wish to install the carpet in a circular shape. It is been decided that the radius of the carpet is to be a whole number of feet. What is the radius of the largest carpet they can afford? (The area of a circle of radius "R" is PI times R², where PI equals approximately 3.14159.)

Solution. Let us compute the area of the circle of radius 1, 2, 3, 4, . . . and determine which of the areas are less than 500.

```
10 LET PI=3.14159
20 LET R=1 : REM R IS THE RADIUS OF THE CIRCLE
30 LET A=PI*R^2 : REM A IS THE AREA OF THE CIRCLE
40 IF A>=500 THEN GOTO 100 : REM IF AREA IS AT LEAST 500, END
50 PRINT R : REM IF AREA IS LESS THAN 500, PRINT R
60 LET R=R+1 : REM GO TO NEXT RADIUS
70 GOTO 30
100 END
```

Note that line 40 contains an **IF . . . THEN** statement. If A, as computed in line 30, is 500 or more, then the computer goes to line 100, the **END**. If A is less than 500, the computer proceeds to the next line, namely 50. It then prints out the current radius, increases the radius by 1, and goes back to line 30 to repeat the entire procedure. Note that lines 30-40-50-60-70 are repeated until the area becomes at least 500. In effect, this sequence of five instructions forms a loop. However, we did not use a **FOR . . . NEXT** instruction because we did not know in advance how many times we wanted to execute the loop. We let the computer decide the stopping point via the **IF . . . THEN** instruction.

Example 3. A school board race involves two candidates. The returns from the four wards of the town are as follows:

	Ward 1	Ward 2	Ward 3	Ward 4
Mr. Thompson	487	229	1540	1211
Ms. Wilson	1870	438	110	597

Calculate the total number of votes achieved by each candidate, the percentage achieved by each candidate, and decide who won the election.

Solution. Let $A_1, A_2, A_3,$ and A_4 be the totals for Mr. Thompson in the four wards; let $B_1 - B_4$ be the corresponding numbers for Ms. Wilson. Let T_A and T_B denote the total votes, respectively, for Mr. Thompson and Ms. Wilson. Here is our program:

```

10 LET A1=487: A2=229: A3=1540: A4=1211
20 LET B1=1870: B2=438: B3=110: B4=597
30 LET TA=A1+A2+A3+A4 : REM TOTAL FOR THOMPSON
40 LET TB=B1+B2+B3+B4 : REM TOTAL FOR WILSON
50 LET T=TA+TB : REM TOTAL VOTES CAST
60 LET PA=100*TA/T : REM PERCENTAGE FOR THOMPSON
65 REM TA/T IS THE FRACTION OF VOTES FOR THOMPSON- MULTIPLY
66 REM BY 100 TO CONVERT TO A PERCENTAGE
70 LET PB=100*TB/T : REM PERCENTAGE FOR WILSON
110 LET A$= "THOMPSON"
120 LET B$= "WILSON"
130 REM 140-170 PRINT THE PERCENTAGES OF THE CANDIDATES
140 PRINT "CANDIDATE","VOTES","PCT"
150 PRINT A$,TA, PA
155 PRINT
160 PRINT B$,TB, PB
165 PRINT
170 REM 180-400 DECIDE THE WINNER
180 IF TA>TB THEN GOTO 300
190 IF TA<TB THEN GOTO 400
200 PRINT "TIE VOTE!"
210 GOTO 1000
300 PRINT A$, "WINS"
305 PRINT
310 GOTO 1000
400 PRINT B$, "WINS"
410 PRINT
1000 END

```

Note the logic used for deciding who won. In line 180 we compare the votes T_A and T_B . If T_A is the larger, then A (Thompson) is the winner. We then go to 300, print the result, and END. On the other hand, if $T_A > T_B$ is *false*, then either B wins or the two are tied. According to the program, if $T_A > T_B$ is false, we go to line 190, where we determine if $T_A < T_B$. If this is true, then B is the winner, we go to 400, print the result, and END. On the other hand, if $T_A < T_B$ is false, then the only possibility left is that $T_A = T_B$. According to the program, if $T_A = T_B$ we go to 200, where we print the proper result, and then END.

INFINITE LOOPS AND CTRL-C

As we have seen above, it is very convenient to be able to execute a loop without knowing in advance how many times the loop will be executed. However, with this convenience comes a danger. It is per-

fectly possible to create a loop which will be repeated an infinite number of times! For example, consider the following program:

```
10 LET N=1
20 PRINT N
30 LET N=N+1
40 GOTO 20
50 END
```

The variable N starts off at 1. We print it and then increase N by 1 (to 2), print it, increase N by 1 (to 3), print it, and so forth. This program will go on forever! Such programs should clearly be avoided. However, even experienced programmers occasionally create infinite loops. When this happens, there is no need to panic. There is a way of stopping the computer. Just press the keys **CTRL** and **C** simultaneously. (In the following we will refer to this combination of keys as the **CTRL C** combination. This key sequence will interrupt the program currently in progress and return the computer to command mode. The computer is then ready to accept a command from the keyboard. Note, however, that any program in RAM is undisturbed.

TEST YOUR UNDERSTANDING 4

Type the above program, RUN it and stop it using CTRL-C. After stopping it, RUN the program again.

THE INPUT STATEMENT

It is very convenient to have the computer request information from you while the program is actually running. This can be accomplished via the **INPUT** statement. To see how, consider the statement

```
570 INPUT A
```

When the computer encounters this statement in the course of executing the program, it types out a ? and waits for you to respond by typing the desired value of A (and then hitting the **RETURN** key). The computer then sets A equal to the numeric value you specified and continues running the program.

You may use an **INPUT** statement to specify the values of several different variables at one time. These variables may be numeric or string variables. For example, suppose that the computer encounters the statement:

```
50 INPUT A,B,C$
```

It will type

?

You then type in the desired values for A, B, and C\$, in the same order as in the program, and separated by commas. For example, suppose that you type

10.5, 11.42, BEARINGS

followed by a **RETURN**. The computer will then set:

A = 10.5, B = 11.42, C\$ = "BEARINGS"

If you respond to the above question mark by typing only a single number, 10.5, for example, the computer will respond with

??

to indicate that it expects more data. If you attempt to specify a string constant where you should have a numeric constant, the computer will respond with the message

? REENTER

and will wait for you to repeat the **INPUT** operation.

It is helpful to include a prompting message which describes the input the computer is expecting. To do so, just put the message in quotation marks after the word **INPUT** and place a semicolon after the message (before the list of variables to be input). For example, consider the statement

175 INPUT "ENTER COMPANY, AMOUNT "; A\$, B

When the computer encounters this program line, the dialog will be as follows:

ENTER COMPANY, AMOUNT? AJAX OFFICE SUPPLIES, 2579.48

The underlined portion indicates your response to the prompt. The computer will now assign the values:

A\$ = "AJAX OFFICE SUPPLIES", B = 2579.48

TEST YOUR UNDERSTANDING 5 (answer on page 57)

Write a program which allows you to set variables A and B to any desired values via an **INPUT** statement. Use the program to set A equal to 12 and B equal to 17.

The next two examples illustrate the use of the **INPUT** statement, and provide further practice in using the **IF . . . THEN** statement.

Example 4. You are a teacher compiling semester grades. Suppose there are four grades for each student and that each grade is on the traditional 0 to 100 scale. Write a program which accepts the grades as input, computes the semester average, and assigns grades according to the following scale:

90–100	A
80–89.9	B
70–79.9	C
60–69.9	D
<60	F

Solution. We will use an **INPUT** statement to enter the grades into the computer. Our program will allow you to compute the grades of students, one after the other, via a loop. You may terminate the loop by entering a negative grade. Here is our program.

```

5 HOME
10 PRINT "ENTER STUDENT'S 4 GRADES."
20 PRINT "SEPARATE GRADES BY COMMAS."
30 PRINT "FOLLOW LAST GRADE WITH RETURN."
40 PRINT "TO END PROGRAM, ENTER NEGATIVE GRADE."
50 INPUT A1,A2,A3,A4
60 IF A1<0 THEN 200
70 IF A2<0 THEN 200
80 IF A3<0 THEN 200
90 IF A4<0 THEN 200
100 LET A=(A1+A2+A3+A4)/4
110 PRINT "SEMESTER AVERAGE", A
120 IF A>=90 THEN PRINT "SEMESTER GRADE = A" : GOTO 10
130 IF A>=80 THEN PRINT "SEMESTER GRADE = B" : GOTO 10
140 IF A>=70 THEN PRINT "SEMESTER GRADE = C" : GOTO 10
150 IF A>=60 THEN PRINT "SEMESTER GRADE = D" : GOTO 10
160 PRINT "SEMESTER GRADE = F"
170 GOTO 10
200 END

```

Note the logic for printing out the semester grades. First compute the semester average A . In line 120 we ask if A is greater than or equal to 90. If so, we assign the grade A, and go to line 10. In case A is less than 90 line 120 sends us to line 130. In line 130, we ask if A is greater than or equal to 80. If so, then we assign the grade B. (The point is that the only way we can get to line 130 is for A to be less than 90. So if A is greater than or equal to 80, we know that A lies in the B range.) If not, we go to line 140, and so forth. This logic may seem a trifle confusing at first, but after repeated use, it will seem quite natural.

Example 5. Write a program to maintain your checkbook. The program should allow you to record an initial balance, enter deposits, and enter checks. It should also warn you of overdrafts.

Solution. Let the variable B always contain the current balance in the checkbook. The program will ask for the type of transaction you wish to record. A "D" will mean that you wish to record a deposit; a "C" will mean that you wish to record a check; a "Q" will mean that you are done entering transactions and wish to terminate the program. After entering each transaction, the computer will figure your new balance, report it to you, will check for an overdraft, and report any overdraft to you. In case of an overdraft, the program will allow you to cancel the preceding check!

```

10 INPUT "WHAT IS YOUR STARTING BALANCE "; B
20 INPUT "WHAT TRANSACTION TYPE (D,C,OR Q) "; A$
30 IF A$= "Q" THEN 1000
40 IF A$= "D" THEN INPUT "DEPOSIT AMOUNT "; D;GOTO 110
100 IF A$= "C" THEN 200
110 LET B=B+D : REM ADD DEPOSIT TO BALANCE
120 PRINT "YOUR NEW BALANCE IS", B
130 GOTO 20
200 INPUT "CHECK AMOUNT"; C
210 LET B=B-C : REM DEDUCT CHECK AMOUNT
220 IF B<0 THEN GOTO 300 : REM TEST FOR OVERDRAFT
230 PRINT "YOUR NEW BALANCE IS ", B
240 GOTO 20
300 PRINT "LAST CHECK CAUSES OVERDRAFT"
310 INPUT "DO YOU WISH TO CANCEL CHECK(Y/N) "; E$
320 IF E$= "Y" THEN GOTO 400
330 PRINT "YOUR NEW BALANCE IS", B
340 GOTO 20
400 LET B=B+C: REM CANCEL LAST CHECK
410 GOTO 20
1000 END

```

You should scan this program carefully to make sure you understand how each of the **INPUT** and **IF . . . THEN** statements is used. In addition, you should use this program to obtain a feel for the dialog between you and your computer when **INPUT** statements are used.

Example 6. Write an Applesoft BASIC program which tests mastery in addition of two-digit numbers. Let the user suggest the problems, and let the program keep score of the number correct out of ten.

Solution. Let us request that the program user suggest pairs of numbers via an **INPUT** statement. The sum will also be requested via an **INPUT** statement. An **IF . . . THEN** statement will be used to judge the correctness. The variable R will keep track of the number correct. We will use a loop to repeat the process ten times.

```

10 FOR N=1 TO 10 : REM LOOP TO GIVE 10 PROBLEMS
20 INPUT "TYPE TWO 2-DIGIT NUMBERS"; A,B
30 INPUT "WHAT IS THEIR SUM"; C

```

```

40 IF A+B=C THEN GOTO 200
50 PRINT "SORRY. THE CORRECT ANSWER IS",A+B
60 GO TO 500 : REM GO TO THE NEXT PROBLEM
200 PRINT "YOUR ANSWER IS CORRECT! CONGRATULATIONS"
210 LET R=R+1 : REM INCREASE SCORE BY 1
500 NEXT N
600 PRINT "YOUR SCORE IS",R,"CORRECT OUT OF 10"
700 PRINT "TO TRY AGAIN, TYPE RUN"
800 END

```

EXERCISES (answers on page 223)

1. Write a computer program to calculate all the perfect squares which are less than 45,000. (Perfect squares are the numbers 1, 4, 9, 16, 25, 36, 49,)
2. Write a computer program to determine all of the circles of integer radius and area less than or equal to 5,000 square feet. (The area of a circle of radius R is $PI * R^2$, where $PI = 3.14159$ approximately.)
3. Write a computer program to determine the sizes of all those boxes which are perfect cubes, have integer dimensions, and have volumes of less than 175,000 cubic feet. (That is, find all integers X for which X^3 is less than 175,000.)
4. Modify the arithmetic testing program of Example 4 so that the operation tested is for multiplication instead of addition.
5. Modify the arithmetic testing program of Example 4 so that it allows you to choose, at the beginning of each group of ten problems, from among these operations: addition, subtraction, or multiplication.
6. Write a program which accepts three numbers via an INPUT statement and determines the largest of the three.
7. Write a program which accepts three numbers via an INPUT statement and determines the smallest of the three.
8. Write a program which accepts a set of numbers via INPUT statements and determines the largest among them.
9. Write a program which accepts a set of numbers via INPUT statements and determines the smallest among them.
10. The following data were collected by a sociologist. Six cities experienced the following numbers of burglaries in 1980 and 1981:

City	Burglaries 1980	Burglaries 1981
1	5,782	6,548
2	4,811	6,129
3	3,865	4,270
4	7,950	8,137
5	4,781	4,248
6	6,598	7,048

For each city, calculate the increase (decrease) in the number of burglaries. Determine which had an increase of more than 500 burglaries.

11. Write a program which does the arithmetic of a cash register. That is, let the program accept purchases via INPUT statements, then total the purchases, figure out the sales tax (assume 5 percent), and compute the total purchase. Let the program ask for the amount of payment given and then let it compute the change due.
12. Write a program which analyzes cash flow. Let the program ask for cash on hand as well as accounts expected to be received in the next month. Let the program also compute the total anticipated cash for the month. Let the program ask for the bills due in the next month, and let it compute the total accounts payable during the month. By comparing the amounts to be received and to be paid out, let the program compute the net cash flow for the month and report either a surplus or a deficit.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, 3, and 5

- 1: a. 10 IF A<B THEN PRINT A+B : END
 b. 10 IF A^2+D>=5000 THEN GOTO 300
 20 GOTO 500
 c. 10 IF N>I+K THEN N=I+K
 20 IF N<=I+K THEN N=K
2. 10 INPUT "UNPAID BALANCE";B
 20 IF B>500 THEN GOTO 100
 30 GOTO 200
 100 LET C=B-500
 110 IN=.015*500+.01*C
 120 GOTO 300
 200 IN=.015*B
 300 PRINT "INTEREST EQUALS";IN

```

310 PRINT "NEW BALANCE EQUALS";B+IN
320 END

3: 100-110-120-400

5: 10 PRINT "THE VALUES OF A AND B ARE":INPUT A,B
20 END

```

2.7 SOME PROGRAMMING TIPS

Now that we have learned the most elementary Applesoft BASIC commands and statements, let us discuss a few topics which will make programming easier.

TWO SHORTCUTS

Here are two shortcuts which will save time in typing programs.

1. It is not necessary to include the word **LET** in a **LET** statement! The statement

```
10 A=5
```

means the same thing to the computer as

```
10 LET A=5
```

2. A question mark may be used in place of the word **PRINT**. Therefore, the statement

```
10 ? A, A$
```

means the same thing as the statement

```
10 PRINT A,A$
```

TEST YOUR UNDERSTANDING 1 (answer on page 61)

What is the output of the following program?

```

10 A=3: B=7
20 A=2*B+3*A
30 ? A,B^2
40 END

```

USING A PRINTER

In writing programs and analyzing their output, it is often easier to rely on written output rather than output on the screen. In computer terminology; written output is called **hard copy** and may be provided by a wide variety of printers. Your Apple II computer may be attached to a large number of such printers, ranging from a dot-matrix thermal printer costing only a few hundred dollars to a daisy wheel printer costing several thousand dollars. As you begin to make serious use of your computer, you will find it difficult to do without hard copy.

Indeed, writing programs is much easier if you can consult a hard copy listing of your program at various stages of program development. (One reason is that in printed output you are not confined to looking at your program in 10–14 line “snapshots.”) Also, you will want to use the printer to produce output of programs, ranging from tables of numerical data to address lists and text files produced via a word processing program.

To use a printer, it is necessary to install a printer interface card in one of the unused slots (#1–#7), Slot #1 is usually used for the printer. In what follows, we will assume that this is the case. To use the printer, you must first redirect output from the screen to the printer via the statement

```
10 PR #1
```

Now, you may produce hard copy on your printer by using the Apple-soft BASIC statement **PRINT**. For example, the statement

```
10 PRINT A,A$
```

will print the current values of A and A\$ on the printer, in print fields 1 and 2. (As is the case with the screen, Applesoft BASIC divides the printer line into print fields which are 16 columns wide.) Moreover, the statement

```
20 PRINT "Customer","Credit Limit","Most Recent Pchs"
```

will result in printing three headings in the first three print fields, namely:

```
Customer      Credit Limit      Most Recent Pchs
```

To return output to the screen only, use the command

```
30 PR #0
```

Printing on the printer proceeds very much like printing on the screen. It is important to realize, however, that in order to print on both the

screen *and* the printer, it may be necessary to use two **PRINT** statements.* For example, to print the values of A and A\$ on both the screen and the printer, we must give instructions as follows:

```
10 PRINT A,A$
20 PR #1
30 PRINT A,A$
40 PR #0
```

The last statement returns all subsequent output to the screen.

SOME THINGS TO CHECK

Writing programs in Applesoft BASIC is not difficult. However, it does require a certain amount of care and meticulous attention to detail. Each person must develop an individual programming style.

Here are a few tips which may help the novice programmer over some of the rough spots of writing those first few programs.

1. Carefully think your program through. Break up the computation into steps. Outline the programming necessary for each of the steps.
2. Work through your program by hand, pretending that you are the computer. Don't rush. Go through your program one step at a time and check that it does what you want it to do.
3. Have you given all variables the values you want? Remember, if you do not specify the value of a variable, Applesoft BASIC will automatically assign it the value 0. This may not be the value you intend!
4. Are all your loops complete? That is, have you included a **NEXT** corresponding to each **FOR**? This is an easy mistake to make, but it is also easy to catch. If Applesoft BASIC doesn't find a **NEXT** corresponding to a **FOR** when it attempts to run the program, it will report the mistake and the line number in which it occurs. This is just one of a series of checks which Applesoft BASIC makes for consistency and completeness. (Later, we will discuss the various error messages which Applesoft BASIC can provide.) **FOR . . . NEXT** loops may be contained in one an-

*It will depend on your particular printer.

other. (They may be nested.) But the loop which starts earlier must end later. In other words, loops may not “cross” one another.

5. Check to see that your **IF . . . THEN** statements do not create any infinite loops. This may be a difficult error to spot. However, it can be located with the following check. When you go back to an earlier part of the program, ask yourself: what condition must be present if the program is not to keep doubling back forever? Is this condition guaranteed to occur?

In the upcoming chapters we will present some further ideas on debugging your programs and on programming technique. For now, however, let's move on with learning to make our computer do interesting things!

ANSWER TO TEST YOUR UNDERSTANDING 1

1: 23 49

3

More About Applesoft BASIC

In this chapter we will continue our introduction to Applesoft BASIC programming. As in Chapter Two, we will organize our discussion by application.

3.1 WORKING WITH TABULAR DATA

In the preceding chapter, we introduced the notion of a variable and used variable names like

AA, B1, CZ, WO

Unfortunately, the supply of variables available to us is not sufficient for many programs. Indeed, as we shall see in this chapter, there are relatively innocent programs which require hundreds or even thousands of variables. To meet the needs of such programs, Applesoft BASIC allows the use of so-called **subscripted variables**. Such variables are used constantly by mathematicians and are identified by numbered subscripts attached to a letter. For instance, here is a list of 1000 variables as they might appear in a mathematical work:

$A_1, A_2, A_3, \dots, A_{1000}$

The numbers used to distinguish the variables are called **subscripts**. Likewise, the Applesoft BASIC language allows definition of variables to be distinguished by subscripts. However, since the computer has difficulty placing the numbers in the traditional position, they are placed in parentheses on the same line as the letter. For example, the above list

of 1000 different variables would be written in Applesoft BASIC as:

```
A(1),A(2),A(3), . . . ,A(1000)
```

Please note that the variable A(1) is not the same as the variable A1. You may use both of them in the same program and Applesoft BASIC will interpret them as different.

A subscripted variable is really a group of variables with a common letter identification which is distinguished by different integer "subscripts." For instance, the above group of variables would constitute the subscripted variable A(). It is often useful to view a subscripted variable as a table or array. For example, the subscripted variable A() considered above can be viewed as providing the following table of information:

```
A(1)
A(2)
A(3)
.
.
.
A(1000)
```

As shown here, the subscripted variable defines a table consisting of 1000 rows. Row number J contains a single entry, namely, the value of the variable A(J). The first row contains the value of A(1), the second the value of A(2), and so forth. Since a subscripted variable can be thought of as a table (or array), subscripted variables are often called *arrays*.

The array shown is a table consisting of 1000 rows and a single column. The Apple II Computer allows you to consider more general arrays. For example, consider the following financial table which records the monthly income for January, February, and March from each of a chain of four dry cleaning stores:

	Store #1	Store #2	Store #3	Store #4
Jan.	1258.38	2437.46	4831.90	987.12
Feb.	1107.83	2045.68	3671.86	1129.47
March	1298.00	2136.88	4016.73	1206.34

This table has three rows and four columns. It's entries may be stored in the computer as a set of 12 of variables:

```
A(1,1) A(1,2) A(1,3) A(1,4)
A(2,1) A(2,2) A(2,3) A(2,4)
A(3,1) A(3,2) A(3,3) A(3,4)
```


This array of variables is very similar to a subscripted variable, except that there are now two subscripts. The first subscript indicates the row number and the second subscript indicates the column number. For example, the variable $A(3,2)$ is in the third row, second column. A collection of variables such as that given above is called a **two-dimensional array** or a **doubly-subscripted variable**. Each setting of the variables in such an array defines a tabular array. For example, if we assign the values

$$\begin{aligned} A(1,1) &= 1258.38, A(1,2) = 2437.46, \\ A(1,3) &= 4831.90 \end{aligned}$$

and so forth, then we will have the table of earnings from the dry cleaning chain.

So far, we have only considered numeric arrays—arrays whose variables can assume only numerical values. However, it is possible to have arrays with variables that assume string values. (Recall that a string is a sequence of characters: letter, numeral, punctuation mark, or other printable keyboard symbol.) For example, here is an array which can contain string data:

A\$(1)
A\$(2)
A\$(3)
A\$(4)

Here the dollar signs indicate that each of the variables of the array is a string variable. If we assign the values

$$A\$(1) = \text{"SLOW"}, A\$(2) = \text{"FAST"}, A\$(3) = \text{"FAST"}, A\$(4) = \text{"STOP"}$$

then the array is just the table of words

SLOW
FAST
FAST
STOP

Similarly, the employee record table

Social Security Number	Age	Sex
178654775	38	M
345861023	29	F
789257958	34	F
375486595	42	M
457696064	21	F

may be stored in an array of the form $B\$(I,J)$, where I assumes any one of the values 1, 2, 3, 4, 5 (I is the row), and J assumes any one of the values 1, 2, 3 (J = the column). For example, $B\$(1,1)$ has the value "178654775", $B\$(1,2)$ has the value "38", $B\$(1,3)$ has the value "M", and so forth.

The Apple II Computer even allows you to have arrays which have three, four, or even more subscripts. For example, consider the dry cleaning chain array introduced above. Suppose that we had one such array for each of ten consecutive years. This collection of data could be stored in a three-dimensional array of the form $C(I,J,K)$, where I and J represent the row and column, just as before, and K represents the year. (K could assume the values 1, 2, 3, . . . , 10.)

An array may involve any number of dimensions up to 88. The subscripts corresponding to each dimension may assume values from 0 to 32767. For all practical applications, any size array is permissible.

You must inform the computer of the sizes of the arrays you plan to use in a program. This allows the computer to allocate memory space to house all the values. To specify the size of an array, use a dimension (**DIM**) statement. For example, to define the size of the subscripted variable $A(J)$, $J = 1, . . . , 1000$, we insert the statement

```
10 DIM A(1000)
```

in the program. This statement informs the computer to expect variables $A(0), A(1), . . . , A(1000)$ in the program and that it should set aside memory space for 1001 variables. Note that, in the absence of further instructions from you, Applesoft BASIC begins all subscripts at 0. If you wish to use $A(0)$, fine. If not, ignore it.

You need not use all the variables defined by a **DIM** statement. For example, in the case of the **DIM** statement above, you might actually use only the variables $A(1), . . . , A(900)$. Don't worry about it! Just make sure that you have defined enough variables. Otherwise you could be in trouble. For example, in the case of the subscripted variable above, your program might make use of the variable $A(1001)$. This will create an error condition. Suppose that this variable is used first in line 570. When you attempt to run the program, the computer will report:

```
? BAD SUBSCRIPT ERROR
```

Moreover, execution of the program will be halted. To fix the error, merely redo the **DIM** statement to accommodate the undefined subscript.

To define the size of a two-dimensional array, use a **DIM** statement of the form

```
10 DIM A(5,4)
```

This statement defines an array $A(I,J)$, where I can assume the values 0, 1, 2, 3, 4, 5 and J can assume the values 0, 1, 2, 3, 4. Arrays with three or more subscripts are defined similarly.

TEST YOUR UNDERSTANDING 1 (answers on page 70)

Here is an array.

12	645.80
148	489.75
589	12.89
487	14.50

- Define an appropriate subscripted variable to store this data.
- Define an appropriate **DIM** statement.

It is possible to dimension several arrays with one **DIM** statement. For example, the dimension statement

```
10 DIM A(1000), B$(5), A(5,4)
```

defines the array $A(0), \dots, A(1000)$, the string array B(0), \dots, B(5) and the two-dimensional array $A(I,J)$, $I = 0, \dots, 5$; $J = 0, \dots, 4$.

We know how to set aside memory space for the variables of an array. We must now take up the problem of assigning values to these variables. We could use individual **LET** statements, but with 1000 variables in an array, this could lead to an unmanageable number of statements. There are more convenient methods which make use of loops. The next two examples illustrate two of these methods.

Example 1. Define an array $A(J)$, $J = 1, 2, \dots, 1000$ and assign the following values to the variables of the array

$$A(1) = 2, A(2) = 4, A(3) = 6, A(4) = 8, \dots$$

Solution. We wish to assign each variable a value equal to twice its subscript. That is, we wish to assign $A(J)$ the value $2*J$. To do this we use a loop:

```
10 DIM A(1000)
20 FOR J=1 TO 1000
30 A(J)=2*J
40 NEXT J
50 END
```

Note that the program ignores the variable A(0). Like any variable which has not been assigned a value, it has the value 0.

TEST YOUR UNDERSTANDING 2 (answer on page 70)

Write a program which assigns the variables A(0), . . . , A(30) the values A(0) = 0, A(1) = 1, A(2) = 4, A(3) = 9,

When the computer is first turned on or is reset, all variables (including those in arrays) are cleared. All numeric variables are set equal to 0, and all string variables are set equal to the null string (the string with no characters in it). If you wish to return all variables to this state during the execution of a program, use the command **CLEAR**. For example, when the computer encounters the command

```
570 CLEAR
```

it will reset all the variables. The **CLEAR** command can be convenient if, for example, you wish to use the same array to store two different sets of information at two different stages of the program. After the first use of the array you could then prepare for the second use by executing a **CLEAR**.

Example 2. Define an array corresponding to the employee record table above. Input the values given and print the table on the screen.

Solution. Our program will print the headings of the columns and then ask for the table entries, one row at a time. We will store the entries in the array B\$(I,J), where I is one of 1, 2, 3, 4, or 5 and J is one of 1, 2, 3, or 4. We dimension the array as B\$(5,3).

```
5 DIM B$(5,3)
10 PRINT "SOC.SEC. #", "AGE", "SEX"
20 FOR I=1 TO 5
30 INPUT "SS #,AGE,SEX"; B$(I,1),B$(I,2),B$(I,3)
40 PRINT B$(I,1),B$(I,2),B$(I,3)
50 NEXT I
60 END
```

TEST YOUR UNDERSTANDING 3 (answer on page 70)

Suppose that your program uses a 9×2 array A\$(I,J), a 9×1 array B\$(I,J), and a 9×5 array C(I,J). Write an appropriate DIM statement(s).

If you plan to dimension an array, you should always insert the **DIM** statement before the variable first appears in your program. Otherwise, the first time Applesoft BASIC comes across the array, it will assume that the subscripts go from 0 to 10. If it subsequently comes across a **DIM** statement, it will think you are changing the size of the array in the midst of the program, something which is not allowed. If you try to change the size of an array in the middle of a program, you will get an error:

? REDIM'D ARRAY ERROR

EXERCISES (answers on page 226)

For each of the following tables, define an appropriate array and determine the appropriate **DIM** statement.

1. 5

2

1.7

4.9

11

2. 1.1 2.0 3.5

1.7 2.4 6.2

3. JOHN

MARY

SIDNEY

4. 1 2 3

5. RENT 575.00

UTILITIES 249.78

CLOTHES 174.98

CAR 348.70

6. Display the following array on the screen:

Receipts

Store #1	Store #2
57,385.48	89,485.45
39,485.98	76,485.49
45,467.21	71,494.25

7. Write a program that displays the array of Exercise 6 along with totals of the receipts from each store.

8. Expand the program in Exercise 7 so that it calculates and displays the totals of ten-day periods. (Your screen will not be wide enough to display the ten-day totals, so display them in a separate array.)
9. Devise a program which keeps track of the inventory of an appliance store chain. Store the current inventory in an array of the form

	Store #1	Store #2	Store #3	Store #4
Refrig.				
Stove				
Air Cond.				
Vacuum				
Disposal				

Your program should: 1) input the inventory corresponding to the beginning of the day, 2) continually ask for the next transaction—the store number and the number of appliances of each item sold, and 3) in response to each transaction, update the inventory array and redisplay it on the screen.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, and 3

- 1: a. $A(I,J)$, $I=1,2,3,4$; $J=1,2$
b. DIM A(4,2)
- 2: 10 DIM A(30)
20 FOR J=0 TO 30
30 A(J)=J^2
40 NEXT J
50 END
- 3: DIM A\$(9,2),B\$(9,1),C(9,5)

3.2 INPUTTING DATA

In the preceding section, we introduced arrays and discussed several methods for assigning values to the variables of an array. The most flexible method was via the **INPUT** statement. However, this can be a tedious method for large arrays. Fortunately, Applesoft BASIC provides us an alternate method for inputting data.

A given program may need many different numbers and strings. You may store the data needed in one or more **DATA** statements. A typical data statement has the form

```
10 DATA 3.457, 2.588, 11234, "WINGSPAN"
```

Note that this data statement consists of four data items, three numeric and one string. The data items are separated by commas. You may include as many data items in a single **DATA** statement as the line allows. Moreover, you may include any number of **DATA** statements in a program and they may be placed anywhere in the program, although a common placement is at the end of the program (just before the **END** statement). Note that we enclosed the string constant "WINGSPAN" in quotation marks. Actually this is not necessary. A string constant in a **DATA** statement does not need quotes as long as the string does not start with a blank.

The **DATA** statements may be used to assign values to variables and, in particular, to variables in arrays. Here's how to do this. In conjunction with the **DATA** statements, you use one or more **READ** statements. For example, suppose that the above **DATA** statement appeared in a program. Further, suppose that you wish to assign the values

$A = 3.457, B = 2.588, C = 11234, Z\$ = \text{"WINGSPAN"}$

This can be accomplished via the **READ** statement

100 READ A,B,C,Z\$

Here is how the **READ** statement works. On encountering a **READ** statement, the computer will look for a **DATA** statement. It will then assign values to the variables in the **READ** statement by taking the values, in order, from the **DATA** statement. If there is insufficient data in the first **DATA** statement, the computer will continue to assign values using the data in the next **DATA** statement. If necessary, the computer will proceed to the third **DATA** statement, and so forth.

TEST YOUR UNDERSTANDING 1 (answer on page 77)

Assign the following values:

$A(1) = 5.1, A(2) = 4.7, A(3) = 5.8, A(4) = 3.2, A(5) = 7.9, A(6) = 6.9.$

The computer maintains an internal pointer which points to the next **DATA** item to be used. If the computer encounters a second **READ** statement, it will start reading where it left off. For example, suppose that instead of the above **READ** statement, we use the two read statements

100 READ A,B
200 READ C,Z\$

Upon encountering the first statement, the computer will look for the location of the pointer. Initially, it will point to the first item in the first

DATA statement. The computer will assign the values $A = 3.457$ and $B = 2.588$. Moreover, the position of the pointer will be advanced to the third item in the **DATA** statement. Upon encountering the next **READ** statement, the computer will assign values beginning with the one designated by the pointer, namely $C = 11234$ and $Z\$ = \text{"WINGSPAN"}$.

TEST YOUR UNDERSTANDING 2 (answer on page 77)

What values are assigned to A and B\$ by the following program?

```
10 DATA 10,30,"ENGINE", "TACH"
20 READ A,B
30 READ C$,B$
40 END
```

The following example illustrates the use of **DATA** statements in assigning values to an array.

Example 1. Suppose that the monthly electricity costs of a certain family are as follows:

Jan.	\$89.74	Feb.	\$95.84	March	\$79.42
Apr.	78.93	May	72.11	June	115.94
July	158.92	Aug.	164.38	Sep.	105.98
Oct.	90.44	Nov.	89.15	Dec.	93.97

Write a program calculating the average monthly cost of electricity.

Solution. Let us unceremoniously dump all of the numbers shown above into **DATA** statements at the end of the program. Arbitrarily, let's start the **DATA** statements at line 1000, with **END** at 2000. This allows us plenty of room. To calculate the average, we must add up the numbers and divide by 12. To do this, let us first create an array $A(J)$, $J = 1, 2, \dots, 12$ and set $A(J)$ equal to the cost of electricity in the J th month. We do this via a loop and the **READ** statement. We then use a loop to add all the $A(J)$. Finally, we divide by 12 and **PRINT** the answer. Here is the program.

```
10 DIM A(12)
15 REM LINES 20-40 ASSIGN VALUES TO A(J)
20 FOR J=1 TO 12
30 READ A(J)
40 NEXT J
50 FOR J=1 TO 12
60 C=C+A(J): REM C ACCUMULATES THE SUM OF THE A(J)
70 NEXT J
80 C=C/12 : REM DIVIDE SUM BY 12
```



```

90 PRINT "THE AVERAGE MONTHLY COST OF ELECTRICITY IS",C
1000 DATA 89.74, 95.84, 79.42, 78.93, 72.11, 115.94
1010 DATA 158.92, 164.38, 105.98, 90.44, 89.15, 93.97
2000 END

```

The following program could be helpful in preparing the payroll of a small business.

Example 2. A small business has five employees. Here are their names and hourly wages.

Name	Hourly Wage
Joe Polanski	7.75
Susan Greer	8.50
Allan Cole	8.50
Betsy Palm	6.00
Herman Axler	6.00

Write a program which accepts as input hours worked for the current week and calculates the current gross pay and the amount of Social Security tax to be withheld from their pay. (Assume that the Social Security tax amounts to 6.70 percent of gross pay.)

Solution. Let us keep the hourly wage rates and names in two arrays, called A(J) and B\$(J), respectively, where J = 1, 2, 3, 4, 5. Note that we can't use a single two-dimensional array for this data since the names are string data, and the hourly wage rates are numerical. (Recall that Applesoft BASIC does not allow us to mix the two kinds of data in an array.) The first part of the program will be to assign the values to the variables in the two arrays. Next, the program will, one by one, print out the names of the employees and ask for the number of hours worked during the current week. This data will be stored in the array C(J), J = 1, 2, 3, 4, 5. The program will then compute the gross wages as A(J)*C(J) (that is, <wage rate> times <number of hours worked>). This piece of data will be stored in the array D(J), J = 1, 2, 3, 4, 5. Next, the program will compute the amount of Social Security tax to be withheld as .0670*D(J). This piece of data will be stored in the array E(J), J = 1, 2, 3, 4, 5. Finally, all the computed data will be printed on the screen. Here is the program:

```

10 DIM A(5),B$(5),C(5),D(5),E(5)
20 FOR J=1 TO 5
30 READ B$(J),A(J)
40 NEXT J
50 FOR J=1 TO 5
60 PRINT "TYPE CURRENT HOURS OF ", B$(J)
70 INPUT C(J)
80 D(J)=A(J)*C(J)

```

```

90 E(J)=.0670*D(J)
100 NEXT J
110 PRINT "EMPLOYEE","GROSS WAGES","SS.TX"
120 FOR J=1 TO 5
130 PRINT B$(J),D(J),E(J)
140 NEXT J
200 DATA JOE POLANSKI, 7.75, SUSAN GREER, 8.50
210 DATA ALLAN COLE, 8.50, BETSY PALM, 6.00 '
220 DATA HERMAN AXLER, 6.00
1000 END

```

In certain applications, you may wish to read the same **DATA** statements more than once. To do this you must reset the pointer via the **RESTORE** statement. For example, consider the following program.

```

10 DATA 2.3, 5.7, 4.5, 7.3
20 READ A,B
30 RESTORE
40 READ C,D
50 END

```

Line 20 sets A equal to 2.3 and B equal to 5.7. The **RESTORE** statement of line 30 moves the pointer back to the first item of data, 2.3. The **READ** statement of line 40 then sets C equal to 2.3 and D equal to 5.7. Note that without the **RESTORE** in line 30, the **READ** statement in line 40 would set C equal to 4.5 and D equal to 7.3.

There are two common errors in using **READ** and **DATA** statements. First, you may instruct the program to **READ** more data than is present in the **DATA** statements. For example, consider the following program.

```

10 DATA 1,2,3,4
20 FOR J=1 TO 5
30 READ A(J)
40 NEXT J
50 END

```

This program attempts to read five pieces of data, but the **DATA** statement only has four. In this case, you will receive an error message

```
? OUT OF DATA ERROR IN 30
```

A second common error is attempting to assign a string value to a numeric variable or vice versa. Such an attempt will lead to

```
? TYPE MISMATCH ERROR
```

The Apple II Computer allows you to save an array on a cassette using the **STORE** instruction. To save A(I,J,K), we would use an instruction of the form

```
100 STORE A
```

Prior to running a program containing a **STORE** instruction, it is necessary to position the cassette to a blank portion of tape and to turn on the cassette recorder to record. To recall the array from the cassette, use the instruction:

200 RECALL A

In order to use a **RECALL** instruction, it is necessary to position the tape at the beginning of the area on which A was stored and to turn on the cassette recorder to play. If you wish to do both **STORE** and **RECALL** in a single program, it will be necessary to program a **STOP** and a screen prompt into the program in order to tell you to reposition the tape and change the recorder from record to play.

EXERCISES (answers on page 228)

Each of the following programs assigns values to the variables of an array. Determine which values are assigned.

1. 10 DIM A(10)
20 FOR J=1 TO 10
30 READ A(J)
40 NEXT J
50 DATA 2,4,6,8,10,12,14,16,18,20
100 END
2. 10 DIM A(3),B(3)
20 FOR J=0 TO 3
30 READ A(J), B(J)
40 NEXT J
50 DATA 1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9
60 END
3. 10 DIM A(3),B\$(3)
20 FOR J=0 TO 3
30 READ A(J)
40 NEXT J
50 FOR J=0 TO 3
60 READ B\$(J)
70 NEXT J
80 DATA 1,2,3,4,A,B,C,D
90 END
4. 10 DIM A(3), B(3)
20 READ A(0),B(0)
30 READ A(1),B(1)

```

40 RESTORE
50 READ A(2),B(2)
60 READ A(3),B(3)
70 DATA 1,2,3,4,5,6,7,8
80 END

```

```

5. 10 DIM A(3,4)
   20 FOR I=1 TO 3
   30 FOR J=1 TO 4
   40 READ A(I,J)
   50 NEXT J
   60 NEXT I
   70 DATA 1,2,3,4,5,6,7,8,9,10,11,12
   80 END

```

```

6. 10 DIM A(3,4)
   20 FOR J=1 TO 4
   30 FOR I=1 TO 3
   40 READ A(I,J)
   50 NEXT I
   60 NEXT J
   70 DATA 1,2,3,4,5,6,7,8,9,10,11,12
   80 END

```

Each of the following programs contains an error. Find it.

```

7. 10 DIM A(5)
   20 FOR J=1 TO 5
   30 READ A(J)
   40 NEXT J
   50 DATA 1,2,3,4
   60 END

```

```

8. 10 DIM A(5)
   20 FOR J=1 TO 5
   30 READ A(J)
   40 NEXT J
   50 DATA 1,A,2,B
   60 END

```

9. Here is a table of Federal Income Tax Withholding of weekly wages for an individual claiming one exemption. Assume that each of the employees, in the business discussed in the text, claims a single exemption. Modify the program given so that it correctly computes Federal Withholding and the net amount of wages. (That is, the total after Federal Withholding and Social Security are deducted.)

Wages at Least	But Less Than	Tax Withheld
200	210	29.10
210	220	31.20
220	230	33.80
230	240	36.40

Wages at Least	But Less Than	Tax Withheld
240	250	39.00
250	260	41.60
260	270	44.20
270	280	46.80
280	290	49.40
290	300	52.10
300	310	55.10
310	320	58.10
320	330	61.10
330	340	64.10
340	350	67.10

10. Here is a set of 24 hourly temperature reports as compiled by the National Weather Service. Write a program to compute the average temperature for the last 24 hours. Let your program respond to a query concerning the temperature at a particular hour. (For example, what was the temperature at 2:00 PM?)

	AM	PM
12:00	10	38
1:00	10	39
2:00	9	40
3:00	9	40
4:00	8	42
5:00	11	38
6:00	15	33
7:00	18	27
8:00	20	22
9:00	25	18
10:00	31	15
11:00	35	12

ANSWERS TO TEST YOUR UNDERSTANDING 1 and 2

```
1: 10 DATA 5.1,4.7,5.8,3.2,7.9,6.9
   20 FOR J=1 TO 6
   30 READ A(J)
   40 NEXT J
   50 END
```

```
2: A=10, B$="TACH"
```

3.3 ADVANCED PRINTING

In this section, we will discuss the various ways in which you can format output on the screen and on the printer. Applesoft BASIC is quite flexible in the form in which you can cast output. You have control over placement of output on the line, degree of accuracy to which calculations are displayed, and so forth. Let us begin by reviewing what we have already learned about printing.

There are 40 print positions in each line. These are divided into print zones of 16 characters each. To start printing at the beginning of the next print zone, insert a comma between the items to be printed. To avoid any space between items, separate them in the PRINT statement by a semicolon. For example, the following program

```
10 A=5
20 PRINT "THE VALUE OF A IS EQUAL TO ";A
30 END
```

will result in the following screen display:

```
THE VALUE OF A IS EQUAL TO 5
```

Note that the first print item ends with a space. This is to guarantee a space between the final O and the 5.

TEST YOUR UNDERSTANDING 1 (answer on page 82)

Write a program which allows you to input two numbers. The program should then display them as an addition problem in the form $5 + 7 = 12$.

HORIZONTAL TABBING

You may begin a print item in any position on a line. To do this, use the **TAB** command. Note that a logical line may be up to 255 characters long. On the screen, an oversized line will wrap around to the next line. However, the line will print correctly on a printer having a wide enough print line. The positions of characters on logical lines are numbered from 0 to 255. The statement **TAB(7)** means to move to position 7. **TAB** is always used in conjunction with a **PRINT** statement. For example, the print statement

```
50 PRINT TAB(7) A
```

will print the value of the variable A, beginning in print position 7. It is

possible to use more than one **TAB** per **PRINT** statement. For example, the statement

```
100 PRINT TAB(5) A; TAB(15) B
```

will print the value of A beginning in print position 5, and the value of B beginning in print position 15. Note the semicolon between the two TAB instructions.

In typing a **PRINT** statement, you may run out of room on the line. To get around this problem, end the **PRINT** statement with a semicolon and continue the list of print items in another **PRINT** statement on the next line. For example, consider the pair of statements

```
100 PRINT "INVENTORY";  
110 PRINT , "OF LADIES SHOES"
```

The first line has a single print item. The semicolon indicates continued printing on the same line. The comma which begins the second **PRINT** statement moves printing to the beginning of the next print zone, where the string in line 110 is printed. Here is what the output looks like:

```
INVENTORY      OF LADIES SHOES
```

TEST YOUR UNDERSTANDING 2 (answer on page 82)

Write an instruction printing the value of A in column 25 and the value of B seven columns further to the right.

FORMATTING NUMBERS

Thus far, we have accepted numerical output from the Apple II in the form in which it was dispensed from the computer. However, this led to occasional embarrassments earlier in this book. For example, columns of numbers didn't line up, dollars and cents items were displayed to hundredths of a cent, and so forth. Let us now consider how to control the format of numerical output.

The Apple II carries out calculations to as many as 10 significant digits. Here are some typical examples of output:

```
1001, 1075.312, 123456789, 1.2415921E12
```

In many applications, it is necessary to round a number X to a given number of decimal places, say N places. This may be accomplished in

Applesoft BASIC using the statement

```
INT(X*10^N+.5)/10^N
```

(For now, don't worry where we came up with this.) For example, 5.758 rounded to one decimal place is equal to

```
INT(5.758*10+.5)/10
```

In preparing financial statements, it is usually necessary to print columns of numbers which are aligned, the one's under the one's column, the ten's under the ten's column, and so forth. This may be done by printing an appropriate number of blanks in front of the number. For example, suppose you wish to print a column of numbers, the longest of which consists of 9 digits and a decimal point. At the beginning of each number in the column, you would insert enough spaces so that the length of the resulting number would be 10 characters (including the decimal point). The "length" of a number X may be computed using the statement

```
LEN(STR$(X))
```

(The STR(X)$ is just X converted to a string constant. LEN gives the length of the string.) The number of spaces to be inserted would then be

```
10 - LEN(STR$(X))
```

Finally, this number of spaces in front of X may be generated by using the **PRINT** statement

```
PRINT SPC(10-LEN(STR$(X)));X
```

TEST YOUR UNDERSTANDING 3 (answer on page 83)

Write an instruction which prints the number 456.75387 rounded to two decimal places.

TEST YOUR UNDERSTANDING 4 (answer on page 83)

Write a program to calculate and display the numbers 2^j , $j = 1, 2, 3, \dots, 15$. The columns of the numbers should be properly aligned on the right.

Example 1. Here is a list of checks written by a family during the month of March.

\$15.32, \$387.25, \$57.98, \$3.47, \$15.88

5. \$ 12.82
 \$117.58
 \$ 5.87
 \$.99

Calculate
 Sum

6. Date 3/18/81
 Pay to the Order of Wildcatters, Inc.
 The sum of *****\$89,385.00

7. 5,787
 387
 127,486
 38,531

Calculate
 Sum

8. \$385.41
 -\$17.85

Calculate
 Difference

9. Write a program which rounds a number to the nearest integer. For example, if the input is 11.7, the output is 12. If the input is 158.2, the output is 158. Your program should accept the number to be rounded via an INPUT statement.
10. Write a program which allows your computer to function as a cash register. Let the program accept purchase amounts via INPUT statements. Let the user tell the program when the list of INPUTs is complete. The program should then print out the purchase amounts, with dollar signs and columns aligned, compute the total purchase, add 5 percent sales tax, compute the total amount due, ask for the amount paid, and compute the change due.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, 3, and 4

- 1: 10 INPUT A,B
 20 PRINT A;"+";B;"=";A+B
 30 END
- 2: 10 PRINT TAB(25) A;TAB(32) B

```

3: 10 PRINT INT(456.73:87*10^2+.5)/10^2
4: 10 FOR J=1 TO 15
    20 PRINT SPC(5-LEN(STR$(2^J))); 2^J
    30 NEXT J
    40 END

```

3.4 GAMBLING WITH YOUR COMPUTER

One of the most interesting features of your computer is its ability to generate events whose outcomes are "random." For example, you may instruct the computer to "throw a pair of dice" and produce a random pair of integers between 1 and 6. You may instruct the computer to "pick a card at random from a deck of 52 cards." You may also program the computer to choose a two-digit number "at random." And so forth. The source of all such random choices is the **random number generator**, which is a part of Applesoft BASIC. So let us begin by explaining what the random number generator is and how to access it. We will then give a number of interesting applications involving computer-assisted instruction and games of chance.

You may generate random numbers using the Applesoft BASIC function **RND(1)**. To explain how this function works, let us consider the following program:

```

10 FOR X=1 TO 500
20 PRINT RND(1)
30 NEXT X
40 END

```

This program consists of a loop which prints 500 numbers, each called **RND(1)**. Each of these numbers lies between 0.000000000 (inclusive) and 1.000000000 (exclusive). Each time **RND(1)** is called (as in line 20 above) the computer makes a "random" choice from among the numbers in the indicated range. This is the number that is printed.

To obtain a better idea of what we are talking about, you should generate some random numbers using a program like the one above. Unless you have a printer, 500 numbers will be too many for you to look at in one viewing. You should print two random numbers on one line (one per print zone) and limit yourself to 24 displayed lines at one time.

Here is a partial printout of such a program.

```
.2451213213      .3050034958
.9845468610      .9011598475
.8966099342      .6602124859
.5839383519      .4481638341
.1371193105      .2265442384
```

What makes these numbers “random” is that the procedure the computer uses to select them is “unbiased,” with all numbers having an equal likelihood of selection. Moreover, if you generate a large collection of random numbers, then numbers between 0 and .1 will comprise approximately 10 percent of those chosen, those between .5 and 1.0 will comprise 50 percent of those chosen, and so forth. In some sense, the random number generator provides a uniform sample of the numbers between 0 and 1.

TEST YOUR UNDERSTANDING 1 (answer on page 89)

Assume that `RND(1)` is used to generate 1000 numbers. Approximately how many of these numbers would you expect to lie between .6 and .9?

The function `RND(1)` generates random numbers lying between 0 and 1. In many applications, however, we will require randomly chosen **integers** lying in a certain range. For example, suppose that we wish to generate random integers chosen from among 1, 2, 3, 4, 5, 6. Let us multiply `RND(1)` by 6, to obtain `6*RND(1)`. This is a random number between 0.000000000 and 5.999999999. Next, let us add 1 to this number. Then `6*RND(1)+1` is a random number between 1.000000000 and 6.999999999. To obtain integers from among 1, 2, 3, 4, 5, 6, we must “chop off” the decimal portion of the number `6*RND(1)+1`. To do this, we use the `INT` function. If X is any number, then `INT(X)` is the largest integer less than or equal to X . For example,

`INT(5.23)=5`, `INT(7.99)=7`, `INT(100.001)=100`

Be careful in using `INT` with negative X . The definition we gave is correct, but unless you think things through, it is easy to make an error. For example,

`INT(-7.4)=-8`

since the largest integer less than or equal to -7.4 is equal to -8 . (Draw -7.4 and -8 on a number line to see the point!) Let us get back to our random numbers. To chop off the decimal portion of `6*RND(1)+1`, we

compute $\text{INT}(6*\text{RND}(1)+1)$. This last expression is a random number from among 1, 2, 3, 4, 5, 6. Similarly, the expression

$\text{INT}(100*\text{RND}(1)+1)$

may be used to generate random numbers from among the integers 1, 2, 3, . . . , 100.

TEST YOUR UNDERSTANDING 2 (answer on page 89)

Generate random integers from 0 to 1. (This is the computer analogue of flipping a coin: 0 = heads, 1 = tails). Run this program to generate 50 coin tosses. How many heads and how many tails occur?

Example 1. Write a program which turns the computer into a pair of dice. Your program should report the number rolled on each as well as the total.

Solution. We will hold the value of die #1 in the variable X and the value of die #2 in variable Y. The program will compute values for X and Y, print out the values and the total $X + Y$.

```

10 HOME
20 LET X=INT(6*RND(1)+1)
30 LET Y=INT(6*RND(1)+1)
40 PRINT "LADIES AND GENTLEMEN, BETS PLEASE!"
50 INPUT "ARE ALL BETS DOWN(Y/N)"; A$
60 IF A$="Y" THEN 100
70 GOTO 40
100 PRINT "THE ROLL IS",X,Y
110 PRINT "THE WINNING TOTAL IS " ; X+Y
120 INPUT "PLAY AGAIN(Y/N)"; B$
130 IF B$="Y" THEN 10
200 PRINT "THE CASINO IS CLOSING. SORRY!"
210 END

```

Note the use of computer-generated conversation on the screen. Note also, how the program uses lines 120–130 to allow the player to control how many times the game will be played.

TEST YOUR UNDERSTANDING 3 (answer on page 89)

Write a program which flips a "biased coin". Let it report "heads" one-third of the time and tails two-thirds of the time.

You may enhance the realism of a gambling program by letting the computer keep track of bets as in the following example.

Example 2. Write a program which turns the computer into a roulette wheel. Let the computer keep track of bets and winnings for up to five players. For simplicity, assume that the only bets are on single numbers. (In the next section, we will let you remove this restriction!)

Solution. A roulette has 38 positions: 1–36, 0, and 00. In our program, we will represent these as the numbers 1–38, with 37 corresponding to 0 and 38 corresponding to 00. A spin of the wheel will consist of choosing a random integer between 1 and 38. The program will start by asking the number of players. For a typical spin of the wheel, the program will ask for bets by each player. A bet will consist of a number (1–38) and an amount bet. The wheel will then spin. The program will determine the winners and losers. A payoff for a win is 32 times the amount bet. Each player has an account, stored in an array A(J), J = 1, 2, 3, 4. At the end of each spin, the accounts are adjusted and displayed. Just as in Example 1 above, the program asks if another play is desired. Here is the program.

```

10 INPUT "NUMBER OF PLAYERS";N
20 DIM A(5),B(5),C(5): REM AT MOST 5 PLAYERS ALLOWED
25 REM LINES 30-60 ALLOW PLAYERS TO PURCHASE CHIPS
30 FOR J=1 TO N : REM FOR EACH OF THE PLAYERS
40 PRINT "PLAYER "; J
50 INPUT "HOW MANY CHIPS"; A(J)
60 NEXT J
100 PRINT "LADIES AND GENTLEMEN!"
105 PRINT "PLACE YOUR BETS PLEASE!"
110 FOR J=1 TO N : REM FOR EACH OF THE PLAYERS
120 PRINT "PLAYER "; J
130 INPUT "NUMBER, AMOUNT"; B(J),C(J):REM INPUT BET
140 NEXT J
200 X=INT(38*RND(1)+1): REM SPIN THE WHEEL
210 REM LINES 210-300 DISPLAY THE WINNING NUMBER
220 PRINT "THE WINNER IS NUMBER "; X
300 REM: LINES 310-590 DETERMINE WINNINGS AND LOSSES
310 FOR J=1 TO N : REM FOR EACH PLAYER
320 IF X = B(J) THEN 400
330 A(J) = A(J)—C(J): REM PLAYER J LOSES. DEDUCT BET
340 PRINT "PLAYER ";J ;" LOSES"
350 GO TO 420
400 A(J)=A(J)+32*C(J): REM PLAYER J WINS. ADD WINNINGS
410 PRINT "PLAYER ";J;" WINS "; 32*C(J); " DOLLARS"
420 NEXT J
430 PRINT "PLAYER BANKROLLS"
440 PRINT
450 PRINT "PLAYER", "CHIPS"
460 FOR J=1 TO N

```

```

470 PRINT J,A(J)
480 NEXT J
500 INPUT "DO YOU WISH TO PLAY ANOTHER ROLL(Y/N) ";R$
510 HOME
520 IF R$="Y" THEN 100
530 PRINT "THE CASINO IS CLOSED. SORRY!"
600 END

```

You should try a few spins of the wheel. The program is fun as well as instructive. Note that the program allows you to bet more chips than you have. We will leave it to the exercises to add in a test that there are enough chips to cover the bet. You could also build lines of credit into the game!

You may treat the output of the random number generator as you would any other number. In particular, you may perform arithmetic operations on the random numbers generated. For example, $5*\text{RND}(1)$ multiplies the output of the random number generator by 5, and $\text{RND}(1) + 2$ adds 2 to the output of the random number generator. Such arithmetic operations are useful in producing random numbers from intervals other than 0 to 1. For example, to generate random numbers between 2 and 3, we may use $\text{RND}(1) + 2$.

Example 3. Write a program which generates 10 random numbers lying in the interval from 5 to 8.

Solution. Let us build up the desired function in two steps. We start from the function $\text{RND}(1)$, which generates numbers from 0 to 1. First, we adjust for the length of the desired interval. From 5 to 8 is 3 units, so we multiply $\text{RND}(1)$ by 3. The function $3*\text{RND}(1)$ generates numbers from 0 to 3. Now we adjust for the starting point of the desired interval, namely 5. By adding 5 to $3*\text{RND}(1)$, we obtain numbers lying between $0+5$ and $3+5$, that is between 5 and 8. Thus, $3*\text{RND}(1) + 5$ generates random numbers between 5 and 8. Here is the program required.

```

10 FOR J=1 TO 10
20 PRINT 3*RND(1)+5
30 NEXT J
40 END

```

Example 4. Write a function to generate random integers from among 5, 6, 7, 8, . . . , 12.

Solution. There are 8 consecutive integers possible. Let us start with the function $8*\text{RND}(1)$, which generates random numbers between 0 and 8. Since we wish our random number to begin with 5, let us add 5 to get $8*\text{RND}(1) + 5$. This produces random numbers between 5.00000000 and 12.99999999. We now use the INT function to chop off the decimal

part. This yields the desired function:

```
INT(8*RND(1)+5)
```

EXERCISES (answers on page 230)

Write Applesoft BASIC functions which generate random numbers of the following sorts.

1. Numbers from 0 to 100.
2. Numbers from 100 to 101.
3. Integers from 1 to 50.
4. Integers from 4 to 80.
5. Even integers from 2 to 50.
6. Numbers from 50 to 100.
7. Integers divisible by 3 from 3 to 27.
8. Integers from among 4, 7, 10, 13, 16, 19, and 22.
9. Modify the dice program so that it keeps track of payoffs and bankrolls, much like the roulette program in Example 2 (page 86). Here are the payoffs on a bet of one dollar for the various bets:

outcome	payoff
2	35
3	17
4	11
5	8
6	6.20
7	5
8	6.20
9	8
10	11
11	17
12	35

10. Modify the roulette program of Example 2 to check that a player has enough chips to cover the bet.
11. Modify the roulette program of Example 2 to allow for a \$100 line of credit for each player.
12. Construct a program which tests one-digit arithmetic facts, with the problems randomly chosen by the computer.

13. Make up a list of ten names. Write a program which will pick four of the names at random. (This is a way of impartially assigning a nasty task!)

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, and 3

1: 30%

```
2: 10 FOR J=1 to 50
    20 PRINT INT(2*RND(1)+1)
    30 NEXT J
    40 END
```

```
3: 10 LET X=INT(3*RND(1) + 1)
    20 IF X=1 THEN PRINT "HEADS": GOTO 40
    30 PRINT "TAILS"
    40 END
```

3.5 SUBROUTINES

In writing programs it is often necessary to use the same sequence of instructions more than once. It may not be convenient (or even feasible) to retype the set of instructions each time it is needed. Fortunately, Applesoft BASIC offers a convenient alternative: the subroutine.

A **subroutine** is a program which is incorporated within another, larger program. The subroutine may be used any number of times by the larger program. Often, the lines corresponding to a subroutine are isolated toward the end of the larger program. This arrangement is illustrated in Figure 3-1 (page 90). The arrow to the subroutine indicates the point in the larger program at which the subroutine is used. The arrow pointing away from the subroutine indicates that, after completion of the subroutine, execution of the main program resumes at the point at which it was interrupted.

Subroutines are handled with the pair of instructions **GOSUB** and **RETURN**. The statement

```
100 GOSUB 1000
```

sends the computer to the subroutine which begins at line 1000. The computer starts at line 1000 and carries out statements in order. When a **RETURN** statement is reached, the computer goes back to the main program, starting at the first line after 100. The next example illustrates the use of subroutines.

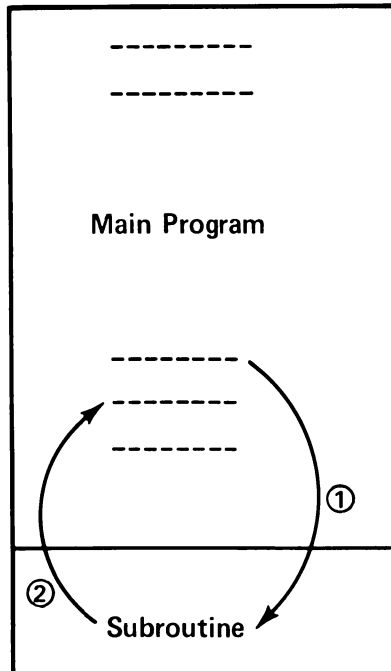


Figure 3-1. A subroutine.

Example 1. Modify the roulette program of Example 2 (page 86), so that it allows bets on EVEN and ODD. A one dollar bet on either of these pays one dollar in winnings.

Solution. Our program will now allow three different bets: on a number and on EVEN or ODD. Let us design subroutines, corresponding to each of these bets, which determine whether player J wins or loses. For each subroutine, let X be the number (1–38) which results from spinning the wheel. In the preceding program, a bet by player J was described by two numbers: B(J) equals the number bet and C(J) equals the amount bet. Now let us add a third number to describe a bet. Let D(J) equal 1 if J bets on a number, 2 if J bets on EVEN, and 3 if J bets on ODD. In case D(J) is 2 or 3, we will again let C(J) equal the amount bet, but B(J) will be 0. The subroutine for determining the winners of bets on numbers can be obtained by making small modifications to the corresponding portion of our previous program, as follows:

```

1000 IF B(J)=X THEN 1100
1005 GOTO 1010
1010 PRINT "PLAYER ";J;" LOSES"
1020 A(J)=A(J)-C(J)
1030 RETURN
1100 PRINT "PLAYER ";J;" WINS "; 32*C(J);" DOLLARS"

```

```

1110 A(J)=A(J)+32*C(J)
1120 RETURN

```

Here is the subroutine corresponding to the bet EVEN.

```

2000 K=1
2010 IF X=2*K THEN 2100
2015 K=K+1:IF K>19 THEN 2030
2020 GOTO 2010
2030 PRINT "PLAYER ";J;" LOSES"
2040 A(J)=A(J)-C(J)
2050 RETURN
2100 PRINT "PLAYER ";J;" WINS ";C(J);" DOLLARS"
2110 A(J)=A(J)+C(J)
2120 RETURN

```

Finally, here is the subroutine corresponding to the bet ODD.

```

3000 K=0
3005 IF X=2*K+1 THEN 3100
3010 K=K+1
3015 IF K > 19 THEN 3030
3020 GOTO 3005
3030 PRINT "PLAYER ";J;" LOSES"
3040 A(J)=A(J)-C(J)
3050 RETURN
3100 PRINT "PLAYER ";J;" WINS "; C(J);" DOLLARS"
3110 A(J)=A(J)+C(J)
3120 RETURN

```

Now we are ready to assemble the subroutines together with the main portion of the program, which is almost the same as before. The only essential alteration is that we must now determine, for each player, which bet was placed.

```

10 INPUT "NUMBER OF PLAYERS";N
20 DIM A(5),B(5),C(5): REM AT MOST 5 PLAYERS ALLOWED
25 REM LINES 30-60 ALLOW PLAYERS TO PURCHASE CHIPS
30 FOR J=1 TO N : REM FOR EACH OF THE PLAYERS
40 PRINT "PLAYER "; J
50 INPUT "HOW MANY CHIPS"; A(J)
60 NEXT J
100 PRINT "LADIES AND GENTLEMEN!"
105 PRINT "PLACE YOUR BETS PLEASE!"
110 FOR J=1 TO N : REM FOR EACH OF THE PLAYERS
120 PRINT "PLAYER "; J
121 PRINT "BET TYPE:1=NUMBER BET, 2=EVEN, 3=ODD"
122 INPUT "BET TYPE (1,2,OR 3)"; D(J)
123 IF D(J)=1 THEN 130
124 INPUT "AMOUNT"; C(J)
125 GOTO 140
130 INPUT "NUMBER,AMOUNT"; B(J),C(J):REM INPUT BET
140 NEXT J

```

```

200 X=INT(38*RND(1)+1): REM SPIN THE WHEEL
210 REM LINES 210-300 DISPLAY THE WINNING NUMBER
220 PRINT "THE WINNER IS NUMBER"; X
300 REM LINES 310-330 DETERMINE WINNINGS AND LOSSES
310 FOR J=1 TO N : REM FOR EACH PLAYER
320 IF D(J)=1 THEN GOSUB 1000
330 IF D(J)=2 THEN GOSUB 2000
335 IF D(J)=3 THEN GOSUB 3000
340 NEXT J
430 PRINT "PLAYER BANKROLLS"
440 PRINT
450 PRINT "PLAYER", "CHIPS"
460 FOR J=1 TO N
470 PRINT J,A(J)
480 NEXT J
500 INPUT "DO YOU WISH TO PLAY ANOTHER ROLL(Y/N)";R$
510 HOME
520 IF R$="Y" THEN 100
530 PRINT "THE CASINO IS CLOSED. SORRY!"
600 END
1000 IF B(J)=X THEN 1100
1005 GOTO 1010
1010 PRINT "PLAYER ";J;" LOSES"
1020 A(J)=A(J)-C(J)
1030 RETURN
1100 PRINT "PLAYER ";J; "WINS "; 32*C(J);" DOLLARS"
1110 A(J)=A(J)+32*C(J)
1120 RETURN
2000 K=1
2010 IF X=2*K THEN 2100
2015 K=K+1: IF K>19 THEN 2030
2020 GOTO 2010
2030 PRINT "PLAYER" ;J; "LOSES"
2040 A(J)=A(J)-C(J)
2050 RETURN
2100 PRINT "PLAYER" ;J; "WINS" ;C(J); "DOLLARS"
2110 A(J)=A(J)+C(J)
2120 RETURN
3000 K=1
3005 IF X=2*K+1 THEN 3100
3010 K=K+1
3015 IF K>19 THEN 3030
3020 GOTO 3005
3030 PRINT "PLAYER" ;J; "LOSES"
3040 A(J)=A(J)-C(J)
3050 RETURN
3100 PRINT "PLAYER"; J; "WINS" ;C(J); "DOLLARS"
3110 A(J)=A(J)+C(J)
3120 RETURN
3130 END

```

Note how the subroutines help organize our programming. Each subroutine is easy to write. Each is a small task and you will have less to

think about when considering the entire program. It is advisable to break a long program into a number of subroutines. Not only is it easier to write in terms of subroutines, but it is much easier to check the program and to locate errors since subroutines may be individually tested.

TEST YOUR UNDERSTANDING 1 (answer on page 95)

Consider the following program.

```

10 GOSUB 500
20 A=5
500 B=8
510 GOSUB 600
600 C=9
700 RETURN
800 END

```

What is the line executed after line 700?

Example 2. Here are production figures for six assembly lines of a factory for January 1980 and January 1981. Calculate the percentage increase (or decrease) for each assembly line and determine the assembly line with the largest percentage increase.

Assembly Line	January 1980	January 1981
1	235,485	239,671
2	298,478	301,485
3	328,946	322,356
4	315,495	318,458
5	198,487	207,109
6	204,586	221,853

Solution. Plan the program in terms of subroutines. Store the January 1980 data in the array A(J) and the January 1981 data in the array B(J), J = 1, 2, 3, 4, 5, 6. Our first step will be to read the data into the arrays from DATA statements. The second step will be to calculate the percentage increase (decrease) for each assembly line. This is done by using a subroutine which we will start at line 1000. Store the percentage increases in the array C(J), J = 1, 2, 3, 4, 5, 6. Finally, determine which of the numbers C(J) is the largest. This calculation can be carried out in a subroutine beginning in line 2000. Let K be the number of the assembly line with the largest percentage increase. We can then write our program as follows:

```

10 DIM A(6),B(6),C(6)
20 DATA 235485, 239671, 298478, 301485

```

```

30 DATA 328946, 322356, 315495, 318458
40 DATA 198487, 207109, 204586, 221853
50 FOR J=1 TO 6: REM READ ARRAYS
60 READ A(J), B(J)
70 NEXT J
80 FOR J=1 TO 6: REM COMPUTE PERCENT INCREASES
90 GOSUB 1000
100 NEXT J
200 PRINT "ASSEMBLY LINE","PERCENT INCREASE"
210 FOR J=1 TO 6: REM DISPLAY PERCENT INCREASES
220 PRINT J, C(J)
230 NEXT J
300 GOSUB 2000: REM DETERMINE LARGEST PERCENT INCREASE
310 PRINT "ASSEMBLY LINE",K,"IS THE WINNER"
400 END

```

This program is not complete. It is still necessary to complete the subroutines in lines 1000 and 2000. The point to make here is that the use of subroutines allowed us to organize the program and to plan it so that programming may be done in small steps. It is now possible to write the subroutines without worrying about the program in its entirety. In order for you to obtain some practice, we will leave construction of the two subroutines for the exercises on page 95. (If you are impatient, you may look at the answers on page 232).

Here is a useful variation of the **GOSUB** statement. Suppose that you wish to use:

- (1) the subroutine beginning at line 100 if the value of J is 1,
- (2) the subroutine beginning at line 500 if the value of J is 2, and
- (3) the subroutine beginning at line 1000 if the value of J is 3.

This complex set of decisions can be requested by a simple statement of the form

```
10 ON J GOSUB 100,500,1000
```

If the value of J is 1, the program goes to line 100; if the value of J is 2, the program goes to line 500; if the value of J is 3, the program goes to line 1000. If the value of J is not an integer, then any fractional part will be ignored. For example, if the value of J is 5.5, then the value 5 will be used. After dropping the fractional part, the value must be between 0 and 255 or an error will occur. If the value of J does not correspond to a given subroutine (that is, if J is 0 or more than 3 in the above example), then the instruction will be ignored. Instead of J, we could use any valid expression, such as $J^2 + 3$ or $3 * J - 2$. The expression will be evaluated and all of the above rules apply.

EXERCISES (answers on page 232)

1. Write a subroutine computing the value of $5 * J^2 - 3 * J$. Incorporate it in a program evaluating the given expression at $J = .1, .2, .3, .4, .5$.
2. Write the subroutine of Example 2 on page 93 which computes the percentage rates of change for each assembly line. Use the formula

$$\langle \% \text{ rate of change} \rangle = 100 \times (\langle \text{Jan.1981 Prod.} \rangle - \langle \text{Jan.1980 Prod.} \rangle) / \langle \text{Jan.1980 Prod.} \rangle$$
3. Write a subroutine for Example 2 determining the largest percentage increase. (Hint: Let the variable M hold the largest number among C(1), . . . , C(6). At the start, set M to C(1). Repeatedly compare M to C(2), C(3), . . . , C(6). After each comparison, replace M by the larger of M and what it is compared to. At the end of 5 comparisons, M will contain the largest among C(1), C(2), . . . , C(6). You may then determine which assembly line that value of M belongs to. That is, is the final value of M equal to C(1), C(2), C(3), C(4), C(5), or C(6)? For example, if M equals C(5), then assembly line 5 has the largest percentage increase.
4. Run the program of Example 2 with the subroutines in place.
5. Modify the roulette program of Example 1 (page 90) to allow the following bets: first 12 (1–12), second 12 (13–24), and third 12 (25–36). A winning one dollar bet of this type pays two dollars. In adding these bets, you may find the ON . . . GOSUB instruction helpful.

ANSWER TO TEST YOUR UNDERSTANDING 1

1: 600



4

Easing Programming Frustrations

As you have probably discovered by now, programming can be a tricky and frustrating business. You must first figure out the instructions to give the computer. Next, you must type the instructions into RAM. Finally, you must run the program and, usually on the first run, figure out why your program won't work. This process can be tedious and frustrating, especially in dealing with long or complex programs. We should emphasize that programming frustrations often result from the limitations and inflexibility of the computer to understand exactly what you are saying. In talking with another person, you usually sift out irrelevant information, correct minor errors, and still maintain the flow of communication. With a computer, however, you must first clear up all of the imprecisions before the conversation can even begin.

Fortunately, your computer has many features designed to ease the programming burdens, and to help you track down errors and correct them. We will describe these features in this chapter. We will also present some more tips which should help you develop programs quicker and with fewer errors.

4.1 EDITING PROGRAM LINES

Suppose that you discover a program line with an error in it. How can you correct it? Up to now, the only way was to retype the line. There is a much better way. The Apple II Computer allows you to **edit** lines. That is, you may add, delete, or change text in existing program lines.

On the Apple II Computer, the editing process consists of four steps:

1. Display the line on the screen.
2. "Copy" the line by using the → key to move the cursor over each character of the line.
3. Add, change, or delete characters as they arise in the "copy" process.
4. Send the line to the computer via the RETURN key.

The best way to understand the editing process is to work through several examples. If at all possible, follow these examples by typing them out on your keyboard. Suppose you have typed the following program lines:

```
10 PRINT X,Y,Z
20 IF A=5 THN 50: GOTO 30
]
```

The third line indicates the cursor position. We immediately see that there are two spelling errors: PRINT and THN. (If the computer had any common sense, it would have known what you meant.) In addition, suppose that we wish to change X, Y, and Z in the first line to read: A, X, Y, Z. Finally, suppose we wish to delete the :GOTO 30 on the second line. Let's use the editing process to correct them.

The fundamental idea of editing on the Apple II is to recopy the line while making the needed corrections. The recopy and correction procedures require that you move the cursor by using certain key combinations. Let's begin by describing how to move the cursor.

The following keys move the cursor without affecting anything already on the screen.

I—move cursor one space up
 J—move cursor one space to the left
 K—move cursor one space right
 M—move cursor one space down

Of course, these keys also generate the letters I, J, K, and M. To use them for moving the cursor, hit the key marked **ESC** (ESCAPE). You may return these keys to their normal functions by typing any key other than I, J, K, M, REPT, or ESC. In moving the cursor, you may find it helpful to use the **REPT** key in conjunction with the above keys. If you hold the **REPT** key down while pressing I, for instance, the cursor will execute a continuous move up. You may stop the cursor by lifting your fingers from the keys.

Let's get back to our lines to be corrected. The lines are already displayed on the screen (Step 1). Next, we move the cursor to the first digit of the line number of the first line to be corrected. We copy any characters which are correct by moving the cursor across them using the → key. (Note that the cursor motion key K will not do for this purpose!) We move the cursor until it rests on the M in PRINT . To correct the M, we type N. The display now looks like this:

```
10 PRINT X,Y,Z
20 IF A=5 THN 50: GOTO 30
```

The first error has now been corrected. We now continue to copy the line with the → key. We move across the T and the space and put the cursor on the X. We must insert the characters A and , before the X. Here is the display:

```
10 PRINT X,Y,Z
20 IF A=5 THN 50 :GOTO 30
```

To insert text at the cursor position, we type ESC I to move the cursor up one line. We then type the material to be inserted. Here is the display:

```
  A,B
10 PRINT X,Y,Z
20 IF A=5 THN 50: GOTO 30
```

We now type ESC M to bring the cursor down one line and type J three times to move the cursor back to the X. Now we resume the copy operation using the → key. After you pass over the last character of the line, hit **RETURN**. This finishes the corrections on the first line.

As the above example shows, it is a good idea to have a blank line above a line in which you wish to make corrections. Line 20 is on the screen, but has line 10 above it. At this point, you should type **LIST 20** followed by **RETURN**. This will display line 20 with a blank line above it

```
LIST 20
20 IF A=5 THN 50 :GOTO 30
]
```

We move the cursor to the first character of line 20. We begin the copy operation as we did in correcting line 10. We correct the misspelling of THEN by moving the cursor to the N (14 spaces to the right), and proceeding just as we did in correcting the **PRINT** error. The final correction is to delete :GOTO 30. This is done by positioning the cursor on the 0 and hitting the backspace key ←. As we backspace over a letter, it is erased. Finally, we hit the **RETURN** key to enter the corrected line. Here is an easier way of deleting the :GOTO 30. Position the

cursor at the first character to be deleted (the :) and hit the RETURN key.

The above example illustrates the various editing features of the Apple II Computer. We may use the editing keys in the same way to alter any line on the screen. If you wish to alter a program line which is not currently on the screen, you may display the desired line using the **LIST** command. Editing would then take place as shown.

IMPORTANT NOTE: Editing changes occur only in the copy of the program in RAM. In order for changes to be reflected in copies of the program on cassette or diskette, it is necessary to save the edited copy of the program.

EXERCISES

What keystrokes accomplish the following editing functions?

1. Move the cursor four spaces to the right.
2. Delete the fourth letter to the right of the cursor.
3. Insert the characters 538 at the current cursor position.
4. Delete the portion of the line to the right of the cursor position.
5. Move the cursor up eight spaces.
6. Move the cursor to the right three spaces.
7. List the current version of the line.
8. Change 0 to a 1 at the current cursor position.
9. Delete the letter "a" eight spaces to the left of the current cursor position.
10. Cancel all changes in the current line.

Use the line editor to make the indicated changes in the following program line. The exercises are to be done in order.

```
300 FOR M=11 TO 99, SETP .5 : X=M^2-5
```

11. Delete the ,
12. Correct the misspelling of the word STEP.
13. Change M^2-5 to M^3-2 .
14. Change .5 to -1.5.
15. Add the following characters to the end of the line. : Y = M + 1.

4.2 FLOW CHARTING

In the last two chapters, our programs, were fairly simple. By the end of Chapter Three we saw them becoming more involved. However, there are many programs which are much more lengthy and complex. You might be wondering how it is possible to plan and execute such programs. The key idea is to reduce large programs to a sequence of smaller programs which can be written and tested separately.

The old saying “A picture is worth a thousand words” is true for computer programming. In designing a program, especially a long one, it is helpful to draw a picture depicting the instructions of the program and their interrelationships. Such a picture is called a *flowchart*.

A flowchart is a series of boxes connected by arrows. Within each box is a series of one or more computer instructions. The arrows indicate the logical flow of the instructions. For example, the following flowchart shows a program for calculating the sum $1 + 2 + 3 + \dots + 100$. The arrows indicate the sequence of operations. Note the notation that $J = 1, 2, \dots, 100$ near the third box. This notation indicates a loop on

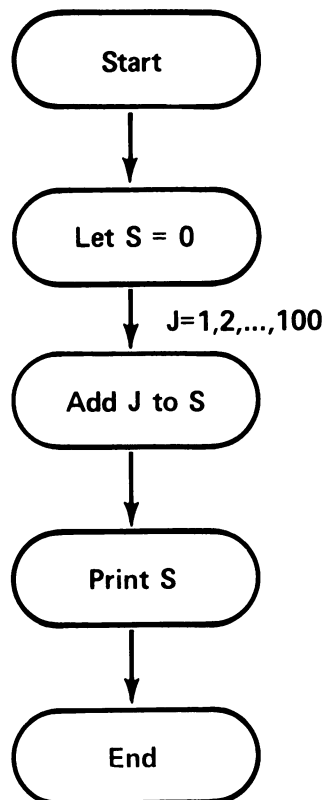


Figure 4-1.

the variable J. This means the operation in the box is to be repeated 100 times, for $J = 1, 2, \dots, 100$. Note how easy it is to proceed from the above flowchart to the corresponding BASIC program:

```

10 LET S=0           (box 2)
20 FOR J=1 TO 100
30 LET S=S+J       (box 3)
40 NEXT J
50 PRINT S         (box 4)
60 END             (box 5)

```

There are many flowcharting rules. Different shapes of boxes represent certain programming operations. We will adopt a very simple rule that all boxes are rectangular, except for decision boxes. Decision boxes are diamond-shaped. The following flowchart (Figure 4-2) shows a program which decides whether a credit limit has been exceeded. Note that the diamond-shaped block contains the decision "Is $D > L$?". Corresponding to the two possible answers to the question, there are two arrows leading from the decision box. Note also how we used the various boxes to help assign letters to the program variables. Once the flowchart is written, it is easy to transform it into the following program:

```

10 INPUT C           (boxes 1,2)
20 INPUT D,L
30 LET D=D+C        (box 3)
40 IF D>L THEN 100  (box 4)
50 GOTO 200         ("Yes" arrow)
100 PRINT "CREDIT DENIED" (box 6)
110 LET D=D-C      (box 7)
120 GOTO 300
200 PRINT "CREDIT OK" (box 5)
300 END           (box 8)

```

You will find flowcharting helpful in thinking out the necessary steps of a program. As you practice flowcharting, you will develop your own style and conventions. That's fine. We support all personalized touches, as long as they are comfortable and help you write programs.

EXERCISES (answers on page 234)

Draw a flowchart planning computer programs to do the following.

1. Calculate the sum $1^2 + 2^2 + \dots + 100^2$, print the result, and determine whether the result is larger than, smaller than, or equal to 487^3 .
2. The payroll program of Section 3.2.
3. The roulette program of Section 3.5 (see page 90).
4. The payroll program in Example 2 of Section 3.2 (see page 73).

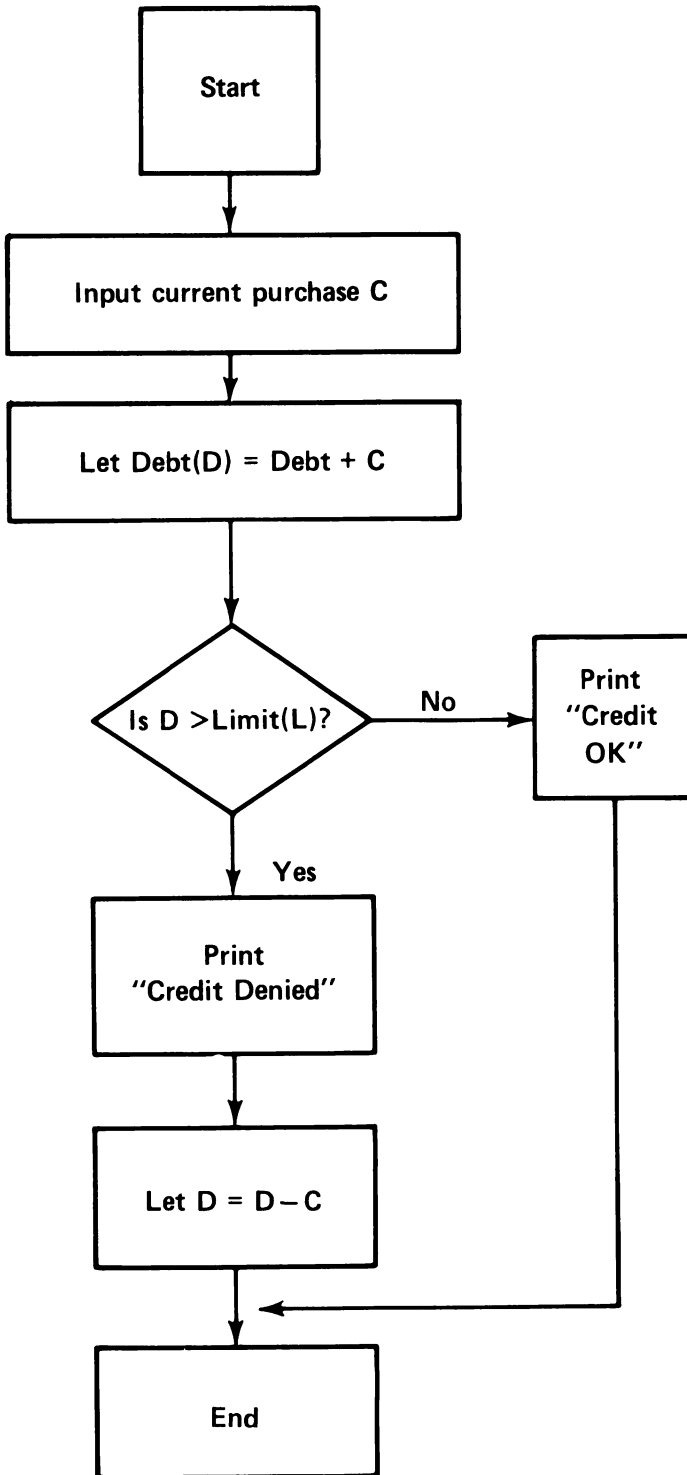


Figure 4-2.

4.3 ERRORS AND DEBUGGING

An error is sometimes called a “bug” in computer jargon. The process of finding these errors or “bugs” in a program is called **debugging**. This can often be a ticklish task. Manufacturers of commercial software must regularly repair bugs they discover in their own programs! Your Apple II Computer is equipped with a number of features to help detect bugs.

THE TRACE

Often your first try at running a program results in failure, while giving you no indication as to why the program is not running correctly. For example, your program might just run indefinitely, without giving you a clue as to what it is actually doing. How can you figure out what’s wrong? One method is to use the **trace** feature. Let us illustrate use of the trace in debugging the following program designed to calculate the sum $1 + 2 + . . . + 100$.

```
10 LET S=0
20 LET J=0
30 LET S=S+J
40 IF J=100 THEN 100
50 GOTO 200
100 LET J=J+1
110 GOTO 20
200 PRINT S
300 END
```

This program has two errors in it. (Can you spot them right off?) All you know initially is that the program is not functioning normally. The program runs, but prints out the answer 0, which we recognize as nonsense. How can we locate the errors? Let’s turn on the trace function by typing **TRACE**. The computer will respond by typing **J**. Now type **RUN**. The computer will run our program and print out the line numbers of all executed instructions. Here is what our display looks like:

```
TRACE
J

RUN
#10 #20 #30 #40 #200 0
#300
```

The numbers after # indicate the line numbers executed. That is, the computer executes, in order, lines 10, 20, 30, 40, 200, and 300. The 0 without a # is the program output resulting from the execution of line 200. The list of line numbers is not what we were expecting. Our program was designed (or so we thought) to execute line 100 after line 40.

No looping is taking place. How did we get to line 200 after line 40? This suggests that we examine line 40: Lo and behold! There is an error. The line numbers 100 and 200 appearing in lines 40 and 50 have been interchanged (an easy enough mistake to make). Let's correct this error by retyping the lines 40 and 50:

```
40 IF J=100 THEN 200
50 GOTO 100
```

In triumph, we run our program again. Here is the output:

```
#10 #20 #30 #40 #100 #110 #20 #30
#40 #100 #110 #20 #30 #40 #100 #110
#20 #30 #40 #100
BREAK IN 110
```

Actually, the above output goes whizzing by us as the computer races madly on executing the instructions. After about 30 seconds, we sense that something is indeed wrong since it is unlikely that our program could take this long. We stop execution by means of the key combination CTRL-C. The last line indicates that we interrupted the computer while it was executing line 110. Actually, your screen will be filled with output resembling the above. You will notice that the computer is in a loop. Each time it reaches line 110, the loop goes back to line 20. Why doesn't the loop ever end? In order for the loop to terminate, J must equal 100. Well, can J ever equal 100? Of course not! Every time the computer executes line 20, the value of J is reset to 0. Thus, J is never equal to 100 and line 40 always sends us back to line 20. We clearly don't want to reset J to 0 all the time. After increasing J by 1 (line 100), we wish to add the new J to S. We want to go to 30, not 20. We correct line 110 to read

```
110 GOTO 30
```

We **RUN** our program again. There will be a rush of line numbers on the screen followed by the output 5050, which appears to be correct. Our program is now running properly. We turn off the trace by typing **NOTRACE** (Trace off). Finally, we run our program once more for good measure. The above sequence of operations is summarized in the following display:

```
#40 #200 5050
#300
]

NOTRACE
]

RUN
5050
]
```

ERROR MESSAGES

In the example above, the program actually ran. A more likely occurrence is that there is a program line (or lines) which the computer is unable to understand due to an error or some other sort of problem. In this case, program execution ends too soon. The computer can often help in this instance since it is trained to recognize many of the most common errors. The computer will print an error message indicating the error type and the line number in which it occurred. You should immediately **LIST** the indicated line and attempt to find the cause of the error. Suppose that the error reads

```
SYNTAX ERROR IN 530
```

To analyze the error, you type

```
LIST 530
```

resulting in the display

```
530 LET Y=(X+2(X^2-2)
```

We note that there is an open parenthesis (without a corresponding close parenthesis). This is enough to trigger an error. We modify line 530 to read

```
530 LET Y=X+2(X^2-2)
```

We **RUN** the program again and find that there is still a syntax error in line 530! This is the frustrating part since not all errors are easy to spot. However, if you look closely at the expression on the right, you will note that we have omitted the ***** to indicate the product of 2 and (X^2-2) . This is a common mistake, especially for those familiar with the use of algebra. (In algebra, the product is usually indicated without any operation sign.) We correct line 530 again. (You may either retype the line or use the line editor.)

```
530 LET Y=X+2*(X^2-2)
```

Now we find that there is no longer a syntax error in line 530!

The appendix to this chapter contains a list of the most common error messages. For a complete list of error messages, the reader is referred to the **BASIC Programming Reference Manual**, Apple Publication #A2L0006.

EXERCISES (answers on page 236)

1. Use the error messages to debug the following program to calculate $(1^2 + 2^2 + \dots + 50^2)$ $(1^3 + 2^3 + \dots + 20^3)$.

```

10 LET S="0"
20 FOR J=1 TO 50
30 S=S+J(2
40 NEXT K
50 LET T=0
60 FOR J=1 TO 20
70 LET T=T+J^3
80 NXT T
90 NEXT T
100 LET A=ST
110 PRINT THE ANSWER IS, A
120 END

```

2. Use the trace function to debug the following program to determine the smallest integer N for which N^2 is larger than 175263.

```

10 LET N=0
20 IF N^2>175263 THEN 100
30 PRINT "THE FIRST N EQUALS"
100 N=N+1
110 GOTO 10
200 END

```

4.4 APPENDIX—SOME COMMON ERROR MESSAGES

? **SYNTAX ERROR.** There is an unclear instruction (misspelled?), mismatched parentheses, incorrect punctuation, illegal character, or illegal variable name in the program.

? **UNDEF'D STATEMENT ERROR.** The program uses a line number which does not correspond to an instruction. This can easily occur from deleting lines which are mentioned elsewhere. It can also occur when testing a portion of a program which refers to a line not yet written.

? **UNDEF'D FUNCTION ERROR.** The program uses a function which has not been defined in a DEF statement.

? **OVERFLOW ERROR.** A number too large for the computer.

? **DIVISION BY ZERO ERROR.** Attempting to divide by 0. This may be a hard error to spot. The computer will round to 0 any number smaller than the minimum allowed. Use of such a number in subsequent calculations could result in division by 0.

? **ILLEGAL QUANTITY ERROR.** (For the mathematically-minded.) Attempting to evaluate a function outside of its mathematically defined range. For example, the square root function is defined only for non-negative numbers, the logarithm function only for positive numbers, and the arctangent only for numbers between -1 and 1 . Any attempt to evaluate a function at a value outside these respective ranges will result in an ILLEGAL QUANTITY error.

? **BAD SUBSCRIPT ERROR.** Attempting to use an array with one or more subscripts outside the range allowed by the appropriate DIM statement.

? **STRING TOO LONG ERROR.** Attempting to specify a string containing more than 255 characters.

? **OUT OF MEMORY ERROR.** Your program will not fit into the computer's memory. This could result from large arrays, too many program steps, or a combination of the two.

? **FORMULA TOO COMPLEX ERROR.** Due to the internal processing of your formula, your formula resulted in a string expression that was too long or complex. This error can be corrected by breaking the string expression into a series of simpler expressions.

? **TYPE MISMATCH ERROR.** Attempting to assign a string constant as the value of a numeric variable, or a numeric constant to a string variable.

? **REDIM'D ARRAY ERROR.** Attempting to DIMension an array which has already been dimensioned. Note that once you refer to an array within a program, even if you don't specify the dimensions, the computer will regard it as being dimensioned at 10.

? **NEXT WITHOUT FOR ERROR.** A NEXT statement which does not correspond to a FOR statement.

? **RETURN WITHOUT GOSUB ERROR.** A RETURN statement encountered while not in a subroutine.

? **OUT OF DATA ERROR.** Attempting to read data which isn't there. This can occur in reading data from DATA statements, cassettes, or diskettes.

? **CAN'T CONTINUE ERROR.** Attempting to give a CONT command after the program has ENDED, or before the program has been RUN (such as after an EDIT session).

5

Your Computer as a File Cabinet

In the previous chapters, we learned the fundamentals for programming the Apple II Computer. We purposely avoided any direct discussion of diskette files. In this chapter, we will discuss the operation of these devices and their application to writing and reading programs and data files. In Section 5.1, we will discuss the difference between program files and data files; Sections 5.2, 5.3, and 5.4 provide an introduction to disk drive operations.

5.1 WHAT ARE DATA FILES?

Computer programs as used in business and industry usually refer to files of information which are stored within the computer. For example, a personnel department would keep a file of personal data on each employee, containing name, age, address, social security number, date employed, position, salary, and so forth. A warehouse would maintain an inventory for each product, with the following information: product name, supplier, current inventory, units sold in the last reporting period, date of last shipment, size of last shipment, and units sold in the last 12 months. Files like this are called *data files*. They could contain hundreds, thousands, or even hundreds of thousands of entries.

Data files are usually stored in a mass storage device. In the case of the Apple II Computer, this storage device is a diskette drive. Data files are to be distinguished from *program files*, which result from saving programs. Program files and data files exist side by side in mass storage.

The following example will give you a better idea of how a file is organized within mass storage. Suppose that a teacher stores grades in a data file. For each student in the class, there are four exam grades. A typical entry in the data file would contain the following data items:

student name, exam grade #1, exam grade #2,
exam grade #3, exam grade #4

In a data file, the data items are organized in sequence. So, the beginning of the above data file might look like this:

“John Smith”, 98, 87, 93, 76, “Mary Young”,
99, 78, 87, 91, “Sally Ronson”, 48, 63, 72,
80, ...

The data file consists of a sequence of either string constants (the names) or numeric constants (the grades), with the various data items arranged in a particular pattern (name followed by four grades). This particular arrangement is designed so the file may be read and understood. For instance, if we read the data items above, we know in advance that the data items are in groups of five with the first one a name and the next four the corresponding grades. In order for the computer to know where one data item ends and another begins, you must separate consecutive data items by characters called **delimiters**. Some examples of delimiters are: spaces, commas, and RETURN.

Your Apple II Computer can read data from a data file while a program is running, so the program can use the data. The file may be used within the program. For example, a personnel department might have a program which (1) enters changes in the personnel data file and (2) displays requested information about a given employee. To perform task (1), the program would read the personnel data file, alter the relevant items and rewrite the file into mass storage. To perform task (2), the program would read the data file, search for the requested information and display it on the screen or printer. In effect, your computer is serving as a convenient file cabinet for the storage of data. Moreover, the programming capability of the computer allows you to easily “shuffle through” the data for a specific piece of information.

In this chapter, we will discuss the mechanics of setting up, writing, and reading data files. We will also take this opportunity to introduce the operation and use of diskette files.

5.2 USING AN APPLE II COMPUTER DISK FILE

DISK FILE

So far, we have ignored the use of any diskette drives on your Apple II Computer. In this section, we will introduce disk files and their operation.

ON DISKS AND DISK FILES

The Apple II Computer has provisions for installing one or more disk drives. Each disk drive must be connected to a disk controller card which is inserted in one of the empty slots in the main system unit. A

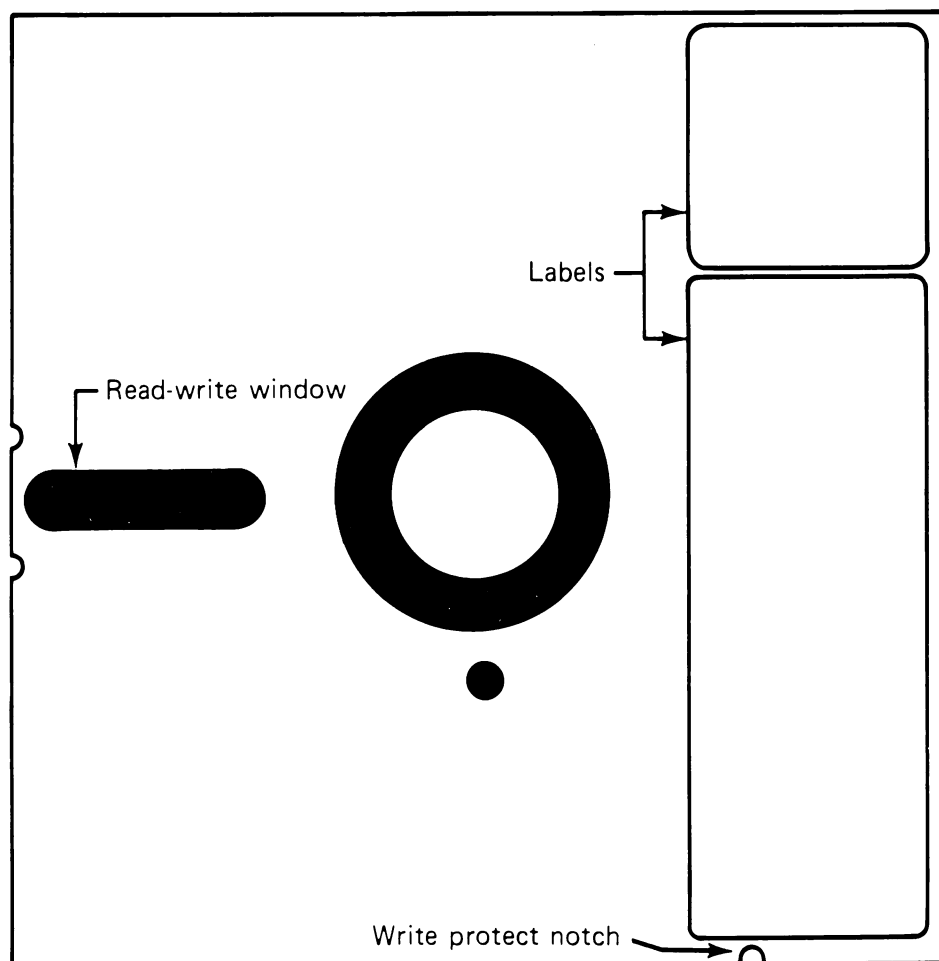


Figure 5-1. A diskette.

single controller card may control one or two disk files. A particular disk drive is described by two numbers: the slot number in which its controller card is inserted and the number 1 or 2 indicating the disk drive on the particular controller card. The slot number is indicated in commands by the letter S and the drive number is indicated by the letter D. Thus, for example,

S3, D2

refers to diskette drive 2 attached to the controller card in slot 3.

A word about the slots inside the system unit. The Apple II Computer allows you to build a customized system by inserting various circuit boards into the various slots of the system unit. It is customary for slot 0 to contain the video display controller board and for slot 6 to contain the disk controller board. **For the rest of this book, we will assume that the circuit boards are arranged in this fashion.**

To store information, the diskette drives use 5 1/4 inch floppy diskettes. Each diskette can accommodate approximately 143,000 characters (about 50 double-spaced typed pages). Figure 5-1 illustrates the essential parts of a diskette. The jacket is designed to protect the diskette. The interior of the jacket contains a lubricant which helps the diskette rotate freely within the jacket. The diskette is sealed inside. You should never attempt to open the protective jacket.

The diskette drive reads and writes on the disk through the **read-write** window. Never, under any circumstances, touch the surface of the diskette. Diskettes are very fragile. A small piece of dust or even oil from a fingerprint will damage the diskette and render all the information on it totally useless.

The **write protect notch** allows you to prevent changes to information on the diskette. When this notch is covered, with one of the metallic labels provided with the diskettes, the computer may read the diskette, but it will not write or change any information on the diskette. To write on a diskette, you must uncover the write protect notch.

To insert a diskette into a disk drive, open the door of the drive. Turn the diskette so that the label side is facing up and the read-write window is facing towards the rear. Push the diskette into the drive. Close the drive door.

Diskettes are extremely fragile. Here are some tips in using them.

1. Always keep a diskette in its paper envelope when not in use.
2. Store diskettes in a vertical position just like you would a phonograph record.

3. Never touch the surface of a diskette or try to wipe the surface of a diskette with a rag, handkerchief, or other piece of cloth.
4. Keep diskettes away from extreme heat, such as that produced from radiators, direct sun, or other sources of heat.
5. Never bend a diskette.
6. When writing on a diskette label, use only a felt-tipped pen. Never use any sort of instrument with a sharp point.
7. Keep diskettes away from magnetic fields, such as those generated by electrical motors, radios, televisions, tape recorders, and other devices. A strong magnetic field may erase data on a diskette.
8. Never remove a diskette while the drive is running. (You can tell if a drive is running by the sound of the motor and the "in use" light on the front of the drive.) Doing so may cause permanent damage to the diskette.

The above list of precautions may seem overwhelming to someone starting out. Once you set up a suitable set of procedures for handling and storing diskettes, you will find that they are a reliable, long-lasting storage medium.

STARTING THE COMPUTER

The version of Applesoft BASIC contained in ROM is not powerful enough to control the flow of information to and from the diskette drives. For this purpose, we need a program called an *operating system*. Such a program acts as a manager for all the activities which go on in the computer, coordinates the flow of information between the keyboard, video display, RAM, ROM, diskette files, and any other peripheral devices which you may have added to your computer system. Apple provides you with an operating system called Apple DOS* (Apple Disk Operating System—pronounced Apple-doss). This program is contained on a *System Master diskette* which is included when you purchase your diskette files.

In order to make use of your diskette drives, it is necessary to read DOS into the computer. (After all, DOS is going to manage the entire show!) To do so, follow this procedure:

1. Insert the System Master diskette into drive 1.
2. Close the drive door.

*Also called DOS for short.

3. Turn on the computer and obtain the displayed Applesoft BASIC prompt].
4. Type
PR#6
followed by **RETURN**. (The 6 indicates the slot containing the disk drive controller.)
5. Diskette drive 1 will turn on and you will hear the whirring action of the disk drive. When the disk drive is in use, the red light on the door will be lit. Never open the door while the red light is on. This may permanently damage the diskette and you may lose some or all of the data on the diskette.
6. DOS will display something like:

DOS VERSION X.X	DATE
APPLE II	SYSTEM MASTER
7. The computer will then display the Applesoft BASIC prompt. You are now ready to type in a program, exactly as we learned in Chapters 1 and 2.

TEST YOUR UNDERSTANDING 1

- a. Start your diskette operating system.
- b. Type in and run the following program.
10 PRINT 1+3+5+7
20 END

USING YOUR DISKETTE SYSTEM FOR THE FIRST TIME

Good programming practice dictates that you keep duplicate copies of all your diskettes. This will prevent the loss of your programs and data due to such accidents as a power blackout, coffee spilled on a diskette, and so forth. You should even make a copy of the system diskette. It is even a good idea to make a copy of the DOS diskette the first time you use it. Later, you should use only the copy. The original DOS diskette should be stored in a safe place so that yet another copy can be made if the first copy is damaged. Here is the procedure for making a backup copy of a diskette. (The instructions are for single disk systems. For systems with two disk drives, see the comment below.)

1. Follow the starting procedure outlined above.

2. Type

RUN COPYA

3. The computer will display

**APPLE DISKETTE DUPLICATION PROGRAM
ORIGINAL SLOT: DEFAULT=6
DRIVE: DEFAULT = 1**

Press

RETURN

4. The computer will display

DUPLICATE SLOT: DEFAULT=6

Type

1

(1 indicates the drive number which will be receiving the copy. If you have two drives and wish to place the copy in drive 2, type 2.)

5. The computer will display:

-PRESS 'RETURN' KEY TO BEGIN COPY

If you wish to copy the System Master diskette, just press the **RETURN** key. To copy some other diskette, place the diskette to be copied in drive 1, place the duplicate in drive 2, and then press **RETURN**.

6. The computer will now copy from the original disk into RAM. If you have only one diskette drive, the computer will prompt you to insert the duplicate diskette in drive 1. After you insert the duplicate diskette, press the **RETURN** key. The data in RAM is now copied onto the duplicate diskette. It will be necessary for you to switch the original and duplicate diskettes several times. Just follow the instructions provided on the screen.

A copy of a diskette, file, or program is called a *backup*.

TEST YOUR UNDERSTANDING 2

Make a backup of the System Master diskette supplied with your disk operating system.

A WORD TO THE WISE

The backup procedure just described may be used to copy the contents of any diskette onto any other. Because diskettes are fragile, it is strongly urged that you maintain duplicate copies of all your diskettes. A good procedure is to update your copies at the end of each session with the computer. This may seem like a big bother, but it will prevent untold grief if, by some mishap, a diskette with critical programs or data is erased or damaged.

5.3 AN INTRODUCTION TO DOS

DOS uses the version of Applesoft BASIC stored in ROM. DOS also provides a number of additional commands which make Applesoft BASIC more flexible and easy to use. It is beyond the scope of this book to cover all the special features of DOS. For a complete description of these features, we refer you to **Apple II—The DOS Manual**. However, here are a few of the most useful features of DOS.

SAVING AND LOADING PROGRAMS

To save a program, use the **SAVE** command. For example, to save a program named BUDGET on the diskette in drive 1, we use the command

```
SAVE BUDGET, D1
```

Here is another example. To save a program named ROULETTE.011 on drive 2, we use the command

```
SAVE ROULETTE.011, D2
```

Suppose that, at some later time, the diskette containing this program is moved to drive 1. The program may then be loaded into RAM via the command

```
LOAD ROULETTE.011, D1
```

TEST YOUR UNDERSTANDING 1

- a. Save the program

```
10 PRINT 5+7
```

```
20 END
```

under the name BUTTER.

- b. Try to load the program without giving the correct name.
- c. Load the program from diskette.

THE CURRENT DISK DRIVE

One of the diskette drives is designated the **current disk drive**. If you give a DOS command without specifying which disk drive, DOS will automatically assume that you are referring to the current disk drive. The start-up sequence of the preceding section (page 113) will designate drive 1 as the current drive. As soon as you give a DOS command specifying a drive other than the current drive, the current drive is changed to the drive specified.

Suppose, for example, that the current disk drive is 1 and that we give the command:

SAVE INVENTORY

The program INVENTORY will then be saved on drive 1. If you then give a DOS command which involved drive 2, the current drive will change to drive 2.

DOS commands may be used in two ways. First, a DOS command may be typed any time the Applesoft prompt is displayed. In this case, the computer is not executing a program. A DOS command given in this situation is executed immediately. When the command is executed, the Applesoft BASIC prompt is redisplayed.

A DOS command may also be used within BASIC programs. In this case, you use an instruction of the form:

10 PRINT CHR\$(4); " DOS command"

The CHR\$(4) is a control character which tells the computer that what follows is a DOS command. The actual DOS command is typed in quotation marks as shown above.

To give you an idea of what DOS can do, let's survey some of the available DOS commands.

CATALOG

The DOS command **CATALOG** allows you to ask for the directory listing the names of all files on a given diskette. For example, to display

the directory of the diskette in drive 1, just type

CATALOG,D1

followed by **RETURN**. (You may omit the D1 if drive 1 is the current drive.)

ERASING FILES

You may erase files from a diskette. To erase the file ROULETTE, type

DELETE ROULETTE

RENAMING A FILE

You may rename a file by using the **RENAME** command. To change the name of ROULETTE to GAME, we use the command

RENAME ROULETTE, GAME

Note that the old name always comes **first**, followed by the new name.

EXERCISES (answers on page 236)

1. a. Write a program which computes $1^2 + 2^2 + \dots + 50^2$.
b. **SAVE** the program under the name SQUARES.
2. Recover the program of 1a. without retyping it.
3. Erase the program SQUARES of 1a.
4. Make a copy of the diskette you are using.

5.4 DATE FILES FOR DISK USERS

In this section we will discuss the procedures for reading and writing data files on diskettes.

In order to either read or write a data file on diskette, it is necessary to **OPEN** the file. When opening a file, you tell DOS whether you plan to read the file or write the file. For example, to write a file with the name INVOICE.034 on disk drive 2, we would use a sequence of instructions like this:

```
100 PRINT CHR$(4); "OPEN INVOICE.0034, D2"
110 PRINT CHR$(4); "WRITE INVOICE.0034, D2"
```

Note that these instructions do not actually write any data into the file. They merely prepare the file for output.

Suppose now that we wish to enter the following data into the file:

DJ SALES \$358.79 4/5/81

We would use the following instruction:

200 ? "DJ SALES": ? \$358.79": ? "4/5/81"

Once the **WRITE** command has been given, all subsequent **PRINT** statements will cause data to be written into the file indicated.

When you are finished writing a file, you must close it with a **CLOSE** instruction. To close the file we have just been considering, we use the instruction

300 PRINT CHR\$(4); "CLOSE INVOICE.034,D1"

Example 1. Create a data file consisting of names, addresses, and telephone numbers from your personal telephone directory. Assume that you will type the addresses into the computer and will tell the computer when the last address has been typed.

Solution. We use **INPUT** statements to enter the various data. Let A\$ denote the name of the current person, B\$ the street address, C\$ the city, D\$ the state, E\$ the zip code, and F\$ the telephone number. For each entry, there is an **INPUT** statement corresponding to each of these variables. The program then writes the data to the diskette. Here is the program:

```

10 INPUT "NUMBER OF ENTRIES IN DIRECTORY";N
20 DIM A$(100),B$(100),C$(100),D$(100),E$(100),F$(100)
30 FOR J=1 TO N
110 INPUT "NAME"; A$(J)
120 INPUT "STREET ADDRESS"; B$(J)
130 INPUT "CITY"; C$(J)
140 INPUT "STATE"; D$(J)
150 INPUT "ZIP CODE"; E$(J)
160 INPUT "TELEPHONE"; F$(J)
170 NEXT J
200 PRINT CHR$(4); "OPEN TELEPHONE"
210 PRINT CHR$(4); "WRITE TELEPHONE"
220 FOR J=1 TO N
230 ?A$(J): ?B$(J): ?C$(J): ?D$(J): ?E$(J): ?F$(J)
240 NEXT J
300 ?"END": ?"": ?"": ?" ":?"": ?""
310 PRINT CHR$(4); "CLOSE TELEPHONE"
320 END

```

The above program may be used to set up a directory consisting of up to 100 entries. You should set up such a computerized telephone directory of your own. It is very instructive. Moreover, when coupled with the search program given below, it will allow you to look up addresses and phone numbers using your computer.

TEST YOUR UNDERSTANDING 1

Use the above program to enter the following address into the file.

John Jones
1 South Main St. Apt. 308
Phila. Pa. 19107
527-1211

The **WRITE** instruction begins to write the file from the beginning. If you wish to add to an existing file, you should use the **APPEND** instruction:

```
10 PRINT CHR$(4); "APPEND TELEPHONE"  
20 PRINT CHR$(4); "WRITE TELEPHONE"
```

Such a sequence of instructions will cause any subsequent **PRINT** statements to print their data at the end of the file.

TEST YOUR UNDERSTANDING 2

Add to the telephone file started in TEST YOUR UNDERSTANDING 1 above, the following entry.

Mary Bell
2510 9th St.
Phila. Pa. 19138
937-4896

WARNING: If you wish to replace a file which already exists with one of the same name, you should erase the first file using an instruction of the form

```
20 PRINT CHR$(4); "DELETE TELEPHONE"
```

Then you may safely write the new version using the instructions shown above. If you fail to delete the old version you run the risk of mixing the old and new versions. In short, you may create a ghastly mess!

Let's now discuss the procedure for reading data files from a diskette. As is the case with writing files, it is necessary to first open the file. Consider the telephone file in Example 1 (page 119). To open it for input, we could use the instruction

```
300 PRINT CHR$(4); "OPEN TELEPHONE"
310 PRINT CHR$(4); "READ TELEPHONE"
```

Once the file is open, it may be read via the instruction

```
400 INPUT A$,B$,C$,D$,E$,F$
```

This instruction will read one of the telephone-address entries from the file. In order to read a file, it is necessary to know the precise format of the data in the file. For example, the form of the above **INPUT** statement was dictated by the fact that each telephone-address entry was entered into the file as six consecutive string constants, separated by commas. The input statement works like any other input statement: Faced with a list of items separated by commas, it assigns values to the indicated variables, in the order in which the data items are presented. Note that the commas in the data file are essential. In order for an **INPUT** statement to assign values to several variables at once, the values must be separated by commas!

Example 2. Write a program which searches for a particular entry of the telephone directory file created in Example 1.

Solution. We will **INPUT** the name corresponding to the desired entry. The program will then read the file entries until a match of names occurs. Here is the program:

```
10 INPUT "NAME TO SEARCH FOR"; Z$
15 PRINT CHR$(4); "OPEN TELEPHONE"
16 PRINT CHR$(4); "READ TELEPHONE"
20 INPUT A$,B$,C$,D$,E$,F$
30 IF A$=Z$ THEN 100
40 IF A$="END" THEN 200
50 GOTO 20
100 HOME
110 PRINT A$
120 PRINT B$
130 PRINT C$: PRINT D$: PRINT E$
140 PRINT F$: PRINT CHR$(4); "CLOSE TELEPHONE"
150 GOTO 1000
200 HOME
300 PRINT CHR$(4) "CLOSE TELEPHONE"
400 PRINT "THE NAME IS NOT ON FILE"
1000 END
```

TEST YOUR UNDERSTANDING 3

Use the above program to locate Mary Bell's number in the telephone file created in TEST YOUR UNDERSTANDING 1 and 2 (both on page 120)

Example 3. Write a program which adds entries to the file TELEPHONE. The additions should be typed via INPUT statements. The program may assume that the file is on the diskette in drive 1.

Solution. Here is the program.

```

110 PRINT "TYPE ENTRY:NAME,STREET ADDRESS,CITY, STATE,"
120 PRINT "ZIP CODE, TELEPHONE NO."
130 INPUT A$,B$,C$,D$,E$,F$
135 PRINT CHR$(4); "OPEN TELEPHONE"
140 PRINT CHR$(4); "APPEND TELEPHONE"
150 PRINT CHR$(4); "WRITE TELEPHONE"
200 PRINT A$, B$, C$, D$, E$, F$
210 PRINT "CLOSE TELEPHONE"
250 INPUT "ANOTHER ENTRY (Y/N)"; Z$
260 IF Z$="Y" THEN 300
270 GOTO 400
300 HOME
310 GOTO 110
400 PRINT CHR$(4); "APPEND TELEPHONE"
410 PRINT CHR$(4); "WRITE TELEPHONE"
430 PRINT "END"
440 PRINT CHR$(4); "CLOSE TELEPHONE"
500 END

```

In the above programs, we have indicated the end of a data file by writing "END" as the last data item. This allows us to read to the end of the file and no further, thereby avoiding an error. Another way of handling the end of the file problem is to read the data items until an error actually does occur. We prepare for the expected error by placing an **ONERR GOTO** statement before the point at which the error will occur. The statement should send the computer to a line containing a **RESUME** statement which in turn sends the computer back to the next line after the one the error occurred at.

In the next example, we present a program useful for parents who wish to teach organizational skills to their children. Most children love to play with the computer. Here is a program which acts as an assignment book and monitors progress on homework. This program was designed for Jonathan Goldstein, a 10-year-old computer enthusiast.

Example 4. Write a program which sets up a data file for homework assignments. The child should enter the assignments, by subject, when returning from school. As assignments are completed, the child may check them off. The program should tell the child whether his homework is complete.

Solution. Our program will first ask if the assignment has been recorded before. If so, it will be in a file. If not, the program will prompt the child to type in the assignments by subject, with prompts like

What is your math assignment?

The only rule is that assignments cannot have commas in their statements. The child types in the assignment followed by RETURN and the computer responds with the next subject. If there is no assignment, type RETURN. (You may customize the program by entering your own subjects.) After all assignments are entered, the computer asks the child if he wishes the assignments displayed. If the answer is yes, the computer then produces a list of all subjects with their corresponding assignments:

SUBJECT	ASSIGNMENT
MATH	
READING	CHAP 3
SPELLING	P80 SENTENCES

The computer now gives the child a chance to check off a completed assignment. The computer doesn't allow for forgetfulness. It asks for the subject, then displays the assignment and asks if, in fact, that assignment has been completed. If so, the next time the list of assignments is displayed, an X will appear beside completed assignments. Finally, the computer scans the list of assignments and decides whether all are complete. If so, it prints "HOMEWORK DONE"; if not, it prints "HOMEWORK NOT DONE". Here is the program.

```

20 DIM B$(20),C$(20),D$(20)
30 HOME
40 PRINT "HAVE YOU ENTERED THIS ASSIGNMENT BEFORE?"
50 INPUT A$
60 IF A$="Y" THEN 70
65 GOTO 140
70 PRINT "SUBJECT";TAB(20) "ASSIGNMENT"
80 PRINT CHR$(4); "OPEN SCHED"
85 PRINT CHR$(4); "READ SCHED"
90 FOR J=1 TO 6
100 INPUT B$(J),C$(J),D$(J)
110 PRINT B$(J);TAB(15) C$(J);TAB(25) D$(J)
120 NEXT J

```

```

130 PRINT CHR$(4); "CLOSE SCHED"
135 GOTO 370
140 HOME
150 DATA "MATH", "SPELLING", "LANGUAGE", "SOCIAL STUDIES"
160 DATA "READING", "SCIENCE", "CURRENT EVENTS"
170 FOR J=1 TO 6
180 READ B$(J)
190 PRINT "DO YOU HAVE ANY ";B$(J);" HOMEWORK TONIGHT?"
200 INPUT A$
210 IF A$="Y" THEN 220
215 GOTO 250
220 PRINT "WHAT IS THE ";B$(J);" ASSIGNMENT?"
230 INPUT C$(J)
250 NEXT J
260 PRINT "DO YOU WISH TO SEE YOUR ASSIGNMENTS?"
270 INPUT A$
280 IF A$="Y" THEN 300
290 GOTO 370
300 HOME
310 PRINT "SUBJECT";TAB(15) "ASSIGNMENT"
320 PRINT
330 FOR J=1 TO 6
340 PRINT B$(J); TAB(15) C$(J)
350 NEXT J
370 PRINT "DO YOU WANT TO CHECK OFF AN ASSIGNMENT?"
380 INPUT A$
400 IF A$="Y" THEN 410
405 GOTO 520
410 INPUT "SUBJECT";B$
420 FOR J=1 TO 6
440 PRINT "IS THIS ASSIGNMENT DONE?"
450 PRINT C$(J)
460 INPUT A$
470 IF A$="Y" THEN D$(J)="X"
475 IF B$=B$(J) THEN PRINT B$(J) : NEXT J
480 HOME
490 FOR J=1 TO 6
500 PRINT B$(J);TAB(15) C$(J);TAB(25) D$(J)
510 NEXT J
520 FOR J=1 TO 6
530 IF D$(J) <> "X" AND C$(J) <> "" THEN E$="W"
540 NEXT J
550 IF E$="W" THEN PRINT "HOMEWORK IS NOT DONE"
555 IF E$ >< "W" THEN PRINT "HOMEWORK DONE"
560 INPUT "DO YOU WISH TO CHECK OFF ANOTHER
ASSIGNMENT";A$
570 IF A$=Y THEN E$ = "" GOTO 380
575 GOTO 577
577 PRINT CHR$(4); "DELETE SCHED"
580 PRINT CHR$(4); "OPEN SCHED"
581 PRINT CHR$(4); "WRITE SCHED"

```

```

590 FOR J=1 TO 6
600 PRINT B$(J):PRINT C$(J):PRINT D$(J)
610 NEXT J
620 PRINT CHR$(4); "CLOSE SCHED"
700 END

```

The files we have been discussing are called *sequential files*. These files must be read in the exact order in which they are written. DOS also allows *you to have random access* files. These files let you read a given piece of data without reading all the data written ahead of it. A discussion of random access files is beyond the scope of this book. The interested reader may refer to the **DOS Manual**.

EXERCISES (answers on page 237)

1. Write a program creating a diskette data file containing the numbers 5.7, 11.4, 123, 485, and 49.
2. Write a program which reads the data file created in Exercise 1 and displays the data items on the screen.
3. Write a program which adds to the data file of Exercise 1 the data items 5,78, 4.79, and -1.27.
4. Write a program which reads the expanded file of Exercise 3 and displays all the data items on the screen.
5. Write a program which records the contents of checkbook stubs in a data file. The data items of the file should be as follows:
check #, date, payee, amount, explanation
Use this program to create a data file corresponding to your previous month's checks.
6. Write a program which reads the data file of Exercise 5 and totals the amounts of all the checks listed in the file.
7. Write a program which keeps track of inventory in a retail store. The inventory should be described by a data file whose entries contain the following information:
item, current price, units in stock

The program should allow for three different operations: Display the data file entry corresponding to a given item, record receipt of a shipment of a given item, and record the sale of a certain number of units of a given item.

8. Write a program which creates a recipe file to contain your favorite recipes.
9. (For Teachers) Write a program which maintains a student file containing your class roll, attendance, and grades.
10. Write a program maintaining a file of your credit card numbers and the party to notify in case of loss or theft.

6

An Introduction to Computer Graphics

In many applications, it is helpful to present data in pictorial form. By displaying numerical information in graphical form, it is often possible to develop insights and to draw conclusions which are not immediately evident from the original data. In this chapter, we will discuss procedures for using your Apple II Computer to create various kinds of pictorial displays on the screen. Such procedures belong to the field of **computer graphics**. This chapter gives you an introduction to this field.

6.1 LOW RESOLUTION GRAPHICS PRINCIPLES

There are three display modes: **text mode**, **low resolution graphics mode**, and **high resolution graphics mode**. The text mode is the one you have been using up to this point to display characters on the screen. The low resolution graphics mode lets you draw figures in as many as 16 colors. High resolution allows you to draw very detailed figures, but only in eight colors. In order to obtain color, you must hook your Apple II Computer up to a color monitor (your color television set will do.) If you use a black and white monitor, you may use the graphics modes exactly as described below, but you will generate only black and white pictures.

You may select between these modes by using the following Apple-soft BASIC commands.

TEXT = text mode
GR = low resolution graphics mode
HGR = high resolution graphics mode

When BASIC is started, the display is automatically in text mode. The above commands may be used to switch from one display mode to another, either within a program or via a keyboard command.

TEXT MODE

As we have already noted, the video display of the Apple II Computer is capable of displaying 24 rows of 40 characters each. This gives us up to 24 x 40 or 960 possible character positions. These various character positions divide the screen into small rectangles. One rectangle corresponds to each character position.

LOW RESOLUTION GRAPHICS MODE

In low resolution graphics mode, the screen is divided into the same number of columns as for text mode, but each text row is divided into two low resolution graphics rows. We have indicated in Figure 6-1 the subdivision of the screen corresponding to the text mode.

The rectangles which we have divided the screen into are arranged in rows and columns. We number the rows from 0 to 47, with row 0 being at the top of the screen and row 47 at the bottom. The columns are numbered from 0 to 39, with column 1 being at the extreme left and column 39 at the extreme right. Each rectangle on the screen is identified by a pair of numbers, indicating the row and column. For example, we have indicated the rectangle in the twelfth row and sixteenth column in Figure 6-2 (page 130).

A graphics rectangle is always specified by giving the column number first.

The low resolution graphics command **GR** blacks out the top 20 lines of the screen, giving a 40 x 40 grid of rectangles for plotting. The bottom 4 lines may be used for text. The plotting color is automatically set to black by the **GR** command. This means that you won't be able to see anything you plot since it will appear as black on black. The first chore on entering low resolution graphics mode is to choose the color. Sixteen colors are possible, numbered as shown here and on page 131.

0-black	2-dark blue
1-magenta	3-purple

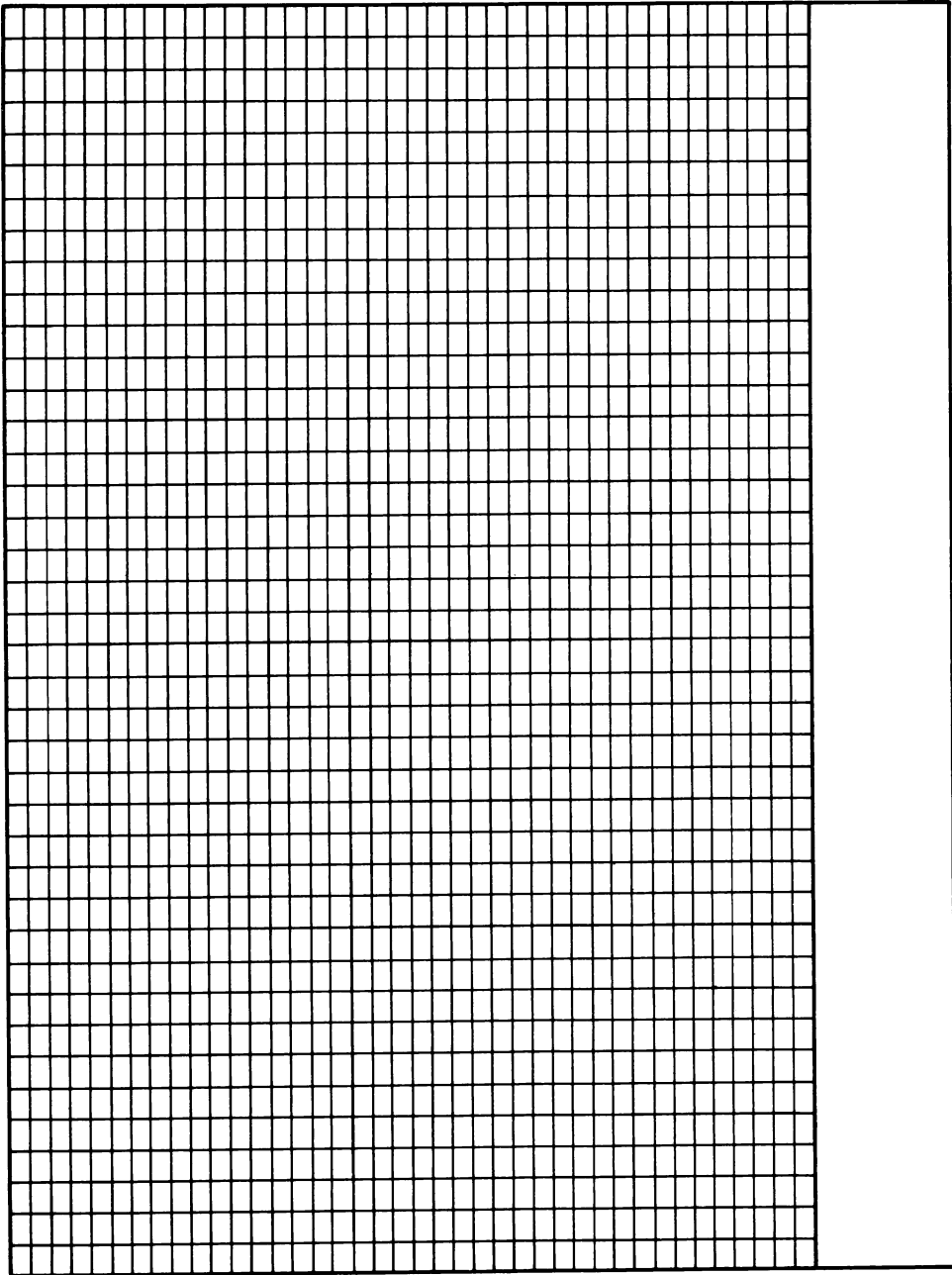


Figure 6-1. Screen layout for low resolution graphics mode

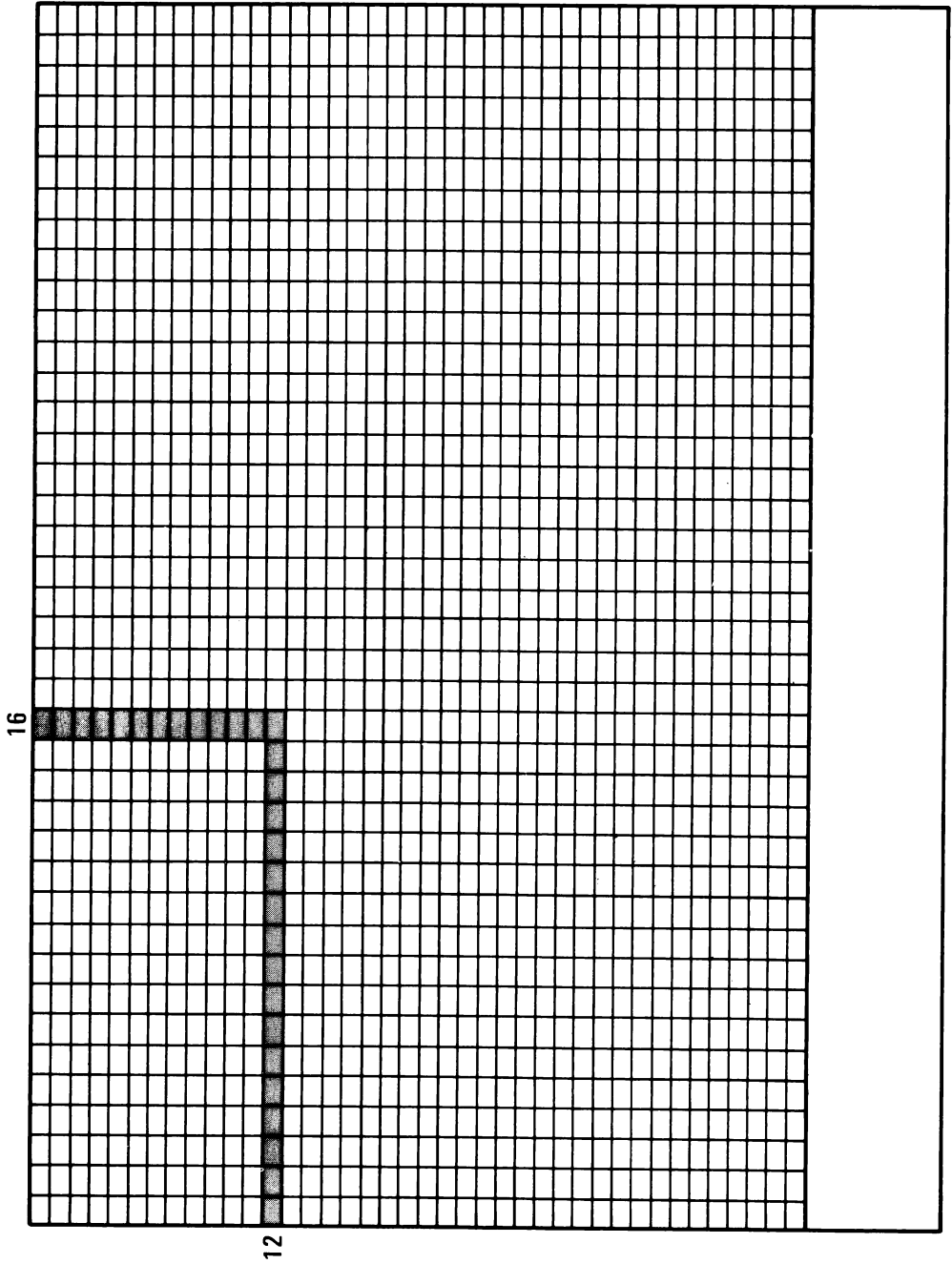


Figure 6-2.

4-dark green	10-grey
5-grey	11-pink
6-medium blue	12-green
7-light blue	13-yellow
8-brown	14-aqua
9-orange	15-white

To set the color, we use an instruction of the form

COLOR=11

Once the color has been set, all subsequent plotting will be in that color until you give another **COLOR** instruction.

To light up the rectangle at position (x,y) (in the current color), we use the graphics instruction

10 PLOT x,y

Here x denotes the column number and y denotes the row number. To turn off the rectangle, we plot it in color 0 (black).

TEST YOUR UNDERSTANDING 1 (answer on page 133)

Write a set of instructions to illuminate the graphics rectangle at row 18, column 22 in color 12.

TEST YOUR UNDERSTANDING 2 (answer on page 133)

Write a set of instructions to blink the graphics rectangle at row 18, column 22 in color 12.

Low resolution graphics has special commands for drawing horizontal and vertical lines. To draw a horizontal line from column 2, row 11 to column 20, row 11, we use an instruction of the form

20 HLIN 2,20 AT 11

Similarly, to draw a vertical line from column 5, row 1 to column 5, row 20, we use an instruction of the form

30 VLIN 1,20 AT 5

Example 1. Write a program which draws a dark green horizontal line across row 10 of the screen.

Solution.

```

10 GR
20 COLOR=4
30 HLIN 0,39 AT 10
40 END

```

Example 2. Write a program which draws a dark green vertical line in column 25 from row 5 to row 15. The program should blink the line 50 times.

Solution. The blinking effect may be achieved by repeatedly clearing the screen. Here is our program.

```

10 GR
20 FOR K=1 TO 50:REM K CONTROLS BLINKING
30 COLOR=4
40 VLIN 5,15 AT 25
50 FOR K=1 TO 500: NEXT K
60 COLOR=0: VLIN 5, 15 AT 25
70 FOR K=1 TO 500: NEXT K
80 NEXT K
90 END

```

<p>TEST YOUR UNDERSTANDING 3 (answer on page 133)</p> <p>Write a program to draw a vertical line from row 2 to row 20 in column 8.</p>

Example 3. Draw a pair of x and y axes as shown in Figure 6-3. Draw the axes in light blue.

Solution.

```

10 GR
20 COLOR=7
30 HLIN 0,39 AT 39
40 VLIN 0,39 AT 0
50 END

```

Your program may determine the color of a given rectangle on the screen using the **SCRN** instruction. For example, **SCRN(7,3)** is equal to the color number of rectangle 7,3.

EXERCISES (answers on page 239)

Draw the following straight lines.

1. A horizontal line completely across the screen in row 18.
2. A vertical line completely up and down the screen in column 17.

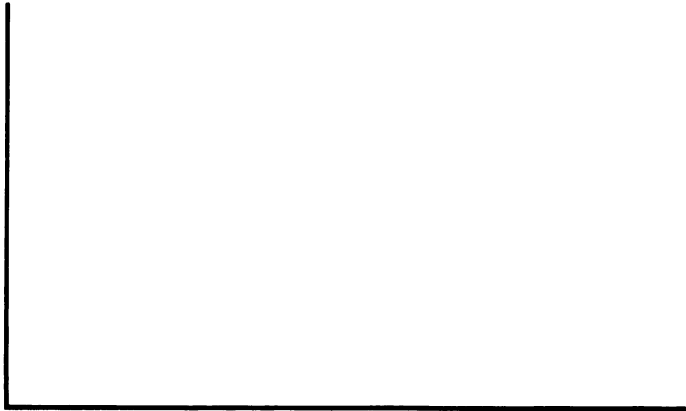
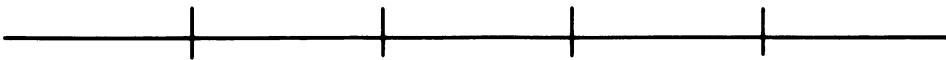


Figure 6-3.

3. A pair of straight lines which divide the screen into four equal squares.
4. Horizontal and vertical lines which convert the screen into a tic tac toe board.
5. A vertical line of double thickness from rows 1 to 24 in column 30.
6. A diagonal line going through the character positions (1,1), (2,2), ..., (24,24).
7. A horizontal line with "tick marks" as follows:



8. A vertical line with "tick marks" as follows:



ANSWERS TO TEST YOUR UNDERSTANDING 1,2, and 3

<pre>1: 10 GR 20 COLOR = 12 30 PLOT 22,18 40 END</pre>	<pre>2: 10 GR 20 COLOR = 12 30 PLOT 22,18 40 COLOR = 0: PLOT 22,18 50 GOTO 20 60 END</pre>	<pre>3: 10 GR 20 COLOR = 1 30 VLIN 2,20 AT 8 40 END</pre>
--	--	---

6.2 HIGH RESOLUTION GRAPHICS

In principle, high resolution graphics on the Apple II Computer works similarly to low resolution graphics, except that the screen is divided into 280 columns (0-279) and 160 rows (0-159). High resolution graphics mode is turned on with the instruction

HGR

Color in high resolution mode is set via the instruction

HCOLOR

However, only eight colors are available in high resolution graphics. They are numbered 0-7 (just which colors these are will depend on your particular television set.)

In high resolution graphics, point plotting is accomplished via the instruction

HPLOT

In addition to point plotting, high resolution graphics has provisions for defining and displaying elaborate shapes using shape tables stored in RAM. These shapes may be moved and even rotated using simple instructions. A complete discussion of high resolution graphics is beyond the scope of this book. We refer you to your **Applesoft BASIC Programming Reference Manual** for further information.

6.3 COMPUTER ART

The computer may be used to create interesting pieces of graphics art. In this book, we will not dwell on this application too much, but it is interesting and certainly deserves a mention.

We will discuss two forms of computer art. The first arises when we give the computer free "artistic license" to create random patterns on the screen. Let us create a program which runs down the rows of graphics blocks and randomly illuminates some of them. We will control the number of blocks to be illuminated by a "density factor" which we input. This factor will be a number between 0 and 1 and will be denoted by the variable D (for density). Our program will consider each graphics block separately. It will use the output of RND to make the decision on whether to illuminate the block or not. Namely, if RND is less than D, the block will be illuminated; if RND is at least D then the block will not be illuminated. Here is our program.

```

3 HOME
5 INPUT "DENSITY FACTOR "; D
6 GR
10 FOR R=0 TO 39 : REM R=ROW NUMBER
20 FOR C=0 TO 39: REM C=COLUMN NUMBER
30 COLOR=INT(16*RND(1))
40 IF RND(1) < D THEN 60
50 GOTO 100
60 PLOT C,R
100 NEXT C
110 NEXT R
200 END

```

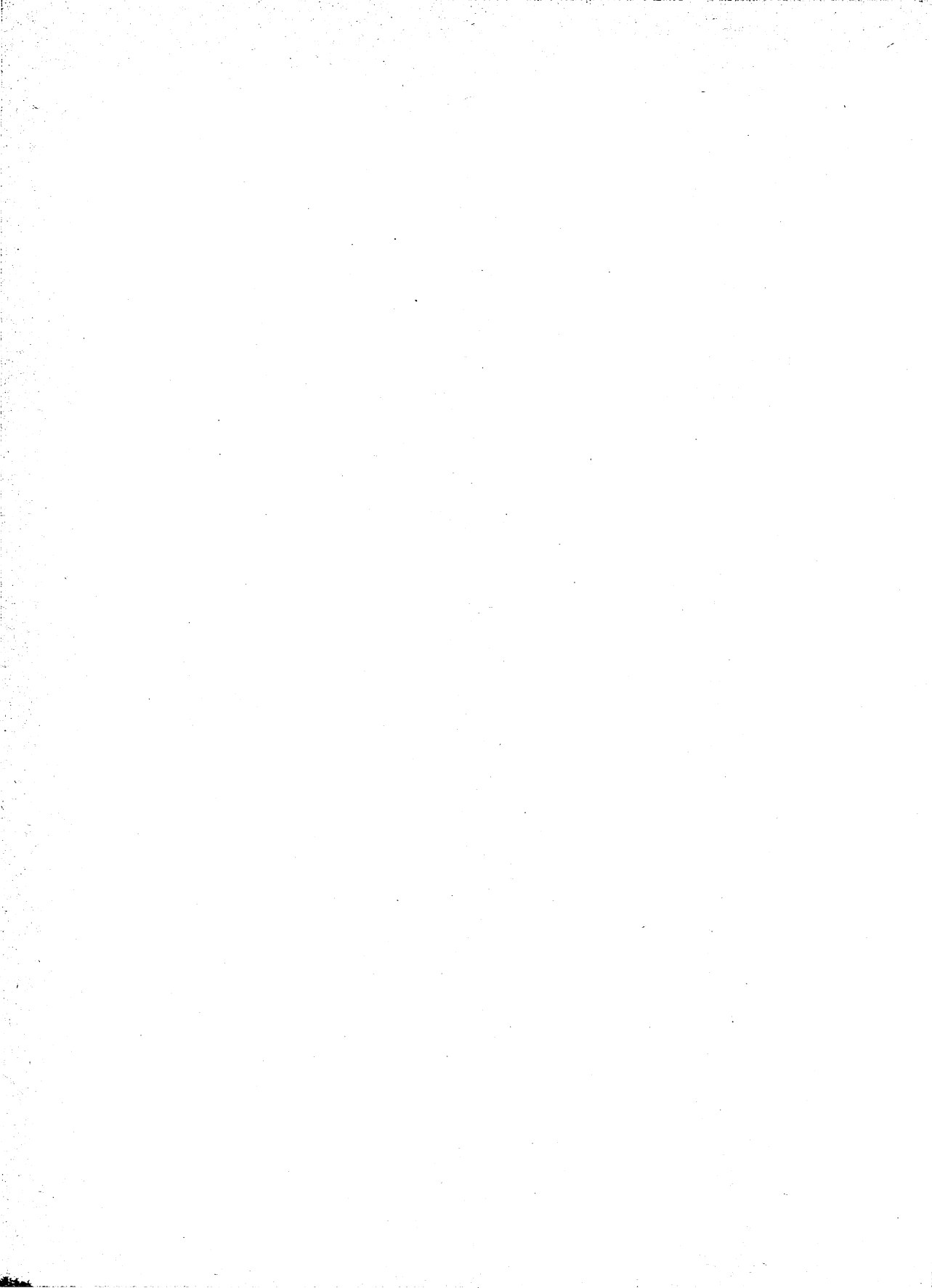
You should run this program for assorted values of A. Some suggested values are $A = .1$, $.5$, and $.7$, respectively. Note that because of the unpredictability of the random number generator, the same value of A will usually yield quite different pictures on successive runs.

A second method of generating art using the computer is to use a graphics pad to effectively "trace" a picture. For example, suppose that your picture is a photograph. Place a video display worksheet over the picture. Fill in all rectangles which touch the subject of the photograph. In this way, you will create an impression of the subject which you may then display on the screen.

To close our very brief discussion, we should mention some related equipment which has recently come within the range of the computer hobbyist. In our above description of tracing, we suggested a rather laborious procedure. However, there are special devices called digitizing pads which enable you to trace a shape with an electronic "pen" and have the same shape transferred to the screen. In addition, there are the so-called light pens which enable you to touch a point on the screen and have the computer read the location of the point. This sort of device can be useful in creating computer art as well as in playing computer games.

EXERCISES (answers on page 240)

1. Run the above program three times for the value $A = .4$.
2. Run the above program for the values $A = .1, .2, .3, . . . , 1.0$. Can you predict the display for $A = 1.0$ in advance?
3. Create a computer impression of a member of your family using a large photograph. ($5'' \times 7''$ or larger will work best. Use high resolution graphics.)



7

Word Processing

7.1 WHAT IS WORD PROCESSING?

Microcomputers are currently causing an office revolution. As microcomputers become cheaper and easier to use, they are finding their way into every aspect of business. Nowhere does the revolutionary impact of microcomputers promise to be greater than in the area of word processing. In brief, a **word processor** is a device made by combining the traditional typewriter with the capabilities of the computer for storing, editing, retrieving, displaying, and printing information. It is no exaggeration to say that the traditional typewriter is now as obsolete as a Model T. Over the next decade or so the typewriter will be completely replaced by increasingly sophisticated word processors.

The basic concept of a word processor is to use the microcomputer as a typewriter. However, instead of using paper to record the words, we use the computer memory. First, the words are stored in RAM. When you wish to make a permanent record of them, you store them on disk as a data file. As you type, the text can be viewed on the video display. This part of word processing is not revolutionary. The true power of a word processor doesn't come into play until you need to edit the data in a document. Using the power of the computer, you can perform the following tasks quickly and with little effort: Move to any point in the document; add words, phrases, sentences, or even paragraphs; delete portions of the text; move a block of text from one part of the document to another; insert "boiler-plate" information (standard pieces of text such as resumes or company descriptions) from another data file (for example, you could add a name and address from a mailing list); selectively change all occurrences of one word (say, "John") to another

(say, "Jim"); or print the contents of a file according to a requested format.

All of the above operations are possible since the computer is able to manipulate strings in addition to numbers. Actually, your Apple II Computer is equipped with a wide variety of commands to manipulate string data. In fact, you may turn your Apple II Computer into quite a respectable word processor.

In this chapter, we will discuss the Applesoft BASIC instructions for string manipulation. We will also discuss formatting output on a printer. Next, we will discuss the features available in commercially available word processing packages which you can purchase for your Apple II Computer. Finally, to give you a taste of actual word processing, we will build a rudimentary word processor which you can use to prepare letters, term papers, memos, or other documents.

7.2 MANIPULATING STRINGS

ASCII CHARACTER CODES

Each keyboard character is assigned a number between 1 and 255. The code number thus assigned is called the *ASCII code* of the character. For example, the letter "A" corresponds to the number 65, while the number 97 corresponds to the letter "a." Also included in this correspondence are the punctuation marks and other keyboard characters. As examples, 40 corresponds to the symbol "(" and 62 corresponds to the symbol ">". Even the various control keys have corresponding numbers. For example, the space bar corresponds to the number 32, and the RETURN key to number 13. Table 7-1 lists all of the printable characters and their corresponding ASCII codes. We will discuss the various control codes in Section 7.4 (page 150).

Table 7-1. ASCII Character Codes for Printable Characters.*

ASCII Code	Character	ASCII Code	Character
32	blank		
33	!	37	%
34	"	38	&
35	#	39	'
36	\$	40	(

Table 7-1. ASCII Character Codes for Printable Characters. (Continued)

ASCII Code	Character	ASCII Code	Character
41)	84	T
42	*	85	U
43	+	86	V
44	,	87	W
45	—	88	X
46	.	89	Y
47	/	90	Z
48	0	91	[
49	1	92	\
50	2	93]
51	3	94	^
52	4	95	—
53	5	96	,
54	6	97	a
55	7	98	b
56	8	99	c
57	9	100	d
58	:	101	e
59	;	102	f
60	<	103	g
61	=	104	h
62	>	105	i
63	?	106	j
64	@	107	k
65	A	108	l
66	B	109	m
67	C	110	n
68	D	111	o
69	E	112	p
70	F	113	q
71	G	114	r
72	H	115	s
73	I	116	t
74	J	117	u
75	K	118	v
76	L	119	w
77	M	120	x
78	N	121	y
79	O	122	z
80	P	123	{
81	Q	124	
82	R	125	}
83	S	126	~

*Note that ASCII codes 91-126 are not displayed by the Apple II. However, most printers will print all the codes.

Note that the Apple II keyboard is not capable of generating the symbols corresponding to all the ASCII codes. For example, there are no keys to generate lower case letters. However, you may specify any ASCII symbols in your program, using the appropriate code numbers. For example, you can insert lower case letters in a diskette file or send lower case letters to the printer.

The computer uses ASCII codes to refer to letters and control operations. Any file, whether it is a program or data may be reduced to a sequence of ASCII codes. Consider the following address.

John Jones
2 S. Broadway

As a sequence of ASCII codes, it would be stored as

74,111,104,110,32,74,111,110,101,115,13,10

50,32,83,46,32,66,114,111,,97,100,119,97,121,13

Note that the spaces are included (numbers 32) as are the carriage returns (RETURN) and line advances (numbers 13 and 10). ASCII codes allow us to describe any piece of text generated by the keyboard. This includes all formatting instructions like spaces, carriage returns, upper and lower case letters, and so forth. Moreover, once a piece of text has been reduced to a sequence of ASCII codes, it may also be faithfully reproduced on the screen or on a printer. This is the fundamental principle underlying the design of word processors.

TEST YOUR UNDERSTANDING 1 (answer on page 146)

Write a sequence of ASCII codes which will reproduce this ad:

FOR SALE: Beagle puppies. Pedigreed.
8 weeks. \$125.

You may refer to characters by their ASCII codes by using CHR\$. For example, CHR\$(74) is the character corresponding to ASCII code 74 (upper case J); CHR\$(32) is the character corresponding to ASCII code 32 (space). The **PRINT** instruction may be used in connection with CHR\$. For example, the instruction

10 PRINT CHR\$(74)

will display an upper case J in the first position of the first print field.

TEST YOUR UNDERSTANDING 2 (answer on page 146)

Write a program which will print the ad of Test Your Understanding 1 from its ASCII codes.

To obtain the ASCII code of a character, use the instruction **ASC**. For example, the instruction

```
20 PRINT ASC("B")
```

will print the ASCII code of the character "B", namely 66. In place of "B", you may use any string. The computer will return the ASCII code of the first character of the string. For example, the instruction

```
30 PRINT ASC(A$)
```

will print the ASCII code of the first character of the string A\$. If A\$ is the empty string "" (the string consisting of no characters), then using ASC(A\$) will result in an ILLEGAL QUANTITY ERROR.

TEST YOUR UNDERSTANDING 3 (answer on page 146)

Determine the ASCII codes of the characters \$, G, X, + without looking at the chart.

You may compute the length of a string by using the **LEN** instruction. For example, **LEN("BOUGHT")** is equal to six since the string "BOUGHT" has six letters. Similarly, if A\$ is equal to the string "Family Income", then **LEN(A\$)** is equal to 13. (The space between the words counts!) Here is an application of the **LEN** instruction.

Example 1. Write a program which inputs the string A\$ and then centers it on a line of the display.

Solution. A line is 40 characters long, with the spaces numbered from 1 to 40. The string A\$ takes up **LEN(A\$)** of these spaces, so there are $40 - \text{LEN}(A\$)$ spaces to be distributed on either side of A\$. The line should begin with half of the $40 - \text{LEN}(A\$)$ spaces, or with $(40 - \text{LEN}(A\$))/2$ spaces. So we should tab to column $(40 - \text{LEN}(A\$))/2 + 1$. Here is our program.

```
10 INPUT A$
20 HOME
30 PRINT TAB((40-LEN(A$))/2+1) A$
40 END
```

TEST YOUR UNDERSTANDING 4 (answer on page 146)

Use the program of Example 1 to center the string "THE APPLE II COMPUTER".

MORE ABOUT STRINGS

A string may consist of as many as 255 characters, but you cannot type more than 40 characters on a line. However, you may type strings containing more than 40 characters by continuing to type *without hitting the RETURN key*. When a line is filled, the computer will automatically place the next letter at the beginning of the next line. However, if you do not hit the RETURN key at the end of the line, then the next line will be a continuation of the first. It is necessary to have long strings if you wish to be able to print hard copy. Most line printers have at least 80 characters per line and some accommodate 132 character lines.

There are a number of operations which may be performed on strings. First, strings may be "added" (or, in computer jargon, "concatenated"). Suppose that we have strings A\$ and B\$, with A\$ = "WORD" and B\$ = "PROCESSOR". Then the sum of A\$ and B\$, denoted A\$ + B\$, is the string obtained by adjoining A\$ and B\$, namely:

"WORDPROCESSOR"

Note that no space is left between the two strings. To include a space, suppose that C\$ = " ". C\$ is the string which consists of a single space. Then A\$ + C\$ + B\$ is the string

"WORD PROCESSOR"

TEST YOUR UNDERSTANDING 5 (answer on page 146)

If A\$ = "4" and B\$ = "7", what is A\$+B\$?

The computer handles relations among strings in much the same way that it handles relations among numbers. For example, we say that the strings A\$ and B\$ are equal, written A\$ = B\$, provided that they consist of exactly the same characters, in the same order. Otherwise the strings are unequal, written A\$ <> B\$ or A\$ >< B\$. The notation A\$ < B\$ means that A\$ precedes B\$ in alphabetical order. (This is fine for strings

consisting only of letters. For numbers or other characters, the ASCII codes of the characters are used to determine order.) Similarly, $A\$ > B\$$ means that $A\$$ succeeds $B\$$ in alphabetical order. For example, the following are true relations among strings:

“bear” < “goat”

“girl” > “boy”

The notation $A\$ \geq B\$$ means that either $A\$ > B\$$ or $A\$ = B\$$. Simply, this means that $A\$$ either succeeds $B\$$ in alphabetical order or $A\$$ and $B\$$ are the same. The notation $A\$ \leq B\$$ has a similar meaning.

Example 2. Write a program which alphabetizes the following list of words: egg, celery, ball, bag, glove, coat, pants, suit, clover, weed, grass, cow, and chicken.

Solution. We set up a string array $A\$(J)$ which contains these 13 words. We set $B\$$ equal to the first word. We successively compare $B\$$ with each of the words in the array. If any compared word precedes $B\$$, we replace $B\$$ with that word. At the end of the comparisons, $B\$$ contains the first word in alphabetical order. We place this as the first item in the array $C\$(J)$. We now repeat the process with the first word eliminated. This gives us the second word in alphabetical order, and so forth. Here is our program.

```

10 DIM A$(13),D$(13),C$(13)
20 DATA EGG,CELERY,BALL,BAG,GLOVE,COAT
30 DATA PANTS,SUIT,CLOVER,WEED,GRASS
40 DATA COW,CHICKEN
50 FOR J=1 TO 13
60 READ A$(J):REM SET UP ARRAY A$
70 NEXT J
80 FOR K=1 TO 13:REM FIND KTH WORD IN ORDER
85 REM 90-200 CREATE AN ARRAY
86 REM D$ CONSISTING OF THE
87 REM WORDS YET TO BE ALPHABETIZED
90 L=1
100 FOR J=1 TO 13
110 E=0
120 FOR M=1 TO 13
130 IF A$(J)=C$(M) THEN E=1
140 NEXT M
150 IF E=0 THEN 160
155 GOTO 200
160 D$(L)=A$(J)
170 L=L+1
200 NEXT J
210 B$=D$(1)
220 FOR L=1 TO 13-K+1

```

```

230 IF D$(L)<B$ THEN 250
235 GOTO 300
250 B$=D$(L)
300 NEXT L
400 C$(K)=B$
410 NEXT K
500 FOR K=1 TO 13
510 PRINT C$(K)
520 NEXT K
600 END

```

This program can be modified to make a program alphabetizing any collection of strings. We will leave the details to the exercises.

It is possible to dissect strings using the three instructions **LEFT\$**, **RIGHT\$**, and **MID\$**. These instructions allow you to construct a string consisting of a specified number of characters taken from the left, right, or middle of a designated string. Consider the instruction:

```
10 LET A$=LEFT$("LOVE",2)
```

The string A\$ consists of the two leftmost characters of the string "LOVE". That is, A\$ = "LO". Similarly, the instructions

```
20 LET B$="TENNIS"
30 LET C$=RIGHT$(B$,3)
```

set C\$ equal to the string consisting of the three rightmost letters of the string B\$, namely C\$ = "NIS". Similarly, if A\$ = "REPUBLICAN", then the instruction

```
40 LET D$=MID$(A$,5,3)
```

sets D\$ equal to the string which consists of three characters starting with the fifth character of A\$, which is D\$ = "BLI". These last three instructions are absolutely fundamental in the design of word processors. For example, they may be used to "form lines," one part of a word processing program.

TEST YOUR UNDERSTANDING 6 (answer on page 146)

Determine the string constant

```
RIGHT$(LEFT$("COMPUTER",4),3)
```

In manipulating strings, it is important to recognize the difference between numerical data and string data. The number 14 is denoted by 14; the string consisting of the two characters 14 is denoted "14". The first is a numerical constant and the second a string constant. We can perform arithmetic using the numerical constants. However, we cannot perform any of the character manipulation supplied by the instructions

RIGHT\$, MID\$, and LEFT\$. Such manipulation may only be performed on strings. How may we perform character manipulation on numerical constants? BASIC provides a simple method. We first convert the numerical constants to string constants by using **STR\$**. For example, the number 14 may be converted into the string "14" using the instruction:

```
10 LET A$=STR$(14)
```

As a result of this instruction, A\$ has the value "14". As another example, suppose that the variable B has the value 1.457. **STR\$(B)** is then equal to the string "1.457".

To convert strings consisting of numbers into numerical constants, use **VAL**. Consider this instruction:

```
20 LET B=VAL("3.78")
```

This instruction sets B equal to 3.78.

TEST YOUR UNDERSTANDING 7 (answer on page 146)

Suppose that A\$ equals "5 PERCENT" and B\$ equals "758.45 DOLLARS". Write a program which starts from A\$ and B\$ and computes five percent of \$758.45.

EXERCISES (answers on page 240)

1. Use the program of Example 2 (page 143) to alphabetize the following sequence of words: justify, center, proof, character, capitalize, search, replace, indent, store, and password.

2. Write a program to form the following string into lines.

Word processing will revolutionize the office. Already, millions of word processing systems are in use. By the end of the decade, the typewriter will be totally obsolete. Word processing systems will increase in sophistication.

3. Write a program which rewrites the addition problem $15 + 48 + 97 = 160$ in the form

$$\begin{array}{r} 15 \\ 48 \\ 97 \\ \hline 160 \end{array}$$

4. Write a program which inputs the string constants "\$6718.49" and "\$4801.96" and calculates the sum of the given dollar amounts.

ANSWERS TO TEST YOUR UNDERSTANDING 1, 2, 3, 4, 5, 6, and 7

1: 70, 79, 82, 32, 83, 65, 76, 69, 58, 32, 66, 101, 97, 103, 108, 101, 32, 112, 117, 112, 112, 105, 101, 115, 46, 32, 13, 10, 56, 32, 119, 101, 101, 107, 115, 46, 32, 36, 49, 50, 53, 46, 13, 10

2: 10 DATA 70,79,.....(insert data from TYU 1 above)
 11 DATA(insert data from TYU1 above)
 12 DATA(insert data from TYU1 above)
 20 FOR J=1 TO 45
 30 READ A(J)
 40 PRINT CHR\$(A(J));
 50 NEXT J
 60 END

3: 10 DATA \$,G,X,+
 20 FOR J=1 TO 4
 30 READ A\$(J)
 40 B(J)=ASC(A\$(J))
 50 PRINT A\$(J),B(J)
 60 NEXT J
 70 END

4: Type RUN followed by the given string.

5: 47

6: "omp"

7: 10 A\$="5 percent":B\$="758.45 dollars"
 20 A=VAL(A\$):B=VAL(B\$)
 30 PRINT A\$,"OF",B\$,"IS"
 40 PRINT A*B*.01
 50 END

7.3 PRINTER CONTROLS AND FORM LETTERS

Recall what we have covered up to this point concerning the printer. The printer is controlled by a circuit board (either a parallel or serial card) placed in one of the slots of the Apple II Computer. For clarity,

let's assume that the card controlling the printer is placed in slot 1. The instruction

10 PR#1

will then direct all further **PRINT** statements to the printer rather than the screen. To go back to displaying the results of **PRINT** statements on the screen, we use the instruction

20 PR#0

The printer accepts a stream of ASCII character codes. Some of these correspond to letters and symbols to be printed, and some correspond to control characters which make the printer perform various non-print functions (such as carriage return, line feed, space to the top of the next page, and so forth). To the printer, a line consists of a sequence of printable characters followed by a **RETURN**. The **RETURN** actually tells the printer two things. First, it causes a carriage return; second, it causes the paper to advance one line (a so-called **line feed**).

A typical printer has controls for setting a number of different parameters. Among the switches which can be set are those which control line width, margins, number of lines per inch, the number of lines per page, and the size of the characters. We refer you to your printer manual for the correct procedures for setting these switches. (In the case of some printers, these various settings are controlled by sequences of characters from the computer. These so-called **ESCAPE sequences** are printed as if they were actual data. However, the printer recognizes them as commands and does not print them.)

FORM LETTERS

You may use the string manipulating capability of your computer to prepare form letters that look like genuine correspondence. Let us illustrate the technique by preparing the following form letter.

April 1, 1981

Dear :

All of us at Neighborhood Building Supplies, Inc. have appreciated your patronage in the past. We are writing to let you know that we will move our store to 110 S. Main St., effective July 1. You will find the new store larger and more convenient than the old. In addition, we will now stock a more extensive line of energy-efficient doors and windows. We look forward to your continued pa-

tronage. Please let us know if we may be of assistance in your building plans.

Cordially yours,

Samuel Gordon,
President
Neighborhood Building Supplies, Inc.

Suppose that this letter is to go out to 100 customers on a mailing list. Also assume that the mailing list is maintained in a data file on diskette and that each address is stored as four strings, corresponding to the name of the individual, company name, street address, and city-state-zipcode. Finally, suppose that the name of the file is CUSTOMER. Write a program to produce the desired stack of 100 letters.

Our program will consist of two parts. The first will allow us to type in the body of the letter. The various lines of the letter will be typed exactly like we were typing on a typewriter. The computer will use a string array A\$(J) to store the body of the letter, with A\$(1) holding the first line, A\$(2) the second, and so forth. We will indicate to the computer that the body of the text is complete by typing % followed by RETURN.

As soon as the character % is recognized, the program goes into its second phase, namely the actual generation of the letters. The program opens the address file for output. One by one it reads the address entries. After reading a given entry, it prints the date at the top of the letter, followed by the address. Next, the program determines the last name of the addressee from the first line of the address and inserts it after the "Dear". Finally, the body of the letter is printed. Here is the program which accomplishes all of this.

```

10 DIM A$(100)
20 PRINT "AFTER EACH ? TYPE ONE LINE OF THE LETTER,"
30 PRINT "FOLLOWED BY RETURN"
40 PRINT "TO END LETTER, TYPE % FOLLOWED BY RETURN"
50 J=1 : REM J IS THE NUMBER OF LINES
60 INPUT A$(J)
70 IF A$(J)="" THEN 100
75 GOTO 80
80 J=J+1: REM NEXT LINE
90 GOTO 60
100 PRINT CHR$(4); "OPEN CUSTOMER"
105 PRINT CHR$(4); "READ CUSTOMER"
110 FOR N=1 TO 100: REM N=CUSTOMER #
120 INPUT B$(1,N), B$(2,N), B$(3,N), B$(4,N)
130 NEXT N :REM GO TO NEXT CUSTOMER

```

```

140 PRINT CHR$(4); "CLOSE CUSTOMER"
150 PR#1: REM ADDRESS PRINTER
155 FOR N=1 TO 100
160 PRINT A$(1): REM PRINT DATE
170 PRINT: REM PRINT BLANK LINE
175 REM 150-180 PRINT ADDRESS
180 PRINT B$(1,N)
181 PRINT B$(2,N)
182 PRINT B$(3,N)
183 PRINT B$(4,N)
190 REM FIND LAST NAME OF ADDRESSEE
200 LET L=LEN(B$(1,N))
210 FOR K=0 TO L-1
220 LET C$=MID$(B$(1,N),L-K,1)
230 IF C$=" " THEN 300
235 GOTO 240: REM TEST FOR SPACE BETWEEN WORDS
240 NEXT K
300 LET D$=RIGHT$(B$(1,N),K): D$=REM LAST NAME
310 PRINT "Dear Mr. "; D$; ","
320 FOR M=2 TO J :REM PRINT BODY OF LETTER
330 PRINT A$(M)
340 NEXT M: REM GO TO NEXT LINE OF LETTER
350 NEXT N: REM GO TO NEXT CUSTOMER
360 PR#0: TURN SCREEN ON
400 END

```

Note that for the above program to work properly, you should type the date in as the first line of the letter. You should not type in the line which begins "Dear". The program generates this line for you. Also note that this program always addresses the customer as "Mr." This will insult a certain number of your customers. Suppose that your customer entries in the address list are labelled with "Mr.", "Mrs." or "Ms." preceding the name. Can you modify the above program to insert the correct title in the salutation of the letter?

The above program was used in the context of a specific letter. Please note that the program may be used to generate any set of form letters from an address list. In the exercises, we will suggest some modifications which you can use to generate invoices or other correspondence with variable text in the body of the letter.

EXERCISES

1. Add to the form letter of the text a second page. At the top of the page should be a date and the page number. The title should be "OPENING SPECIAL". The text should consist of the following message:

This letter is being sent only to our most valued customers! Bring this coupon with you for a 10 percent discount on any order placed in the month of JULY, 1981.

2. Change the form letter of the text so that in the third and fourth sentences, the name of the addressee is used (for example, "Ms. Thomas, you will find . . .")
3. Write a program that prints invoices. Assume that the invoices are stored in a file called "INVOICE", where a particular invoice contains for each item shipped: quantity, price, item description (limited to 15 characters), and total cost. Assume that the invoice entry starts with a four-string entry giving the customer name, address, and date. The file entry for a given invoice ends with the character %. You may assume that the first entry in the file is the number of invoices contained in the file. Your program should print out invoices corresponding to all entries in the file.
4. Suppose that the file INVOICE of Exercise 3 contains only a customer identification number rather than a customer name and address. Suppose that the whole list of customer addresses contains customer identification numbers. Modify the program of Exercise 3 so that it locates the customer name and address automatically.
5. Assume that a change in local ordinances now allows you to not charge local sales tax to any customer who lives outside the city limits. Suppose that city consists of ZIP CODE 91723. Modify the program of the text so that it checks the ZIP CODE of the customer. For customers not in ZIP CODE 91723, insert the following paragraph in the letter:

Good news! You will no longer be charged local sales tax, in accordance with the change in local ordinances. This will yield even further savings from our already low prices.

7.4 USING YOUR COMPUTER AS A WORD PROCESSOR

So far in this chapter, we have introduced you to the various text manipulation features of the Apple II Computer. We have mentioned "word processing" quite often and have attempted to give you a smat-

tering of word processing which you can accomplish using homegrown programs. However, your computer is capable of quite a bit more. More, in fact, than we can possibly describe in this introductory book and more than a beginning programmer can expect to accomplish at this stage of learning. Although we cannot go into great detail, we will close this chapter with a description of some of the word processing you can expect your computer to accomplish using commercially available software.

Let's begin by describing some of the typical features of a word processing system you can run on your Apple II Computer. A word processing system is a computer program for creating, storing, and editing text.

At its most basic level, you use a word processing system like you would use a typewriter. Suppose that you wish to prepare a proposal. You would turn on the computer and run the word processing program. The program first asks for the type of work you would like to perform. Possibilities include: type in a new document, edit an old document, save a document on diskette, or print a document. We would select the first option. Next we would describe various format parameters to the word processor: line width, number of characters per inch, number of lines per page, spacing between lines, and so forth.

We would then type the document exactly like we would on a typewriter. There are several huge exceptions, however! First of all, we don't worry about carriage returns. The word processor takes care of forming lines. It accepts the text we type, decides how much can go on a line, forms the line, and displays it. Any text left over is automatically saved for the next line. The only function of the carriage return is to indicate a place where you definitely want a new line, such as at the end of a paragraph.

A second advantage of a word processor is in correcting errors. To correct an error, we move the cursor to the site of the mistake, give a command to erase the erroneous letter(s) or word(s), and type in the replacements. Of course, such action will generally destroy the structure of the lines. (Some lines may now be too long and others too short.) By using a simple command, it is possible to "reform" the lines according to the requested format.

Typically, a word processor has commands which enable you to scroll through the text of a document to look for a particular paragraph. Some word processors even allow you to mark certain points so that you may turn to them without a visual search.

When the document is finally typed to your satisfaction, you give the computer an instruction which saves a copy of it on diskette. At a future time, you may recall the document, and add to it at any point (even within the bodies of paragraphs!). Typically, word processors have certain "block operations" which allow you to "mark" a block and then either delete it, copy it, or move it to another part of the document. You may also insert other documents into the current document. This is convenient, for example, in adding boiler plate, such as resumes, to your document. You may even use the block operations to alter boiler plate to fit the special needs of the current document.

You may construct your document in as many sessions as you wish. When your diskette finally contains the document as you want it, you finally give the instruction to print. Your printer will now produce a finished, error-free copy of the document.

As if the above were not enough of an improvement over the conventional typewriter, the typical word processor can do even more. The features available depend, of course, on the word processor selected. Here are some of the goodies to look for:

Global Search and Replace. Suppose you wish to resubmit your document to another company, Acme Energetics. In your original document, you included numerous references to the original company, Jet Energetics. A global search and replace feature allows you to instruct the computer to replace every occurrence of a particular phrase with another phrase. For example, we could replace every occurrence of "Jet Energetics" with "Acme Energetics". Global search and replace can be even more sophisticated. In some systems, the word processor can be instructed to ask you whether or not to make each individual change. Another variation is to instruct the word processor to match any capitalization in the phrases replaced.

Centering. After typing a line you may center it using a simple command.

Boldface. You may print certain words in darker type.

Underscore. You may indicate underscoring portions of text.

Subscripts and Superscripts. You may indicate printing of subscripts (as in a_1) and superscripts (as in a^2). This is extremely useful for scientific typing.

Justification. You may instruct the word processor to "justify" the right hand margins of your text, so that the text always ends exactly at the end of a line. This is possible only if you have a printer which is

capable of spacing in increments smaller than the width of a single letter.

Spelling Correction. There now exist a number of spelling correction programs which compare words of your document against a dictionary (sizes range from 20,000 to 70,000 words). If the program doesn't find a match, it asks you if the word is spelled correctly and gives you an opportunity to add the word to the dictionary. In this way the output of a word processor can be proofread by computer.

APPLEWRITER*

The easiest way to get into word processing on your Apple II Computer is to use the Applewriter word processing program, available from your Apple II Computer dealer. This program has been specially designed for the Apple II Computer and should be first on your list of software purchases.

7.5 A DO-IT-YOURSELF WORD PROCESSOR

It is really quite impractical for you to build your own word processor. For one thing, such a program is long and complicated. Moreover, if you write in BASIC, the operation of the program will tend to be rather slow. An efficient word processor will almost always be written in machine language. Nevertheless, in order to acquaint you with a few of the virtues of word processing, let's ignore what we just said and build a word processor anyway!

Our word processor will be line oriented. You will type in each line just as if you are typing it on a typewriter. At the end of each line, you will give a carriage return by typing RETURN. The Jth line will be stored in the string variable A\$(J). Let's assume you have a 16K version of the Apple II Computer. This will allow us to store and edit a document of about five double-spaced, typed pages. Our word processor will have five modes. In the first mode, we input the text of our document. This operation will proceed exactly as if you were typing on a typewriter. At the beginning of each line, the word processor will display a ?. You type your line after the question mark. Terminate the line with RETURN. In

*Applewriter is a registered trademark of Apple Computer Corp.

order to indicate that you don't wish to type any more lines, type % followed by **RETURN**.

A second mode allows us to save our document. For the purposes of this word processor, let's assume that you have a diskette file. The program then saves your document as a data file under a file name requested by the program. The first item in a document file will always be the number of lines in the document. This quantity will be denoted by the variable L. Next come the lines of the document: A\$(1), A\$(2), . . . , A\$(L).

A third mode lets you produce a draft version of the document. In this mode, the document is printed with each line preceded by its line number. The line numbers allow you to easily identify lines having errors. Note that in order to print a document, you must first save it on the disk.

A fourth mode allows document editing. To correct errors, you identify the line by number and retype the line. To end the edit session type % followed by **RETURN**. This will bring you back to the beginning of the program, but you will still be working on the same document. After ending an edit session, your customary next action should be to save the document. The fifth and final mode allows printing of a final draft of a document.

When the word processor is first run, you will see the following prompt:

```
WORD PROCESSING PROGRAM
CHOOSE ONE OF THE FOLLOWING MODES

INPUT TEXT(I)
DISPLAY DRAFT (DD)
PRINT FINAL DRAFT (PF)
SAVE FILE (S)
EDIT (E)
QUIT (Q)
```

In response, you type one of I, DD, PF, S, E, or Q, followed by RETURN. If you choose I, the screen will be cleared and you may begin typing your document. For the other modes, there are prompts to tell you what to do. Here is a listing of the program.

```
5 DIM A$(150)
10 PRINT "WORD PROCESSING PROGRAM"
20 PRINT "CHOOSE ONE OF THE FOLLOWING MODES"
30 PRINT, "INPUT TEXT(I)"
40 PRINT, "DISPAY DRAFT(DD)"
50 PRINT, "PRINT FINAL DRAFT(PF)"
60 PRINT, "SAVE FILE(S)"
```

```
70 PRINT, "EDIT(E)"
80 PRINT, "QUIT(Q)"
90 INPUT X$
100 IF X$="I" THEN 1000
110 IF X$="DD" THEN 2000
120 IF X$="PF" THEN 3000
130 IF X$="S" THEN 4000
140 IF X$="E" THEN 5000
150 IF X$="Q" THEN 6000
160 GOTO 90: REM IF X$ DOES NOT MATCH ANY OF THE PROMPTS
1000 L=1
1010 INPUT A$(L)
1020 IF A$(L)="#" THEN 10
1030 L=L+1
1035 IF L<=150 THEN 1010
1040 IF L>150 THEN PRINT "DOCUMENT TOO LARGE"
1050 GOTO 10
2000 INPUT "DOCUMENT NAME";Y$
2010 PRINT CHR$(4); "OPEN"; Y$
2015 PRINT CHR$(4); "READ"; Y$
2020 INPUT L
2030 FOR K=1 TO L
2040 INPUT A$(K)
2050 PRINT K;">";A$(K)
2060 NEXT K
2070 PRINT CHR$(4); "CLOSE"; Y$
2090 GOTO 10
3000 INPUT "DOCUMENT NAME";Y$
3010 PRINT CHR$(4); "OPEN"; Y$
3015 PRINT CHR$(4); "READ"; Y$
3020 INPUT L
3030 FOR K=1 TO L
3040 INPUT A$(K)
3050 NEXT K
3060 PRINT CHR$(4) "CLOSE"; Y$
3070 PR#1
3080 FOR K=1 TO L
3090 PRINT A$(K)
3100 NEXT K
3110 PR#0
3120 GOTO 10
4000 INPUT "DOCUMENT NAME";Y$
4010 PRINT CHR$(4); "OPEN"; Y$
4015 PRINT CHR$(4); "WRITE"; Y$
4020 PRINT L
4030 FOR K=1 TO L
4040 PRINT A$(K)
4050 NEXT K
4060 PRINT CHR$(4); "CLOSE"; Y$
4070 GOTO 10
5000 INPUT "DOCUMENT NAME"; Y$
```

```
5010 PRINT CHR$(4); "OPEN";Y$
5015 PRINT CHR$(4); "READ"; Y$
5020 INPUT L
5030 FOR K=1 TO L
5040 INPUT A$(K)
5050 NEXT K
5055 PRINT CHR$(4); "CLOSE"; Y$
5060 INPUT "NUMBER OF LINE TO EDIT";Z
5070 HOME
5080 PRINT A$(Z)
5090 INPUT "TYPE CORRECTED LINE";A$(Z)
5100 IF A$(Z)<>"%" THEN 5060
5110 GOTO 10
6000 END
```

You should use this program to type a few letters. You will find it a big improvement over a conventional typewriter. Moreover, this will probably whet your appetite for the more advanced word processing features we described in the preceding section.

EXERCISES

1. Modify the word processor to allow input of line width. (You will not be able to display lines longer than 40 characters on a single line. However, string variables may contain up to 255 characters.)
2. Modify the word processor so that you may extend a line. This modification should let your corrected line spill over into the next line of text. The program should then correct all of the subsequent lines to reflect the addition.
3. Modify the word processor to allow deletions from lines. Subsequent lines should be modified to reflect the deletion.

8

Computer Games

In the last few years computer games have captured the imaginations of millions of people. In this chapter, we will build several computer games which utilize both the random number generator and the graphics capabilities of the Apple II Computer. In many games, we need a clock to time moves. The Apple II Computer does not come with a built-in clock, but there are several available from outside vendors. We will start by learning to use one of these optional clocks to tell time with the computer.

8.1 TELLING TIME WITH YOUR COMPUTER

You may equip your Apple II Computer with a clock (a **realtime-clock** in computer jargon) which allows you to let your programs take into account the time of day (in hours, minutes, and seconds) and the date (day of the week, date, and month). You can use this feature for many purposes, such as keeping a computer-generated personal calendar (see Example 1) or timing a segment of a program (see Example 2). There are several clocks available in the form of circuit boards. You insert the circuit board into one of the unused slots of your Apple II. In this chapter, we will assume for clarity that you are using the **THUNDERCLOCK PLUS*** and that this circuit board has been inserted into slot 3.

*THUNDERCLOCK PLUS is a registered trademark of Thunderware, Inc.

READING THE REAL-TIME CLOCK

The THUNDERCLOCK PLUS real-time clock keeps track of six pieces of information in the following order:

Month: MO (1–12)

Day of week: DW (0–6), Sunday = 0

Date: DT (1–31)

Hours: HR (0–23)

Minutes: MN (0–59)

Seconds: SEC (0–59)

The date and time are displayed in the following format:

02 03 15 14 38 27

The above display corresponds to Wednesday, February 15, at 27 seconds after 2:38 PM. Note that the hours are counted using a 24-hour clock, with 0 hours corresponding to midnight. Hours 0-11 correspond to AM and hours 12-23 correspond to PM.

To obtain the reading of the clock in numerical format, use the following sequence of BASIC commands:

```
10 PR#3: IN#3
20 INPUT "#";MO,DW,DT,HR,MN,SEC
```

The variables MO, DW, DT, HR, MN, and SEC will then contain integers equal to the current reading of the various entries of the clock. For example, to display the current time on the screen, use the sequence of commands

```
10 PR#3: IN#3
20 INPUT "#";MO,DW,DT,HR,MN,SEC
30 PR#0: IN#0
40 PRINT MO,DW,DT,HR,MN,SEC
```

TEST YOUR UNDERSTANDING 1

Display the current time and date.

SETTING THE CLOCK

You have an opportunity to set the clock when installing the clock. A battery keeps the clock running when the computer is turned off. We

refer you to the THUNDERCLOCK PLUS manual for the instructions on setting the clock.

TEST YOUR UNDERSTANDING 2

Write a program which continually displays the correct time on the screen.

Example 1. Use the real-time clock to build a computerized appointment calendar.

Solution. Enter appointments in **DATA** statements, with one **DATA** statement for each appointment. Let the **DATA** statements be arranged in the following format:

month day hour minute appointment

10 DATA 10,23,11,30,DENTIST

The appointments will be numbered by a variable *J*. For appointment *J*, we will store the five pieces of data in the variables *A(J)*, *B(J)*, *C(J)*, *D(J)*, *E\$(J)*, respectively. Let us allow for 300 appointments. We dimension each of the variables for an array of size 301 (we won't use *J* = 0).

1. Read the various appointments in the **DATA** statements. It will stop reading when there is no more data to read.
2. Ask for the current date and time data.
3. Determine today's appointments.

Here is the program:

```

10 DIM A(300),B(300),C(300),D(300),E$(300)
15 ONERR GOTO 100 : REM IF OUT OF DATA GO TO 100
20 REM 30-60 READ THE DATA STATEMENTS IN 1000-1300
30 J=1
40 READ A(J),B(J),C(J),D(J),E$(J)
50 J=J+1 : REM FINAL VALUE OF J= # APPOINTMENTS+1
60 GOTO 40
100 RESUME : REM END ERROR CONDITION AND GO TO 105
105 REM 110-160 ASK FOR CURRENT TIME AND DATE
110 PR#3:IN#3
115 INPUT "#";MO,DW,DT,HR,MN,SEC
120 PR#0:IN#0
200 PRINT "REQUEST SCHEDULE FOR MONTH,DAY"
210 PRINT "FOR TODAY, ENTER 0,0"
220 PRINT "FOR TOMORROW ENTER -1,-1"
230 INPUT X,Y: REM X=MONTH, Y=DAY
240 IF X>0 THEN 300
245 REM IF TODAY OR TOMORROW, FIND DATE AND DAY

```

```

250 X=MO-X: Y=DT-Y
300 FOR K=1 TO J-1
310 IF A(K)=X THEN 400: REM IS APPT. FOR RIGHT MONTH?
320 GOTO 510: REM IF NOT GO TO NEXT APPT.
400 IF B(J)=Y THEN 500: REM IS APPT. FOR RIGHT DAY?
410 GOTO 510: REM IF NOT GO TO NEXT APPT.
500 PRINT A(K),B(K),C(K),D(K),E$(K)
510 NEXT K
10000 DATA
10001 DATA
10002 DATA
.
.
.
10299 DATA
11000 END

```

To use the above program, type appointments into the data lines as you make them. You can store the current list of appointments along with the program. As new appointments are made, you can recall the program and add corresponding data lines. As appointments become obsolete, they may be removed from the list using the **DEL** command.

A few comments are in order concerning the **ONERR GOTO** and **RESUME** statements. At any given moment, you have no way of knowing how many appointments are in the calendar. Therefore, you have no way of knowing in advance how many of the data lines in 10000-10299 you must read. If you attempt to read past the last data line, you will get an **OUT OF DATA** error. The **ONERR GOTO** statement allows you to handle this error, when it occurs, without terminating program execution. In the above program, we have instructed the computer to respond to an error (namely the **OUT OF DATA** error we expect) by going to line 110. In this line, we tell the computer to **RESUME**; that is, to continue execution of the program with the next line after 105, just as if the error never occurred. This procedure allows us to read data until there is no more.

The above program assumes that you store the appointments in **DATA** statements. Of course, it is perfectly possible to store the appointments in a data file on either cassette or diskette. We leave it as an exercise to modify the above program to allow for use of such files.

Example 2. In Example 6 of Chapter 2, Section 6, (page 55) we developed a program to test mastery in the addition of two-digit numbers. Redesign this program to allow 15 seconds to answer the question.

Solution. Let us use the real-time clock. After a particular problem has been given, we will record the number of seconds as

$$3600*HR+60*MN+SEC$$

and perform a loop which continually compares this number with the current number of seconds. The loop will end when the two numbers differ by 15 (15 seconds has elapsed). Then the program will print out "TIME'S UP. WHAT IS YOUR ANSWER?" Here is the program. Lines 50 and 60 contain the loop.

```

10 FOR J=1 TO 10 : REM LOOP TO GIVE 10 PROBLEMS
20 INPUT "TYPE TWO 2-DIGIT NUMBERS"; A,B
30 PRINT "WHAT IS THEIR SUM?"
40 PR#3:IN#3
45 INPUT "#";MO,DW,DT,HR,MN,SEC
47 A=3600*HR+60*MN+SEC
49 INPUT"#";MO,DW,DT,HR,MN,SEC
50 B=3600*HR+60*MN+SEC
55 IF B-A=15 THEN GOTO 100: REM COUNT 15 SECONDS
60 GOTO 49
100 PR#0:IN#0
110 INPUT "TIME'S UP! WHAT IS YOUR ANSWER ";C
120 IF A+B=C THEN 200
130 PRINT "SORRY. THE CORRECT ANSWER IS ",A+B
140 GO TO 500 : REM GO TO THE NEXT PROBLEM
200 PRINT "YOUR ANSWER IS CORRECT! CONGRATULATIONS"
210 LET R=R+1 : REM INCREASE SCORE BY 1
500 NEXT J: REM GO TO NEXT PROBLEM
600 PRINT "YOUR SCORE IS ",R," CORRECT OUT OF 10"
700 PRINT "TO TRY AGAIN, TYPE RUN"
800 END

```

TEST YOUR UNDERSTANDING 3

Modify the above program so that it allows you to take as much time as you like to solve a problem, but keeps track of elapsed time (in seconds) and prints out the number of seconds used.

EXERCISES

1. Set the clock with today's date and the current time.
2. Print out the current time on the screen.
3. Write a program which prints out the date and time at one second intervals.
4. Write a program which prints out the date and time at one minute intervals.
5. Set up the appointment program of Example 1 (page 159) and use it to enter a week's worth of appointments.
6. Modify the appointment program of Example 1 so that it prints out the appointments on a per hour basis.

7. Modify the addition tester program so that it allows a choice of four levels of speed: Easy (2 minutes), Moderate (30 seconds), Hard (15 seconds), and Whiz Kid (8 seconds).
8. Write a program which can be run for an entire day and at the start of each hour prints out the appointments for that hour. (Such a program would be useful in a doctor's or dentist's office.)

8.2 BLIND TARGET SHOOT

The object of this game is shoot down a target on the screen by moving your cursor to hit the target. The catch is that you only have a two-second look at your target! The program begins by asking if you are ready. If so, you type READY. The computer then randomly chooses a spot to place the target. It lights up the spot for two seconds. The cursor is then moved to the upper left position of the screen (the so-called "home" position). You must then move the cursor to the target, based on your brief glimpse of it. You have 10 seconds to hit the target. (See Figure 8-1.)

Your score is based on your distance from the target, as measured in terms of the moves it would take to get to the target from your final

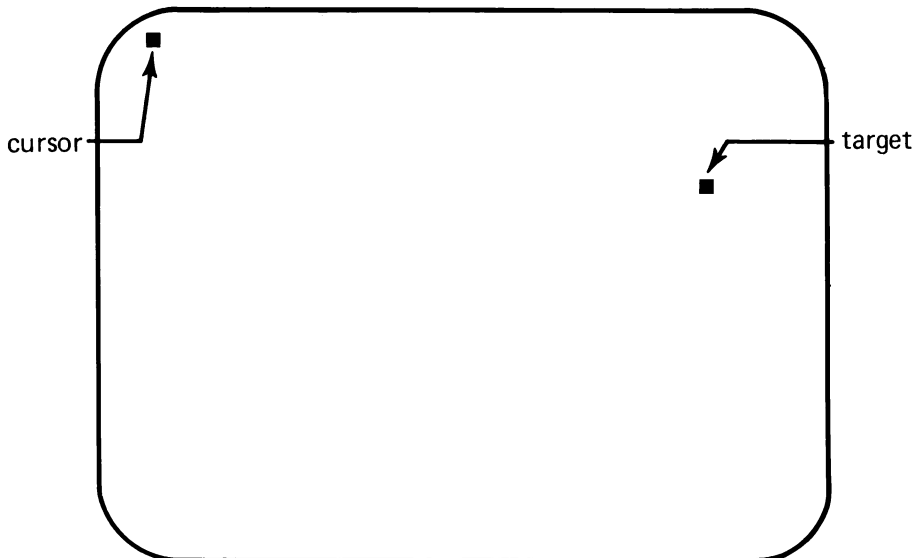


Figure 8-1. Blind target shoot.

position. Here is the list of possible scores:

Distance From Target	Score
0	100
1 or 2	90
3 to 5	70
6 to 10	50
11 to 15	30
16 to 20	10
over 20	0

To input cursor moves while the program is running, we will use the optional game paddles which may be connected to the Apple II Computer. By rotating the paddles, you input a number to the computer. This number is between 1 and 255. We will set up the following correspondence between paddle input and cursor moves:

0–63 = move one unit to the left

64–127 = move one unit to the right

128–191 = move one unit up

192–255 = one unit down

The actual correspondence between the physical rotation of the paddle and the cursor moves must be determined by practice.

The computer reads the game paddle using the instruction

10 PDL(0)

The number 0 is the number of the paddle. There may be up to four paddles connected (with numbers 0-3).

There is one further complication. In either of the Apple II's graphics modes, the cursor sits in the text window (the bottom four rows of the screen) and cannot move into the graphics area. Therefore, to use the cursor in a game, it is necessary to create a simulated cursor using a block lit up by the **PLOT** command. To move the simulated cursor, we first turn the cursor off in one location and then turn it on in another.

Here is a sample session with the game. The underlined lines are those you type.

RUN

**BLIND TARGET SHOOT
TO BEGIN GAME, TYPE "READY"
READY**

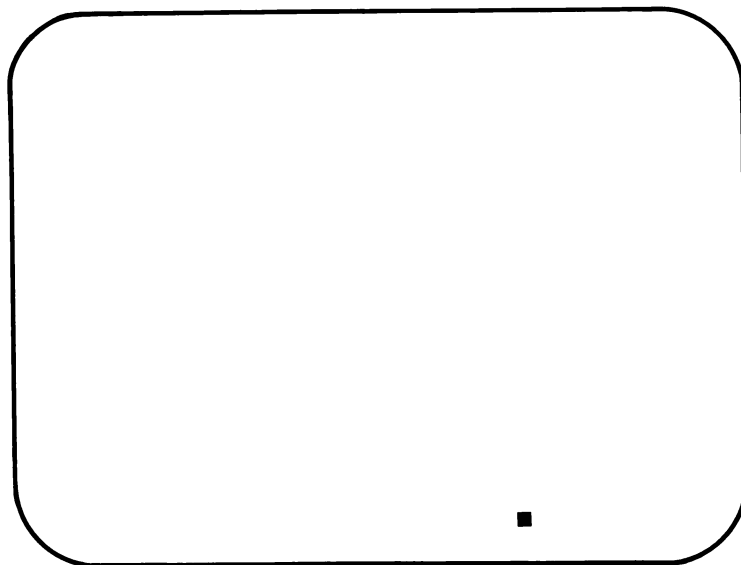


Figure 8-2.

The screen clears. The target is displayed. See Figure 8-2.

The screen is cleared and the cursor is moved to the home position. See Figure 8-3A. The cursor is then moved to the remembered position of the target. See Figure 8-3B. Time runs out. See Figure 8-3C.

The score is calculated. See Figure 8-4.

Here is a listing of our program.

```

10 PRINT "BLIND TARGET SHOOT"
20 PRINT "TO BEGIN GAME, TYPE  READY"
30 INPUT A$
40 IF A$="READY" THEN 90
90 GOTO 10: GR
100 GOSUB 2000
110 S=3600*HR+60*MN+SEC

```

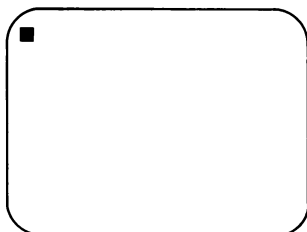


Figure 8-3A.

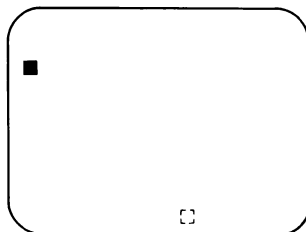


Figure 8-3B.

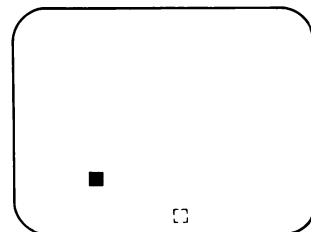


Figure 8-3C

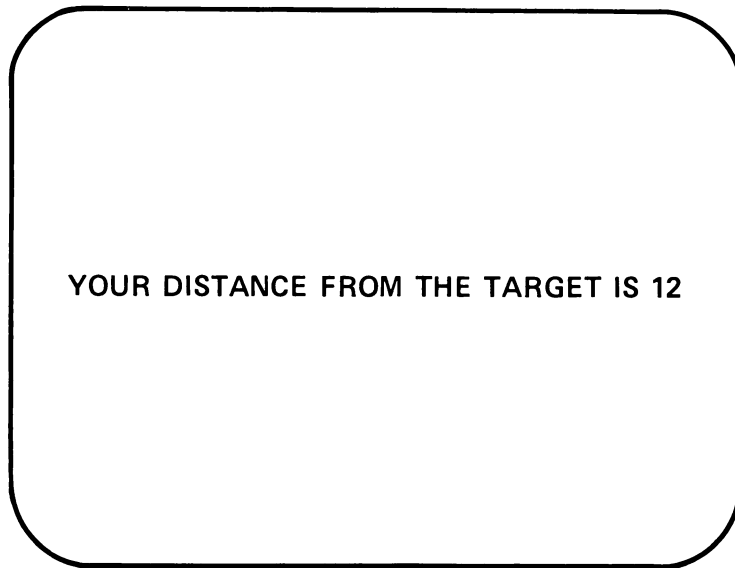


Figure 8-4

```

120 Y(0)=INT(40*RND(S)) :REM CHOOSE RANDOM COLUMN
130 X(0)=INT(40*RND(S)) :REM CHOOSE RANDOM ROW
135 COLOR=1
140 PLOT X(0),Y(0):REM DISPLAY TARGET
145 GOSUB 2000
147 T=3600*HR+60*MN+SEC
150 IF T-S=2 THEN 200
155 GOTO 160
160 GOTO 145 : REM WAIT 2 SECONDS
200 COLOR 0 : PLOT X(0),Y(0):REM TURN OFF TARGET
210 COLOR 1
220 PLOT 1,1 :REM TURN CURSOR BACK ON
300 GOSUB 2000
305 S=3600*HR+60*MN+SEC
310 X=1:Y=1: REM (X,Y) ARE COORDINATES OF CURSOR
400 A=PDL(0): REM READ GAME PADDLE
500 IF A>= 128 AND A <= 191 THEN 510:REM CURSOR UP
505 GOTO 600
510 IF Y>0 THEN 520 : GOTO 1000
515 GOTO 1000
520 COLOR 0: PLOT X,Y: COLOR 1
530 Y=Y-1: PLOT X,Y
540 GOTO 1000
600 IF A >= 192 AND A <= 255 THEN 610:REM CURSOR DOWN
605 GOTO 700
610 IF Y<39 THEN 620
615 GOTO 1000
620 COLOR 0 :PLOT X,Y:COLOR 1
630 Y=Y+1: PLOT X,Y

```

```

640 GOTO 1000
700 IF A>=64 AND A<=127 THEN 710: REM CURSOR RIGHT
705 GOTO 800
710 IF X<39 THEN 720
715 GOTO 1000
720 COLOR 0: PLOT X,Y:COLOR 1
730 X=X+1:PLOT X,Y
740 GOTO 1000
800 IF A<= 63 THEN 810:REM CURSOR LEFT
805 GOTO 1000
810 IF X>0 THEN 820
815 GOTO 1000
820 COLOR 0:PLOT X,Y:COLOR 1
830 X=X-1:PLOT X,Y
840 GOTO 1000
1000 REM IS TIME UP?
1010 GOSUB 2000
1015 T=3600*HR+60*MN+SEC
1020 IF T-S=10 THEN 1100
1025 GOTO 400
1100 T=ABS(X-X(0))+ABS(Y-Y(0)):REM T=DIST. TO TARGET
1105 HOME
1110 PRINT "YOUR DISTANCE FROM THE TARGET IS ",T
1120 IF T=0 THEN PRINT "CONGRATULATIONS!"
1130 IF T=0 THEN PRINT "YOU HIT THE TARGET."
1140 SC=100
1150 IF T>0 THEN SC=SC-10
1160 IF T>2 THEN SC=SC-20
1170 IF T>5 THEN SC=SC-20
1180 IF T>10 THEN SC=SC-20
1190 IF T>15 THEN SC=SC-20
1200 IF T>20 THEN SC=SC-10
1300 PRINT "YOUR SCORE IS", SC
1410 IF B$="Y" THEN CLEAR: GOTO 90
1410 IF B$="Y" THEN 90
1500 END
2000 PR#3:IN#3:REM READ CLOCK
2010 INPUT "#"; MO,DW,DT,HR,MN,SEC
2020 PR# 0:IN#0
2030 RETURN

```

EXERCISES

1. Experiment with the above program by making the time of target viewing shorter or longer than 1 second.
2. Experiment with the above program by making the time for target location shorter or longer than 10 seconds.

3. Modify the program to keep a running total score for a sequence of 10 games.
4. Modify the program to allow two players, keeping a running total score for a sequence of 10 games. At the end of ten games, the computer should announce the total scores and declare the winner.

8.3 TIC TAC TOE

In this section, we present a program for the traditional game of tic tac toe. We won't attempt to let the computer execute a strategy. Rather, we will let it be fairly stupid and choose its moves randomly. We will also use the random number generator to "flip" for the first move. Throughout the program, you will be "O" and the computer will be "X." Here is a sample game.

TEST YOUR UNDERSTANDING 1 (answer on page 173)

How can the computer toss to see who goes first?

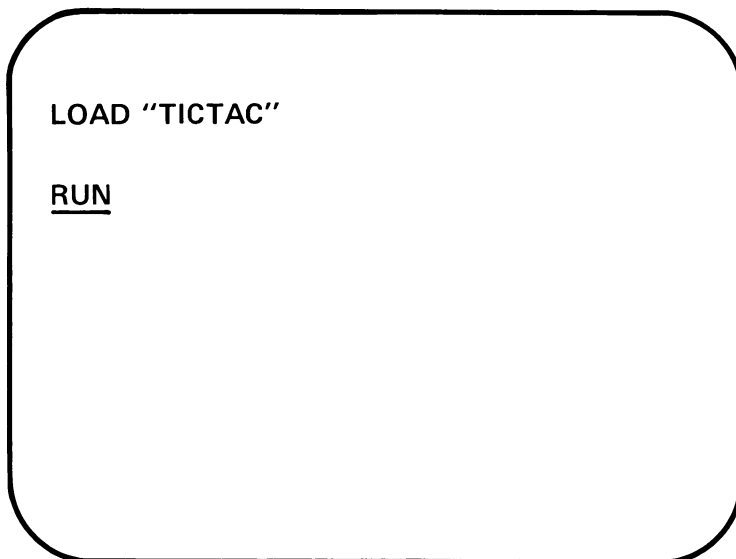


Figure 8-5

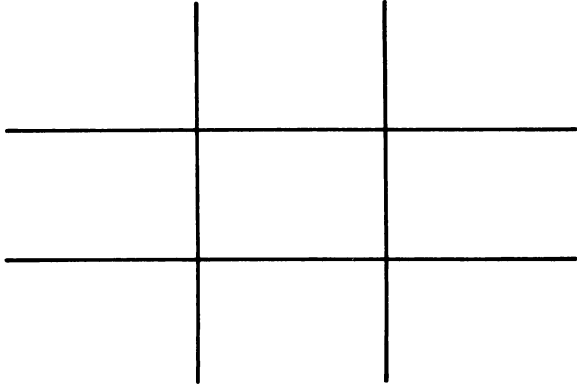
TIC TAC TOE
YOU WILL BE O;THE COMPUTER WILL BE X
THE POSITIONS OF THE BOARD ARE NUMBERED
AS FOLLOWS:

1	2	3
4	5	6
7	8	9

THE COMPUTER WILL TOSS FOR FIRST.
YOU GO FIRST.
WHEN READY TO BEGIN TYPE 'R'
R

Figure 8-6.

The computer now draws a TIC TAC TOE board. See Figure 8-7.
The computer now displays your move and makes a move of its own.
See Figure 8-8.
The computer will now make its move and so on until someone wins or
a tie game results.



TYPE YOUR MOVE(1-9)
?5

Figure 8-7.

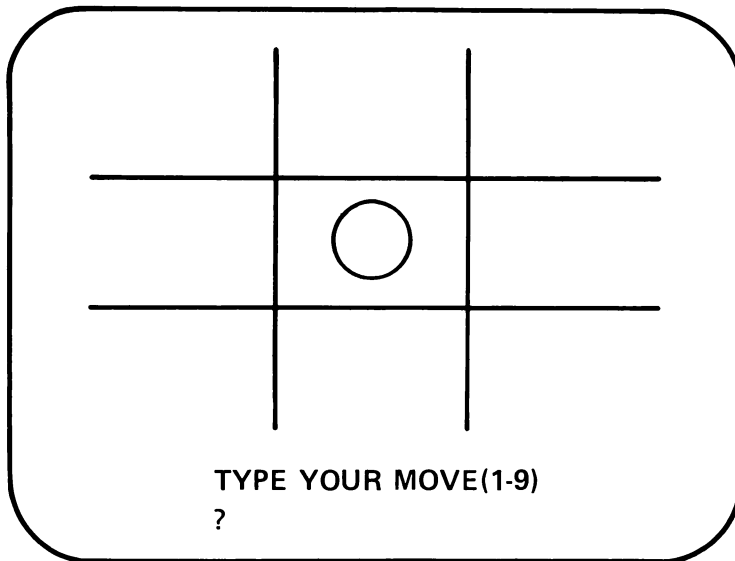


Figure 8-8.

The program below makes use of much of what we have learned. First of all, we have used the graphics mode to draw the TIC TAC TOE board, beginning in line 2000. We have structured the program so that it consists of a series of subroutines. Lines 2000–2999 contain the instructions to draw the board and to display the current status of the game. Lines 3000–3999 contain a subroutine which inputs your move. Lines 4000–4999 contain a subroutine which lets the computer decide its move. Lines 5000–5999 process the current move and decide if the game is over. The logic in lines 6000–8000 allows the computer to determine if there is a winning move or if it should block its opponent.

Here are the variables used in the program.

Z = 0 if it's your move and Z = 1 if it is the computer's

A\$(J) (J=1,2,...,9) contain either O, X, or the empty string, indicating the current status of position J

S = the position of the current move

M = the number of moves played (including the current one)

We used a video display worksheet to lay out the board, and to determine the coordinates for the lines and the X's and O's.

Here is a listing of our program.

```
5 INPUT "RANDOM NUMBER SEED";Q
10 DIM A$(9)
20 DIM B$(3,3)
```

```
30 HOME
40 PRINT "TIC TAC TOE"
50 PRINT "YOU WILL BE O; THE COMPUTER WILL BE X"
60 PRINT "THE POSITIONS ON THE BOARD ARE NUMBERED"
70 PRINT "AS FOLLOWS"
80 PRINT "1,2,3"
90 PRINT "4,5,6"
100 PRINT "7,8,9"
110 PRINT "THE COMPUTER WILL TOSS FOR FIRST"
120 IF RND(Q)>.5 THEN 130
125 GOTO 160
130 PRINT "YOU GO FIRST"
140 LET Z=0
150 GOTO 180
160 PRINT "I'LL GO FIRST"
170 LET Z=1
180 PRINT "WHEN READY TO BEGIN TYPE 'R' "
190 INPUT C$
200 IF C$="R" THEN 210
205 GOTO 180
210 GOSUB 2000
300 FOR M=1 TO 9: 'M=MOVE #
310 IF Z=0 THEN GOSUB 3000
320 IF Z=1 THEN GOSUB 6000
330 Z=1-Z: REM Z=1 MEANS NEXT MOVE IS COMPUTER
350 IF W=1 THEN 400
355 NEXT M
360 PRINT "THE GAME IS TIED"
400 END

2000 REM DRAW TIC TAC TOE BOARD
2010 GR:COLOR=1
2020 FOR J=1 TO 2
2030 FOR K=0 TO 39
2040 PLOT K, 13*J
2050 NEXT K
2060 NEXT J
2070 FOR J=1 TO 2
2080 FOR K=0 TO 39
2090 PLOT 13*J,K
2100 NEXT K
2110 NEXT J
2115 REM DISPLAY CURRENT GAME STATUS
2120 A=7:B=7
2121 IF A$(1)="X" THEN GOSUB 2200
2122 IF A$(1)="O" THEN GOSUB 2300
2123 A=20:B=7
2124 IF A$(2)="X" THEN GOSUB 2200
2125 IF A$(2)="O" THEN GOSUB 2300
2126 A=33:B=7
2127 IF A$(3)="X" THEN GOSUB 2200
2128 IF A$(3)="O" THEN GOSUB 2300
```

```
2129 A=7: B=20
2130 IF A$(4)="X" THEN GOSUB 2200
2131 IF A$(4)="O" THEN GOSUB 2300
2132 A=20:B=20
2133 IF A$(5)="X" THEN GOSUB 2200
2134 IF A$(5)="O" THEN GOSUB 2300
2135 A=33:B=20
2136 IF A$(6)="X" THEN GOSUB 2200
2137 IF A$(6)="O" THEN GOSUB 2300
2138 A=7:B=33
2139 IF A$(7)="X" THEN GOSUB 2200
2140 IF A$(7)="O" THEN GOSUB 2300
2141 A=20:B=33
2142 IF A$(8)="X" THEN GOSUB 2200
2143 IF A$(8)="O" THEN GOSUB 2300
2144 A=27:B=33
2145 IF A$(9)="X" THEN GOSUB 2200
2146 IF A$(9)="O" THEN GOSUB 2300
2150 RETURN
2200 REM DRAW X
2205 COLOR=1
2210 PLOT A-2,B
2211 PLOT A-2,B+1
2212 PLOT A-2,B-1
2213 PLOT A+2,B
2214 PLOT A+2,B+1
2215 PLOT A+2,B-1
2216 PLOT A,B-2
2217 PLOT A-1,B-2
2218 PLOT A+1,B-2
2219 PLOT A,B+2
2220 PLOT A-1,B+2
2221 PLOT A+1, B+2
2230 RETURN
2300 REM DRAW O
2305 COLOR=1
2310 PLOT A,B
2311 PLOT A-1,B-1
2312 PLOT A-1,B+1
2313 PLOT A-2,B-2
2314 PLOT A-2,B+2
2315 PLOT A+1,B+1
2316 PLOT A+1,B-1
2317 PLOT A+2,B+2
2318 PLOT A+2,B-2
2320 RETURN
3000 PRINT "TYPE YOUR MOVE (1-9)"
3010 INPUT S
3020 A$(S)="O"
3030 GOSUB 2000
3040 GOSUB 5000
```

```
3050 RETURN
4000 LET S=INT(9*RND(Q)+1)
4010 IF A$(S)="" THEN 4060
4015 GOTO 4000
4060 A$(S)="X"
4070 GOSUB 2000
4080 GOSUB 5000
4090 RETURN
5000 IF Z=0 THEN C$="O"
5005 IF Z=C THEN C$="X"
5010 IF A$(1)=A$(2) THEN 5012
5011 GOTO 5020
5012 IF A$(1)=A$(3) THEN 5014
5013 GOTO 5020
5014 IF A$(3)=C$ THEN 5900
5020 IF A$(1)=A$(4) THEN 5022
5021 GOTO 5030
5022 IF A$(4)=A$(7) THEN 5024
5023 GOTO 5030
5024 IF A$(7)=C$ THEN 5900
5030 IF A$(1)=A$(5) THEN 5032
5031 GOTO 5040
5032 IF A$(5)=A$(9) THEN 5034
5033 GOTO 5040
5034 IF A$(9)=C$ THEN 5900
5040 IF A$(2)=A$(5) THEN 5042
5041 GOTO 5050
5042 IF A$(5)=A$(8) THEN 5044
5043 GOTO 5050
5044 IF A$(8)=C$ THEN 5900
5050 IF A$(3)=A$(6) THEN 5052
5051 GOTO 5060
5052 IF A$(6)=A$(9) THEN 5054
5053 GOTO 5060
5054 IF A$(9)=C$ THEN 5900
5060 IF A$(4)=A$(5) THEN 5062
5061 GOTO 5070
5062 IF A$(5)=A$(6) THEN 5064
5063 GOTO 5070
5064 IF A$(9)=C$ THEN 5900
5070 IF A$(7)=A$(8) THEN 5072
5071 GOTO 5080
5072 IF A$(8)=A$(9) THEN 5074
5073 GOTO 5080
5074 IF A$(9)=C$ THEN 5900
5080 IF A$(3)=A$(5) THEN 5082
5081 GOTO 5990
5082 IF A$(5)=A$(7) THEN 5084
5083 GOTO 5990
5084 IF A$(7)=C$ THEN 5900
5085 GOTO 5990
5900 PRINT C$, "WINS THIS ROUND" :W=1
```

```

5990 RETURN
6000 REM LINES 6100-6600 LOOK FOR A WINNING MOVE
6010 COUNT=0
6100 REM
6110 FOR I=1 TO 9
6120 IF A$(I)="X" THEN B(I)=1
6140 IF A$(I)="" THEN B(I)=0
6150 IF A$(I)="O" THEN B(I)=-1
6160 NEXT I
6200 READ I,J,K
6210 COUNT=COUNT+1
6220 IF COUNT=8 THEN 6500
6300 S=B(I)+B(J)+B(K)
6310 IF S=2 THEN 6400
6320 GOTO 6200
6400 IF B(J)=0 THEN A$(J)="X"
6405 GOTO 7000 .
6410 IF B(K)=0 THEN A$(K)="X"
6415 GOTO 7000
6420 IF B(I)=0 THEN A$(I)="X"
6425 GOTO 7000
6430 GOTO 6210
6500 RESTORE
6505 COUNT=0
6510 READ I,J,K
6520 COUNT=COUNT+1
6530 IF COUNT=8 THEN 7100
6540 IF S=-2 THEN 6600: GOTO 6510
6550 GOTO 6510
6600 IF B(J)=0 THEN A$(J)="X"
6610 IF B(K)=0 THEN A$(K)="X"
6620 IF B(I)=0 THEN A$(I)="X"
6630 GOTO 7000
7000 RESTORE:GOTO 4080
7100 RESTORE:GOTO 4000
8000 DATA 1,2,3,4,5,6,7,8,9,1,4,7,2
8010 DATA 5,8,3,6,9,1,5,9,3,5,7

```

EXERCISES

1. Modify the above program so that you and the computer may play a series of ten games. The computer should decide the champion of the series.
2. Modify the above program to play 4 x 4 Tic Tac Toe.

ANSWER TO TEST YOUR UNDERSTANDING 1

See lines 120-170 of the program on page 170.



9

Programming for Scientists

In this chapter we will discuss some aspects of Apple II Computer programming of interest to scientists, engineers, and mathematicians. In particular, we will introduce the library of mathematical functions which can be used in Applesoft BASIC.

9.1 INTEGER AND REAL CONSTANTS

Up to this point, we have used the computer to perform arithmetic without giving much thought to the level of accuracy of the numbers involved. However, when doing scientific programming, it is absolutely essential to know the number of decimal places of accuracy of the computations. Let's begin this chapter by discussing the form in which Applesoft BASIC stores and utilizes numbers.

Applesoft BASIC recognizes two different types of numeric constants: integer and real.

An **integer** numeric constant is an ordinary integer (positive or negative) in the range -32768 to $+32767$. (32768 is 2 raised to the 15th power. This number is significant for the internal workings of the Apple II Computer.) Here are some examples of integer numeric constants:

7, 58, 3712, -15 , -598

Integer numeric constants may be stored very efficiently in RAM. Therefore, in order to realize these efficiencies, the Apple II Computer recognizes integer numeric constants and handles them in a special way.

by the approximations of the computer are called **round-off errors**. Consider the problem of calculating

$$1/3 + 1/3 + 1/3$$

As we have seen above, $1/3$ is stored as the real constant .33333333. The computer will form the sum as

$$.33333333 + .33333333 + .33333333 = .99999999$$

The sum has a round-off error of .000000001.

The Apple II Computer displays up to nine digits for a real constant. However, due to round-off error, the answer to any single arithmetic operation is guaranteed accurate to only eight places.

EXERCISES (answers on page 240)

For each of the constants below, determine the number stored by the computer.

- | | |
|---------------------------|------------------------|
| 1. 3 | 2. 2.37 |
| 3. 5.78E5 | 4. 2 |
| 5. 3.25 | 6. -4.1578912345 |
| 7. -4.1 | 8. 3500.6847586958658 |
| 9. 2.176E2 | 10. -5.94E12 |
| 11. 3.5869504003837265374 | 12. -234542383746.21 |
| 13. -2.367E20 | 14. 457000000000000000 |

For each of the arithmetic problems below, determine the number as stored by the computer.

- | | |
|--------------------|-----------------------|
| 15. $1 + 45$ | 16. $2/4$ |
| 17. $3/5$ | 18. $3/5 + 1$ |
| 19. $2/3$ | 20. $2/3 + .53$ |
| 20. $.5E4 - .37E2$ | 21. $1.75E3 - 1.0E-5$ |
22. Calculate $1/3 + 1/3 + 1/3 + \dots + 1/3$ (10 $1/3$'s) using real constants. What answer is displayed? Is this answer accurate to eight digits? If not, explain why.

9.2 VARIABLE TYPES

In the previous section we introduced the types of numerical constants: integer and real. There is a parallel set of types for variables.

A variable of **integer type** takes on values which are integer type constants. An integer type variable is indicated by the symbol % after the variable name. For example, here are some variables of integer type:

A%, BB%, A1%

In setting the value of an integer type variable, the computer will truncate any fractional parts to obtain an integer. For example, the instruction

```
10 LET A% = 2.54
```

will set the value of A equal to the integer constant 2. Integer type variables are useful when keeping track of integer quantities, such as line numbers in a program.

A variable of **real type** is one whose value is a real constant. A variable of real type is any variable not of integer or string type. Here are some examples of real variables:

K, W7, ZX

In setting the value of a real variable, all digits beyond the ninth are truncated. For example, the instruction

```
20 LET A = 1.23456789123
```

will set A equal to 1.23456789.

If a variable is used without a type designator, then the computer will then assume that it is a real variable. All of the variables we have used until now have been real variables. These are, by far, the most commonly used variables.

Note that the variables A%, A, and A\$ are three distinct variables. You could, if you wish, use all of them in a single program.

TEST YOUR UNDERSTANDING 1 (answers on page 179)

What values are assigned to each of these variables?

- a. A = 1
- b. C% = 5.22
- c. BB = 1387.5699

EXERCISES (answers on page 241)

Calculate the following quantities in real arithmetic.

1. $(5.87 + 3.85 - 12.07)/11.98$
2. $(15.1 + 11.9)^4/12.88$
3. $(32485 + 9826)/(321.5 - 87.6^2)$
4. Write a program to determine the largest integer less than or equal to X, where the value of X is supplied in an INPUT statement.

Determine the value assigned to the variable in each of the following exercises.

- | | |
|---------------------------------|-------------------------------|
| 5. A% = -5 | 6. A% = 4.8 |
| 7. A% = -11.2 | 8. A = 1.78 |
| 9. A = .001 | 10. A = 32.653426278374645237 |
| 11. A = 4.25234544321E21 | 12. A = -1.23456789E-32 |
| 13. A = 3.283646493029273646434 | 14. A = -5.74 |

<p>ANSWERS TO TEST YOUR UNDERSTANDING 1</p> <p>a. 1.00000000</p> <p>b. 5</p> <p>c. 1387.56999</p>
--

9.3 MATHEMATICAL FUNCTIONS IN APPLESOFT BASIC

In performing scientific computations, it is often necessary to use a wide variety of mathematical functions, including the natural logarithm, the exponential, and the trigonometric functions. The Apple II Computer has a wide range of these functions "built-in." In this section we will describe these functions and their use.

All mathematical functions in Applesoft BASIC work in a similar fashion. Each function is identified by a sequence of letters (**SIN** for sine, **LOG** for natural logarithm, and so forth). To evaluate a function at a

number X , we write X in parentheses after the function name. For example, the natural logarithm of X is written $\text{LOG}(X)$. The program will use the current value of the variable X and will evaluate the natural logarithm of that value. For example, if X is currently 2, then the computer will calculate $\text{LOG}(2)$.

Instead of the variable X , we may use any type of variable: integer or real. We may also use a numerical constant of any type. For example, $\text{SIN}(.435678889658595)$ asks for the sine of a real numerical constant. Note that with only a few exceptions (see below), all Applesoft BASIC functions return a real result, accurate to nine digits. For example, the above value of the sine function will be computed as

$\text{SIN}(.435678889658595) = .422025969$

Applesoft BASIC lets you evaluate a function at any expression. Consider the expression $X^2 + Y^2 - 3*X$. It is perfectly acceptable to call for calculations such as

$\text{SIN}(X^2 + Y^2 - 3*X)$

The computer will first evaluate the expression $X^2 + Y^2 - 3*X$ using the current values of the variables X and Y . For example, if $X = 1$ and $Y = 4$, then $X^2 + Y^2 - 3*X = 1^2 + 4^2 - 3*1 = 14$. So the above sine function will be evaluated as $\text{SIN}(14) = .990607356$.

TRIGONOMETRIC FUNCTIONS

The Apple II Computer has the following trigonometric functions available.

$\text{SIN}(X)$ = the sine of the angle X

$\text{COS}(X)$ = the cosine of the angle X

$\text{TAN}(X)$ = the tangent of the angle X

Here the angle X is expressed in terms of radian measure. In this measurement system, 360 degrees equals 2π radians. Or one degree equals .017453 radians and one radian equals 57.29578 degrees. If you want to calculate trigonometric functions with the angle X expressed in degrees, use these functions:

$\text{SIN}(.017453*X)$

$\text{COS}(.017453*X)$

$\text{TAN}(.017453*X)$

The three other trigonometric functions, $\text{SEC}(X)$, $\text{CSC}(X)$, and $\text{COT}(X)$, may be computed from the formulas

$$\text{SEC}(X) = 1/\text{COS}(X)$$

$$\text{CSC}(X) = 1/\text{SIN}(X)$$

$$\text{COT}(X) = \text{COS}(X)/\text{SIN}(X)$$

Here, as above, the angle X is in radians. To compute these trigonometric functions with the angle in degrees, replace X by

$$.017453 * X$$

Applesoft BASIC has only one of the inverse trigonometric functions, namely the arctangent, denoted $\text{ATN}(X)$. This function returns the angle whose tangent is X . The angle returned is expressed in radians. To compute the arctangent with the angle expressed in degrees, use the function

$$57.29578 * \text{ATN}(X)$$

TEST YOUR UNDERSTANDING 1 (answer on page 184)

Write a program which calculates $\sin 45^\circ$, $\cos 45^\circ$, and $\tan 45^\circ$.

LOGARITHMIC AND EXPONENTIAL FUNCTIONS

Applesoft BASIC allows you to compute e^x using the exponential function:

$$\text{EXP}(X)$$

Furthermore, you may compute the natural logarithm of X via the function

$$\text{LOG}(X)$$

You may calculate logarithms to base b using the formula:

$$\text{LOG}_b(X) = \text{LOG}(X)/\text{LOG}(b)$$

Example 1. Prepare a table of values of the natural logarithm function for values $X = .01, .02, .03, \dots, 100.00$. Output the table on the printer.

Solution. Here is the desired program. Note that we have prepared our table in two columns with a heading over each column.

```

5 PR#1
10 PRINT "X", "LOG(X)"
20 J=.01
30 PRINT J, LOG(J)
40 IF J=100.00 THEN 100
45 GOTO 50
50 J=J+.01
60 GOTO 30
70 PR#0
100 END

```

TEST YOUR UNDERSTANDING 2 (answer on page 184)

Write a program which evaluates the function $f(x) = \sin x / (\log x + e^x)$ for $x = .45$ and $x = .7$.

Example 2. Carbon dating is a technique for calculating the age of ancient artifacts by measuring the amount of radioactive carbon-14 remaining in the artifact, as compared with the amount present if the artifact were manufactured today. If r denotes the proportion of carbon-14 remaining, then the age A of the object is calculated from the formula:

$$A = -(1/.00012) * \text{LOG}(r)$$

Suppose that a papyrus scroll contains 47% of the carbon-14 of a piece of papyrus just manufactured. Calculate the age of the scroll.

Solution. Here $r = .47$. So we use the above formula:

```

10 LET R=.47
20 LET A=-(1/.00012)*LOG(R)
30 PRINT "THE AGE OF THE PAPYRUS IS", A, "YEARS"
40 END

```

POWERS

Applesoft BASIC has a square root function, denoted $\text{SQR}(X)$. As with all the functions considered so far, this function will accept as input an integer or a real constant and will output a real constant. For example, the instruction

```
10 LET Y=SQR(2)
```

will set Y equal to 1.41421356.

Actually, the exponentiation procedure which we learned in Chapter 2 will work equally well for fractional and decimal exponents, and

therefore provides an alternative method for extracting square roots. Here is how to use it. Taking the square root of a number corresponds to raising the number to the $1/2$ power. We may calculate the square root of X as

$$X^{(1/2)}$$

Note that the square root function $SQR(X)$ operates with greater speed and is therefore preferred. The alternate method is more flexible, however. For instance, we may extract the cube root of X as

$$X^{(1/3)}$$

or we may raise X to the 5.389 power as follows:

$$X^{5.389}$$

GREATEST INTEGER, ABSOLUTE VALUE, AND RELATED FUNCTIONS

Here are several extremely helpful functions. The greatest integer less than or equal to X is denoted $INT(X)$. For example, the largest integer less than or equal to 5.46789 is 5, so

$$INT(5.46789) = 5$$

Similarly, the largest integer less than or equal to -3.4 is -4 (on the number line, -4 is the first integer to the left of -3.4). Therefore:

$$INT(-3.4) = -4$$

Note that the INT function throws away the decimal part of a positive number, although this description does not apply to negative numbers.

The absolute value of X is denoted $ABS(X)$. Recall that the absolute value of X is X itself if X is positive, or 0 and equals $-X$ if X is negative. Thus:

$$ABS(9.23) = 9.23$$

$$ABS(0) = 0$$

$$ABS(-4.1) = 4.1$$

Just as the absolute value of X “removes the sign” of X , the function $SGN(X)$ throws away the number and leaves only the sign. For example:

$$SGN(3.4) = +1$$

$$SGN(-5.62) = -1$$

$$SGN(0) = 0.$$

EXERCISES (answers on page 242)

Calculate the following quantities:

1. $e^{1.54}$
2. $e^{-2.376}$
3. $\log 58$
4. $\log .0000975$
5. $\sin 3.7$
6. $\cos 45^\circ$
7. $\arctan 1$
8. $\tan .682$
9. $\arctan 2$ (expressed in degrees)
10. $\log_{1018.9}$
11. Make a table of values of the exponential function $\exp(x)$ for $x = -5.0, -4.9, \dots, 0, .1, \dots, 5.0$.
12. Evaluate the function

$$3x^{(1/4)}\log(5x) + \exp(-1.8x)\tan x$$

for $x = 1.7, 3.1, 5.9, 7.8, 8.4, \text{ and } 10.1$.

13. Write an Applesoft BASIC program to graph the function $y = \sin x$ for x from 0 to 6.28. Use an interval of .05 on the x axis.
14. Write an Applesoft BASIC program to graph the function $y = \text{ABS}(x)$.
15. Write a program to calculate the fractional part of x . (The fractional part of x is the portion of x which lies to the right of the decimal point.)

ANSWERS TO TEST YOUR UNDERSTANDING 1 and 2

```

1: 10 LET A = .017453
   20 PRINT SIN(45*A), COS(45*A), TAN(45*A)
   30 END

2: 10 DATA .45, .7
   20 FOR J = 1 TO 2
   30 READ A(J)
   40 PRINT SIN(A(J))/(LOG(A(J))+EXP(A(J)))
   50 NEXT J
   60 END

```

9.4 DEFINING YOUR OWN FUNCTIONS

In mathematics, functions are usually defined by specifying one or more formulas. For instance, here are formulas which define three functions $f(x)$, $g(x)$, and $h(x)$:

$$f(x) = (x^2 - 1)^{1/2}$$

$$g(x) = 3x^2 - 5x - 15$$

$$h(x) = 1/(x - 1)$$

Note that each function is named by a letter, namely f , g , and h , respectively. Applesoft BASIC allows you to define functions like these and to use them by name throughout your program. To define a function, you use the DEF FN instruction. This instruction is used before the first use of the function in the program. For example, to define the function $f(x)$ above, we could use the instruction

```
10 DEF FNF(X) = (X^2 - 1)^(1/2)
```

To define the function $g(x)$ above, we use the instruction

```
20 DEF FNG(X) = 3*X^2 - 5*X - 15
```

Note that in each case, we use a letter (F or G) to identify the function. Suppose that we wish to calculate the value of the function F for $X = 12.5$. Once the function has been defined, this calculation may be described to the computer as FNF(12.5). Such calculations may be used throughout the program and save the effort of retyping the formula for the function in each instance.

You may use any valid variable name as a function name. For example, you may define a function INTEREST by the statement:

```
10 DEF FNINTEREST(X) = . . .
```

Note, however, that only the first two letters of the variable are used to identify the function. Any other variable name beginning with IN would refer to the same function as the one above.

In defining a function, you may use other functions. For example, if FNF(X) and FNG(X) are as defined above, then we may define their product by the instruction

```
30 DEF FNC(X) = FNF(X)*FNG(X)
```

EXERCISES (answers on page 243)

Write instructions to define the following functions.

1. $x^2 - 5x$

2. $1/x - 3x$

3. $5\exp(-2x)$

4. $x \log(x/2)$

5. $\tan x/x$

6. $\cos(2x) + 1$

7. Write a program which tabulates the value of the function in Exercise 3 for $x = 0, .1, .2, .3, .4, \dots, 10.0$.

10

Computer-Generated Simulations

10.1 SIMULATION

Simulation is a powerful analysis tool which let's you use your computer to perform experiments to solve a wide variety of problems which might be too difficult to solve otherwise.

To describe what simulation is, let us use a concrete example. Assume that you own a dry cleaning store. At the moment, you have only one salesperson behind the counter, but you are considering adding a second. Your question is: Should you hire the extra person? Being an analytical person, you have collected the following data. Traffic through your store varies by the hour. However, you have kept a log for the past month and are able to estimate the average number of potential customers arriving in the shop according to the following table:

7-8 AM	30
8-9	15
9-10	6
10-11	3
11-12	8
12-1PM	25
1-2	9
2-3	8
3-4	12
4-5	12
5-6	35
6-7	22

You have observed that you are currently paying a penalty for not having a second salesperson: If there is too long a wait, a customer will go somewhere else to have his or her clothes cleaned! In your observations, you have noted that, on the average, of people entering the shop a certain percentage will leave, depending on the size of the line. The likelihood that a person leaves depends on whether it is a drop-off or a pick-up. Those picking up clothes are more likely to wait in line. Here are the results of your observations:

line size	% leaving (drop-off)	% leaving (pick-up)
0	0	0
1-3	15	5
4-6	25	15
7-10	60	35
11-15	80	50

The average time to wait on a person is four minutes and the size of the average cleaning bill is \$5.75. The cost of hiring the new salesperson is 200 dollars per week. Assuming that the salesperson works continuously while the shop is open, what action should you take?

This problem is fairly typical of the problems which arise in business. It is characterized by data accumulated from observations and unpredictable events. (When will a given customer arrive? Will he or she encounter a long line? Will he or she be the impatient sort who walks out?) Nevertheless, you must make a decision based on the data you have. How should you proceed?

One technique is to let your computer "imitate" your shop. Let the computer play a game which consists of generating customers at random times. These customers enter the "shop" and, on the basis of the current line, decide whether or not to stay. The computer will keep track of the line, the number of customers who leave, the revenue generated, and the revenue lost. The computer will keep up the simulated traffic for an entire "day" and present you with the results of the daily activity. But, you might argue, the computer data may not be valid. Suppose that it generates a "non-typical" day. Its data might be biased. This could, indeed, happen. In order to avoid this pitfall, we run the program for many simulated days and average the results. The process we have just described is called **simulation**.

In this chapter we will provide a glimpse of the power of simulation and provide you with enough of an idea to build simple simulations of your own.

First, let us handle some of the mathematical ideas we will need in the next section. The required notions center around the following question: How do we make the computer imitate an unpredictable event? Consider the irate customer who arrives to drop off cleaning and encounters a line of four people ahead of him. According to the above table, the customer will leave 25 percent of the time and remain in line 75 percent of the time. How do you let the computer make the decision for the customer?

Easy! Use the random number generator. Recall that RND(1) generates a random number between 0 (included) and 1 (excluded). Suppose we ask how often RND(1) is larger than .25. If, indeed, the numbers produced by the random number generator show no biases, approximately 75 percent of the numbers produced will lie in the given interval since this interval occupies 75 percent of the length of the interval from 0 to 1. We let our customer decide as follows: If $RND(1) > .25$ then the customer joins the line; otherwise, the customer walks out in a huff. We will employ this simple idea several times in designing our simulation.

10.2 SIMULATION OF A DRY CLEANERS

Let us build a simulation to solve the problem stated in the preceding section. We must decide on techniques for imitating each of the important aspects of the problem.

Since the problem calls for analysis of actions as time passes, we must somehow measure the passage of (simulated) time. To do this, we will use the variables TH (time-hours) and TM (time-minutes) to keep track of the current time. In order to avoid a problem with AM and PM, let's use the military time system. In this system the PM hours are denoted as 13 through 24. For example, 1:15 PM is shown as 13:15. As our unit of simulated time, let's use four minutes, the time it takes to serve one customer. Our program will then look at time in four-minute segments. During each four-minute segment, it will take certain actions and then advance to the next time segment by adjusting TH and TM. When TM exceeds 60 minutes (an hour), we subtract 60 and credit the 60 minutes to TH, which is increased by 1. Let us do the time advance in a subroutine at line 1000. Here is the subroutine.

```

1000 REM TIME ADVANCE
1010 LET TM=TM+4
1020 IF TM>=60 THEN 1030
1025 GOTO 1100
1030 TM=TM-60: TH=TH+1
1100 RETURN

```

Let us store the statistical data on customer arrivals in the array A(H) (H = 7, 8, . . . , 18). A(7) will equal the number of customers arriving between 7 and 8 AM, A(8) the number arriving between 8 and 9 AM, . . . , A(18), the number arriving between 6 and 7 PM. The first action of the program is to set up this array.

```

10 DIM A(18)
20 DATA 30,15,6,3,8,25,9,8,12,12,35,22
30 FOR H=7 TO 18
40 READ A(H)
50 NEXT H

```

The next step will be to read in the customer "impatience data." Let DROPOFF(K) be the percentage of drop-off customers who leave when the line is K people long. Let PICKUP(K) be the corresponding statistic for pick-up customers. Here is the portion of the program which sets up these arrays.

```

100 DIM DROPOFF(20), PICKUP(20)
110 DATA 0,0,.15,.05,.25,.15,.60,.35,.80,.50
115 READ DROPOFF(0),PICKUP(0)
120 READ DROPOFF(1),PICKUP(1)
130 DROPOFF(2)=DROPOFF(1): DROPOFF(3)=DROPOFF(1)
140 PICKUP(2)=PICKUP(1): PICKUP(3)=PICKUP(1)
150 READ DROPOFF(4), PICKUP(4)
160 DROPOFF(5)=DROPOFF(4): DROPOFF(6)=DROPOFF(4)
170 PICKUP(5)=PICKUP(4): PICKUP(6)=PICKUP(4)
180 READ DROPOFF(7), PICKUP(7)
190 DROPOFF(8)=DROPOFF(7): DROPOFF(9)=DROPOFF(7):
    DROPOFF(10)=DROPOFF(7)
200 PICKUP(8)=PICKUP(7): PICKUP(9)=PICKUP(7): PICKUP(10)=PICKUP(7)
210 READ DROPOFF(11), PICKUP(11)
220 FOR J=12 TO 15
230 DROPOFF(J)=DROPOFF(11): PICKUP(J)=PICKUP(11)
240 NEXT J

```

The next step in our program is to set the clock at the beginning of a day. Likewise, the length of the waiting line, indicated by the variable L, is set equal to 0, the total of lost cash indicated by the variable LC, and the total sales for the day indicated by the variable CF (cash flow), are both set equal to 0.

```

300 TH=7: TM=0
310 L=0
320 LC=0
330 CF=0

```

At the beginning of each hour, the program will schedule the arrival of the customers. For the Jth hour, it will schedule the arrival of A(J) customers. Each customer will be given a time of arrival in minutes past the hour. The computer will choose this arrival time using the random

number generator. In the absence of any other information, let's assume that the customers spread themselves out in a random, but uniform manner, over the hour. The way we'll handle things inside the computer is as follows. At the beginning of each simulated hour, we set up an array $D(T)$ with 15 entries, one for every four-minute period in the hour. This array will indicate how many customers arrive in each four-minute interval. For example, if $D(10) = 4$, then four customers will arrive between 36 and 40 minutes past the hour (that is, in the tenth four-minute interval in the hour). The program will randomly place each of the $A(J)$ customers in four-minute intervals using the random number generator. Our program will test the time for the beginning of an hour. This will be done by determining if TM equals 0 (in line 410). If so, the program will go to a subroutine at 1200 which schedules the arrival of the customers for the hour.

```

11 DIM D(15)
410 IF TM=0 THEN GOSUB 1200
1200 FOR S=1 TO 15
1210 D(S)=0
1220 NEXT S
1230 FOR I=1 TO A(TH)
1240 X=INT(15*RND(1))+1
1250 D(X)=D(X)+1
1260 NEXT I
1270 RETURN

```

The program now progresses through the simulated hour in four-minute segments. For the T th four-minute segment, it causes $D(T)$ customers to arrive at the shop. Let's assume, that of these customers, half are drop-off and half are pick-up. The computer lets these customers each look at the line and decide whether to leave or stay. If a customer decides to stay, then he or she is added to the line. If the customer decides to go, the computer makes a note of the \$5.75 cash flow lost. The lost cash flow will be stored in the variable LC . After the customers are either in line or have left, the salesperson services a customer (remember, one customer is serviced every four minutes) and \$5.75 is added to the cash flow, which is tallied in the variable CF . Finally, the time is updated and the entire procedure is repeated for the next four-minute segment. Let's be rather hard-hearted. If there are any customers left in line at closing time, we don't wait on them and add their business to that lost. This rather odd way of doing business is appropriate since we are analyzing the need for more personnel and any overtime should be considered in that analysis. Here is the portion of the program which accomplishes these tasks of the simulation. Note that line 720 tests the time for the end of the working day ($TH = 19$). When

this time arrives, the program goes to line 1500, where the end of day statistics are printed out.

```

420 T=TM/4+1:REM T=# OF 4 MIN. SEGMENT
430 FOR J=1 TO D(T)
440 LET C=INT(2*RND(1))+1:
445 REM 1=DROP-OFF,2=PICK-UP
450 IF C=2 THEN 600
490 REM 500-560 DOES DROP-OFF CUST. STAY?
500 IF RND(1)>DROPOFF(L) THEN 550 : GOTO 510
510 LC=LC+5.75: REM CUSTOMER LEAVES
520 GOTO 690
550 L=L+1:REM CUSTOMER JOINS LINE
560 GOTO 690
590 REM 600-660 DOES PICK-UP CUST. STAY?
600 IF RND(1)>PICKUP(L) THEN 640 : GOTO 610
610 LC=LC+5.75:REM CUSTOMER LEAVES
620 GOTO 690
640 L=L+1:REM CUSTOMER JOINS LINE
690 NEXT J
700 IF L=0 THEN 710 : GOTO 701:REM THERE WERE NO CUSTOMERS
701 L=L-1:REM WAIT ON CUSTOMER
705 CF=CF+5.75 : REM TAKE CUSTOMER'S MONEY
710 GOSUB 1000 : REM UPDATE TIME
720 IF TH=19 THEN 1500 : GOTO 800
725 GOTO 800
730 REM TH=19 IS THE END OF THE DAY
800 GOTO 410:REM GO TO NEXT 4 MINUTE SEGMENT
1500 PRINT "END OF DAY STATISTICS"
1510 PRINT "BUSINESS LOST", LC+L*5.75
1520 PRINT "CASH FLOW", CF
1530 PRINT "LINE AT CLOSING", L
2000 END

```

Our program simulates the activities of a single day. In order to average the statistics over a number of days, let's set up a loop which repeats the above program for a certain number of days. Let's make an arbitrary choice of ten days repetition. Let the variable D denote the day number. Let us also denote the day number by E, then our change of day will be controlled by a loop in lines 290 and 1700.

```

290 FOR E=1 to 10
1700 NEXT E

```

As statistics, let us compute the average of the revenue lost (LC), cash flow (CF), and the line length at closing (L). We keep the totals of these three variables for all the days up to the present in the variables L1, C1, and CL, respectively. We modify lines 1500-1530 as follows:

```

1500 LET L1=LC+L1+L*5.75
1510 LET C1=CF+C1
1520 LET CL=L+CL

```


Lines 1800-1850 compute the averages of L1, C1, and CL and display the results.

```

1800 LET L1=L1/10
1810 LET C1=C1/10
1820 LET CL=CL/10
1830 PRINT "AVERAGE CASH LOST PER DAY", L1
1840 PRINT "AVERAGE CASH FLOW PER DAY", C1
1850 PRINT "AVERAGE LINE LENGTH AT CLOSING", L

```

This completes the construction of our program. We have carried out the construction of the program in detail so that you could see how a reasonably lengthy program is developed. However, our program is in a rather poor form to read, so let's recopy it in order.

```

10 DIM A(18)
11 DIM D(15)
20 DATA 30,15,6,3,8,25,9,8,12,12,35,22
30 FOR H=7 TO 18
40 READ A(H)
50 NEXT H
100 DIM DROPOFF(20), PICKUP(20)
110 DATA 0,0,.15,.05,.25,.15,.60,.35,.80,.50
115 READ DROPOFF(0), PICKUP(0)
120 READ DROPOFF(1),PICKUP(1)
130 DROPOFF(2)=DROPOFF(1): DROPOFF(3)=DROPOFF(1)
140 PICKUP(2)=PICKUP(1): PICKUP(3)=PICKUP(1)
150 READ DROPOFF(4), PICKUP(4)
160 DROPOFF(5)=DROPOFF(4): DROPOFF(6)=DROPOFF(4)
170 PICKUP(5)=PICKUP(4): PICKUP(6)=PICKUP(4)
180 READ DROPOFF(7), PICKUP(7)
190 DROPOFF(8)=DROPOFF(7): DROPOFF(9)=DROPOFF(7):
    DROPOFF(10)=DROPOFF(7)
200 PICKUP(8)=PICKUP(7): PICKUP(9)=PICKUP(7):
    PICKUP(10)=PICKUP(7)
210 READ DROPOFF(11), PICKUP(11)
220 FOR J=12 TO 15
230 DROPOFF(J)=DROPOFF(11): PICKUP(J)=PICKUP(11)
240 NEXT J
290 FOR E=1 TO 10
300 TH=7: TM=0
310 L=0
320 LC=0
330 CF=0
410 IF TM=0 THEN GOSUB 1200
420 T=TM/4+1
430 FOR J=1 TO D(T)
440 LET C=INT(2*RND(1))+1:REM 1=DROP-OFF, 2=PICK-UP
450 IF C=2 THEN 600
490 REM 500-560 DOES DROP-OFF CUST. STAY?
500 IF RND(1)>DROPOFF(L) THEN 550
510 LC=LC+5.75: REM CUSTOMER LEAVES

```

```

520 GOTO 690
550 L=L+1:REM CUSTOMER JOINS LINE
560 GOTO 690
590 REM 600-660 DOES PICK-UP CUST. STAY?
600 IF RND(1)>PICKUP(L) THEN 640
610 LC=LC+5.75:REM CUSTOMER LEAVES
620 GOTO 690
640 L=L+1:REM CUSTOMER JOINS LINE
690 NEXT J
700 IF L=0 THEN 710 : GOTO 701:REM THERE WERE NO CUSTOMERS
701 L=L-1:REM WAIT ON CUSTOMER
705 CF=CF+5.75 : REM TAKE CUSTOMER'S MONEY
710 GOSUB 1000 : REM UPDATE TIME
720 IF TH=19 THEN 1500
730 REM TH=19 IS THE END OF THE DAY
800 GOTO 410:REM GO TO NEXT 4 MINUTE SEGMENT
1000 REM TIME ADVANCE
1010 LET TM=TM+4
1020 IF TM<60 THEN RETURN
1030 TM=TM-60: TH=TH+1
1100 RETURN
1200 FOR S=1 TO 15
1210 D(S)=0
1220 NEXT S
1230 FOR I=1 TO A(TH)
1240 X=INT(15*RND(1))+1
1250 D(X)=D(X)+1
1260 NEXT I
1270 RETURN
1500 LET L1=LC+L1+L*5.75
1510 LET C1=CF+C1
1520 LET CL=L+CL
1700 NEXT E
1800 LET L1=L1/10
1810 LET C1=C1/10
1820 LET CL=CL/10
1830 PRINT "AVERAGE CASH LOST PER DAY"; L1
1840 PRINT "AVERAGE CASH FLOW PER DAY";C1
1850 PRINT "AVERAGE LINE LENGTH AT CLOSING"; L
2000 END

```

In order to see what is happening at our hypothetical dry cleaning establishment, we run our program. Below are the results of five program runs.

```

RUN #1
AVERAGE CASH LOST PER DAY      258.75
AVERAGE CASH FLOW PER DAY     805.00
AVERAGE LINE AT CLOSING        9

```

RUN #2

AVERAGE CASH LOST PER DAY	270.25
AVERAGE CASH FLOW PER DAY	793.50
AVERAGE LINE AT CLOSING	3

RUN #3

AVERAGE CASH LOST PER DAY	264.50
AVERAGE CASH FLOW PER DAY	799.25
AVERAGE LINE AT CLOSING	4

RUN #4

AVERAGE CASH LOST PER DAY	270.83
AVERAGE CASH FLOW PER DAY	792.93
AVERAGE LINE AT CLOSING	6

RUN #5

AVERAGE CASH LOST PER DAY	287.50
AVERAGE CASH FLOW PER DAY	776.25
AVERAGE LINE AT CLOSING	4

We note several interesting facts about the output. First note that the runs are not all identical. This is because the random number generator generates new random customer arrival patterns for each run. Second, note the small percentage error in the data from the various runs. We seem to have discovered a statistical pattern which persists from run to run.

Finally, and most significantly, note that we are losing several hundred dollars per day in business because of our inability to service customers. At 200 dollars per week, the additional salesperson is a bargain! Even a single day's lost sales is enough to pay the salary. It appears that we should add the extra salesperson. Actually, a bit more caution is advisable. We were dealing with cash flow rather than profit. In order to make a final decision, we must compute the profit generated by the additional sales. For example, if our profit margin on plant costs (exclusive of sales) is 40 percent, then the profit generated by the extra sales will clearly amount to more than 200 dollars per week and the extra salesperson should be hired.

The above example is fairly typical of the way in which simulation may be applied to analyze even fairly complicated situations in a small business. We will present some further refinements in the exercises.

EXERCISES

1. Run the above program for ten consecutive runs and record the data. Does your data come close to the data presented above?

2. Suppose that customers become more impatient and the likelihood of leaving is doubled in each case. Rerun the experiment to determine the lost cash flow in this case.
3. Suppose that customers become more patient and the likelihood of leaving is cut in half in each case. Redo the experiment to determine the lost cash flow in this case.
4. Consider the original set of experimental data. Now assume that the second salesperson has been hired. Rerun the experiment to determine the average lost cash flow and the average line at closing.
5. Modify the given program so that you may calculate the average waiting time for each customer.

11

Software You Can Buy

We have concentrated mostly on writing BASIC programs to do various tasks. Actually, most people do not have the time or the inclination to write programs to perform all the tasks they want their computer to perform. At some time or other you will want to purchase programs which have been written by others. In recent years there has been a virtual explosion of commercially available programs (or, in computer jargon, software). In this chapter we will discuss some of the “ins and outs” of purchasing such software for your computer. We will begin by discussing one of the programs you are almost sure to want, VISICALC.

11.1 VISICALC*

One of the most popular pieces of software ever created is the VISICALC program. This program is a financial planning program that can be used in hundreds of situations, both business and personal.

Let’s illustrate a typical VISICALC application—a home budget. Suppose that the following data describe the income and spending of the Smith family.

*VISICALC is a registered trademark of Personal Software, Inc.

Income	
John	\$1580 per month
Sally	\$1145 per month
Interest	\$280 in Jan., Apr., Jul., Oct.
Bonus	\$3500 in Dec.
Expenses	
Mortgage	\$487.73 per month
Car payment	\$187.50 per month
Utilities	\$207.00 per month
Clothing	\$200.00 per month
House repairs	\$100.00 per month
Food	\$640.00 per month
Entertainment	\$120.000 per month
Fuel	\$135.00 per month
Life insurance	\$583.00 in Oct.
Vacation	\$1,000.00 in Jul.
Health insurance	\$285.00 in Apr. and Oct.
Pension plan	\$400.00 per month

Everyone has probably compiled a list of this sort at one time or another. Once the list is compiled, there are a number of important questions to ask, such as:

1. Is the Smiths' income sufficient to cover their expenses?
2. Is there any month in which they will be short of cash?
3. How much is their annual savings (deficit)?

To answer such questions (especially the second one) the best approach is to compile a month-by-month description of income and expenses. Such a description could be written out on an accounting worksheet in the following fashion:

	Jan.	Feb.	Mar.	. . .	Dec.	Year
Income						
·						
·						
·						
Expenses						
·						
·						
·						
Total						
Income						

Total
Expenses

Income
minus
Expenses

Cumulative
Total

We will leave it to you to fill in the above chart from the given data. If you do, you will discover that the Smith's are not making ends meet. There are a number of months in which they have a deficit and they are running at a deficit for the entire year. Being sensible people, the Smith's want to recast their budget so that they are not spending more than they are earning, and so that they are not running a deficit in any month.

They begin to experiment with various cutbacks in expenditures. For each proposed cutback, they must rework the above chart to determine if they have erased all the monthly deficits. It might take 15 or 20 tries before they arrive at a budget they can live with and which is free of deficits. The calculations involved are quite burdensome if they are done by hand. However, they are exactly the sort of calculations which can be handled easily by VISICALC!

VISICALC turns your computer screen into a large accounting worksheet, just like the budget worksheet above. Note that the Smith's budget required a chart consisting of 14 columns and 20 rows. VISICALC actually lets you have charts of up to 63 columns and 254 rows.

Here is how to use the VISICALC program to solve the Smiths' problem. To load the VISICALC program, you turn on the computer as usual and insert the VISICALC diskette. The program will automatically load into RAM. (It takes about 30 seconds.) When the disk drive light goes out, remove the diskette and hit **RETURN**. You are now ready to use the VISICALC program.

The first step is to enter the Smiths' data into the VISICALC worksheet. Note that the screen looks very much like the worksheet above. (See Figure 11-1.) The columns are labelled by letters and the rows by numbers. You may only refer to a single worksheet position at a time, namely the position indicated by the cursor. For example, in the worksheet of Figure 11-1, the currently indicated position is the first row, first column or position A1.

Our first task is to customize the worksheet to the data given in the Smiths' budget. Let's put the headings in the first row and first column.

A1

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										

Figure 11-1. A VISICALC worksheet.

You move the cursor around the worksheet in a manner similar to that used for editing programs. Just position the cursor at the appropriate workspace position and type the desired entry. To correct typing mistakes, use the Backspace key exactly as you would if you were typing a BASIC program. In our example, we would move the cursor to position B1 and type "Jan" followed by **RETURN**. We would then move the cursor one position to the right and type "Feb" and so forth. Similarly, we type the row labels beginning in the first column of row 2. Note that the first line of the display always indicates the current position of the cursor.

Let's now assume that the row and column headings are typed as in Figure 11-2. Note how the entries in the first column are truncated. VISICALC assumes a column width of seven characters, including a leading space. **Even though a complete entry is not displayed, it is retained in the computer (up to 125 characters).** You may increase the width of the columns by using the VISICALC command /GC. For example, suppose you type:

```
/GC23
```


A1

	A	B	C	D	E	F	G	H	I	J
1		Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.
2	Income									
3	John									
4	Sally									
5	Inter									
6	Bonus									
7										
8	Expens									
9	Mortg									
10	Car p									
11	Utili									
12	Cloth									
13	House									
14	Food									
15	Enter									
16	Fuel									
17	Life									
18	Vacat									
19	Healt									
20	Pensi									
21	Total									
22	Total									
23	Income									
24	Cumula									

Figure 11-2. VISICALC worksheet for the Smith budget.

The column width will expand to 23 columns, or enough to display the longest entry in column 1. After this command the worksheet will be as displayed in Figure 11-3. Note that the penalty imposed by using wider columns is that fewer of them simultaneously fit on the screen. However, by moving the cursor we may bring the hidden columns into view. For purposes of this example, let's go back to a seven character wide column by typing

/GC7

followed by **RETURN**.

We now wish to enter the budget data into the columns Jan.-Dec. Let's begin with the Jan. column. We must position the cursor at location B3. We could do this using a combination of one-space cursor moves. However, it is easier to use the command

>B3

	A	B	C
1		Jan.	Feb.
2	Income		
3	John		
4	Sally		
5	Interest		
6	Bonus		
7			
8	Expenses		
9	Mortgage		
10	Car payment		
11	Utilities		
12	Clothing		
13	House repairs		
14	Food		
15	Entertainment		
16	Fuel		
17	Life insurance		
18	Vacation		
19	Health insurance		
20	Pension plan		
21	Total Income		
22	Total Expenses		
23	Income Minus Expenses		
24	Cumulative Total		

Figure 11-3. Worksheet with expanded columns.

followed by **RETURN**. The cursor is now positioned at B3 and we type John's income for January, namely 1580. We then move the cursor down one entry and type Sally's income for January, and so forth. In similar fashion, we enter all the data in the budget for the month of January.

Let's now calculate total income and total expenses for January. Total income for January is obtained by forming the sum of the entries B3-B6. This may be indicated on the worksheet by typing

+B3+B4+B5+B6

(Note the leading + sign. This is necessary to notify VISICALC that B3 is not a heading but the location of an entry.) If you type this entry into location B21, VISICALC will automatically insert the required sum at this location.

To calculate the total expenses, it is necessary to add up the entries in B9 through B20. We could write this as a sum as shown above. However, VISICALC provides a shortcut. Position the cursor at position B22

and type

@SUM(B9..B20)

The entry in B22 will then be calculated as the sum of the entries in B9 through B20. Entry B23, income minus expenses, is just

+B21-B22

Entry B24 is the same as B23, so we type

+B23

This completes the entry of the column corresponding to January.

To complete the second column, it is not necessary to go through all the trouble we just went through for the month of January. Indeed, all entries are the same, except that the 280 dollar item for interest must be changed to 0. VISICALC has a feature which allows you to replicate any sequence of entries. To replicate the B column, we type

/R

followed by **RETURN**. At the top of the screen, you will see the prompt

Replicate:Source Range or RETURN

The computer is asking us what it is we wish to copy. Type

B2..B24

followed by **RETURN** to indicate that we wish to copy the contents of the B column. The program will respond with the prompt:

Replicate: Target Range

Respond by typing

C2..M2

followed by **RETURN** to indicate that the source B2-B24 is to be replicated beginning in each of the locations C2, D2, E2, . . . , M2. (M is the thirteenth letter of the alphabet, corresponding to the 12 columns to be replicated.) This sequence of inputs will cause the numbers in each of the entries of column B to be copied across to the 11 other month columns of the worksheet. However, when VISICALC encounters a formula in the B column (as in positions 21-24), it asks whether the formula is to be copied literally, or changed relative to its new position. For example, when the program reaches position B21, the following lines will be displayed:

B21: +B2+B3+B4+B5

Replicate:N=No Change R=Relative

In response, you type either N or R, followed by **RETURN**. Typing N will

duplicate precisely the same formula across row 21. However, typing R will change the formula to

+C2+C3+C4+C5

in column C,

+D2+D3+D4+D5

in column D, and so forth. Clearly, we want the latter.

Actually, we didn't mean to replicate every entry in column B. For example, the interest entry does not occur every month, and the bonus entry must be changed in the December column. These changes are easily made.

The worksheet is now complete except for the final column, N, which records totals, and the final row which records cumulative totals. Let's first fill in the final column. For example, in position N2, we want the sum of B2, C2, . . . , M2. We calculate this sum by positioning the cursor at N2 and typing

@SUM(B2.M2)

We now replicate this formula down the column. The Source Range is N2.N2 and the target range is N3.N23. The formula is relative to its position. (In N3 we want **@SUM(B3.M3)**.) Following the procedure outlined above, we then fill in the last column.

In the final row, the entry at C24 is B24 + C23. That is, the cumulative surplus (deficit) for February is the surplus (deficit) for January (entry B24) plus the surplus (deficit) for February (entry C23). In a similar fashion, the cumulative surplus for each month may be obtained by adding the cumulative surplus for the preceding month to the surplus for the current month. We then insert **+B24 + C23** in position C24 and replicate this formula in relative fashion in positions D24 through M24. This completes the development of the worksheet for the Smith budget. We may save the worksheet by typing

/SS BUDGET

The worksheet will be saved on a diskette under the file name "budget". To retrieve "budget" from the diskette on drive 1:, type:

/SL 1:BUDGET

Once the worksheet has been created, you may alter any entry in it, and all the other entries will be recalculated according to the relationships specified. In the exercises we will explore ways of removing the Smith deficit.

In the above discussion, we have explored only a few of the clever features of the VISICALC program. A complete discussion is beyond the

scope of this book. But we hope that we have aroused your interest sufficiently so that you will consider adding this program to your software library.

EXERCISES

1. Prepare the Smith budget worksheet as outlined in the discussion of the section.
2. What is the Smith deficit for the year?
3. What are the monthly surpluses (deficits) in the Smith budget?
4. The Smiths' propose reducing their entertainment budget by 25 dollars per month. What effect does this have on their deficits?
5. The Smiths' propose reducing their entertainment budget by 50 dollars per month. What effect does this have on their deficits?
6. Go back to the original budget worksheet. Suppose that the Smiths use savings to pay off their car loan. This will reduce their quarterly interest to 147 dollars . What effect will this have on their monthly deficits?
7. Assume that Mrs. Smith gets a raise yielding an increase in take home pay of 400 dollars per month. What effect will this have on monthly deficits?

11.2 BUYING SOFTWARE

As you probably have observed by now, it is not always easy to get a complex program up and running. In order to write and debug a complex program takes a considerable investment in time, wit, and dogged determination. In addition, to build the most complicated programs it takes a considerable amount of technical expertise in using the various features of the computer. Most people want to use their computer to simplify various everyday tasks, but are not interested in building their major applications programs from the ground up. For these people, there is a growing collection of programs which are available through computer stores and mail order houses.

There are commercially available programs for almost every conceivable need. These programs include computer games, word processing systems, inventory control systems, appointment and record-keeping systems for professionals (doctors, dentists, lawyers), bookkeeping systems for small, medium, and large businesses, and so forth. Unfortunately, the rapid introduction of new products and the large number of

available programs has made the purchase of software quite a chore. In this section, let's discuss a few pointers to help direct you through the "software jungle."

Here are some questions which you should ask yourself as part of any software purchase.

Will the program run on my computer?

In order to run a particular program, you must have the proper operating system and an adequate amount of memory.

Programs are designed to run with a particular operating system. In this book, we have discussed only Apple DOS. There are other operating systems. For example, there is the popular CP/M operating system. This operating system is available on most microcomputers and therefore there are an exceptionally large number of programs which run with CP/M. In any case, in buying a program check that it will run under your operating system.

A description of a program will usually specify the amount of memory required. Obviously, you can't run a program requiring 64K of memory if you have only a 16K system!

Will the program do what I want it to?

In purchasing a program for a particular application, you will be faced with a wide range of choices. How can you properly choose among them? Frankly, this is a real problem.

As early as possible in the software selection process, you should define your own needs as exactly as you can. To determine the extent to which a given piece of software meets those needs, you must do a fair amount of digging. Your local computer store, reviews in computer journals, and software documentation are good sources of information.

Your local computer store is the first place you should seek information. You will often be able to make a choice based on the information you get there. Many computer journals contain reviews on major pieces of software. Especially useful are reviews which compare several similar programs.

Another source of comparative information is the software documentation itself. You may often purchase the manual separately from the software. This is an expense, but it is often the only way you can be sure that the program will do exactly what you want it to do. After you have narrowed your program down to the top two contenders, why not

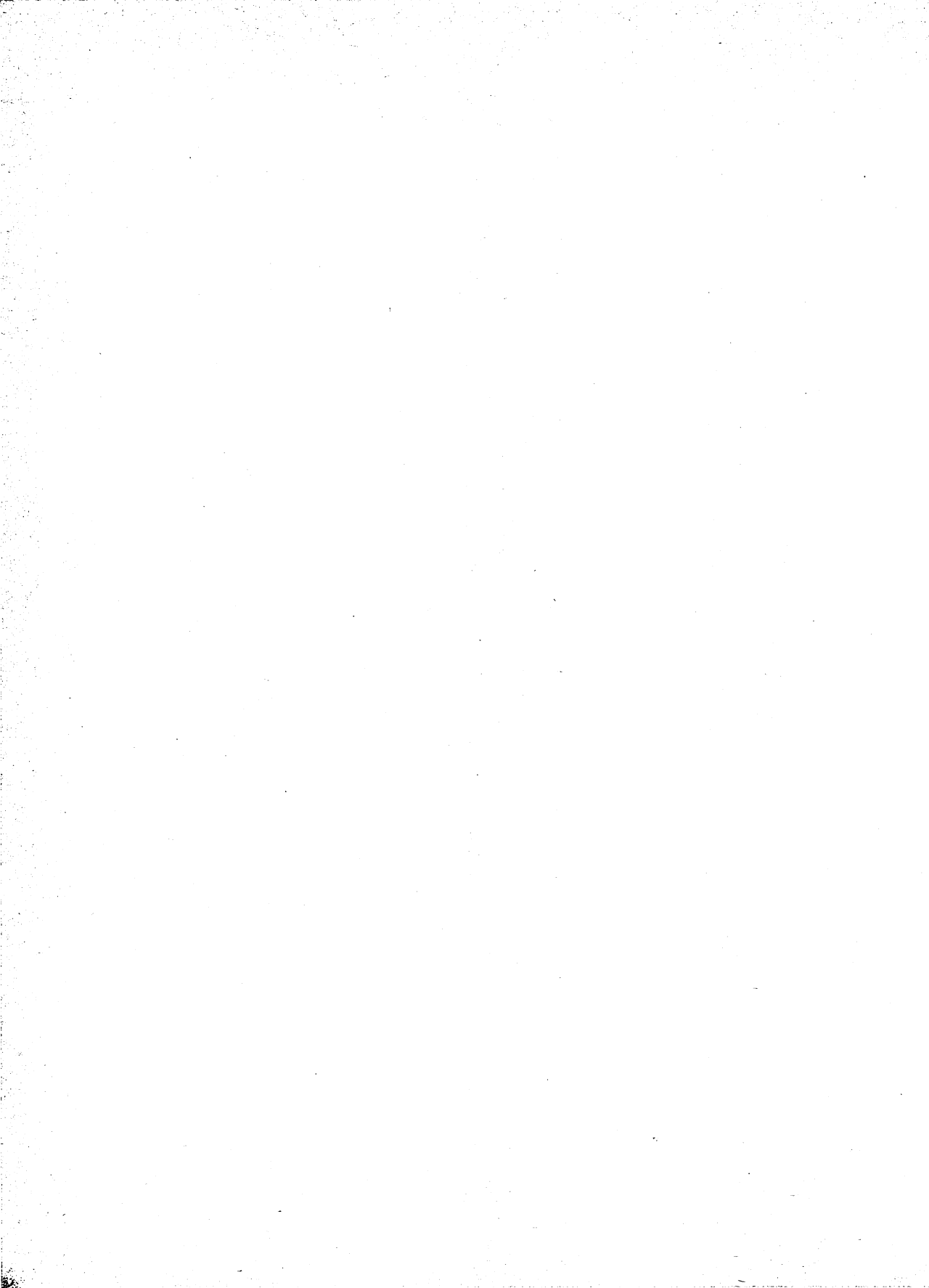
purchase the manuals of each and inspect the two programs at close range. It is much cheaper to invest in a manual you won't need than to spend several hundred dollars on a program which will not do everything you want it to!

What do I get for my software dollar?

When you purchase software the minimum you get is the program on disk or cassette, plus the applicable documentation. What else do you get?

1. Will the vendor accept phone calls seeking help in setting up and using the program?
2. What are the costs of updates to the software?
3. Will the vendor automatically notify you of major bugs in the software?
4. What is the quality of the documentation? Is it readily intelligible? Does it provide clear examples? Can it easily be used as a reference manual?
5. To what extent can you make required customizations of the software?
6. How much will it cost to replace defective copies of the program? Many software vendors currently supply you with only two copies of the program, which cannot be copied further. You may have to pay to replace them if they are damaged. Some programs may be copied at will from the original disk.

You should not be intimidated by the discussion above. Most software purchases proceed quite smoothly. However, as with any other area of consumer affairs, it pays to be an informed purchaser. Hopefully the suggestions above will help you spend your software dollars wisely.



12

Some Other Applications of Your Computer

In this chapter we will discuss additional applications of your computer. We will only be able to present a brief overview of these applications since a complete discussion would carry us beyond the scope of this book. Nevertheless, these applications are extremely important and you should be aware of them.

12.1 COMPUTER COMMUNICATIONS

At some point you will want to connect your computer to external devices (called **peripheral devices**). There are many devices available and more are being introduced at a frightening pace. At the moment, such devices include graphics screens, light pens, plotters, voice synthesizers, music synthesizers, and temperature probes, to mention only some of the possibilities. You will want the capability of connecting your computer to other computers so that you may interchange programs and data with other users.

In this section, we present some of the fundamentals of computer communications. Our purpose is not to make you an expert, but to introduce you to the ideas and vocabulary so that you may read and understand the articles in various computer journals.

In most cases, it is not possible to connect two electronic devices directly to one another. It is necessary to have an intermediate device which translates the electronic signals of one device into a form intelligible to the other device. Such an intermediate device is called an **interface**; the task of electronically mating the devices is called **interfacing**. For microcomputer interfacing, there is a standard interface device called an **RS232-C** interface. The RS232-C interface allows two devices to communicate with one another using a 25 wire cable. Each of the wires carries a signal having a standardized meaning. You may purchase an RS232-C interface for your Apple II Computer at your local computer dealer. Using this interface, you can connect your Apple II Computer to a wide variety of peripheral equipment manufactured by Apple and other outside vendors as well.

Before we go any further, a word of caution: Many devices are advertised as having a built-in RS232-C interface or as being "RS232-C compatible." It may require some hard work to make them operate with your computer! There are several reasons for this: Although all RS232-C interfaces utilize a 25 wire cable, not all the wires are necessarily used. Therefore, your computer (or rather your programs) may require a signal which is not being sent, or it is not sending a signal required at the other end. A further problem lies in the confusion of connecting data sets to data terminals. There are two conventions for wiring RS232-C interfaces—one for data terminals and one for data sets. In order to connect two devices using an RS232-C interface, one must be a data terminal and one must be a data set. If both are the same variety (say both computers), then it will be necessary to connect the interfaces on the devices by means of a special cable. The moral of all this is: When purchasing peripheral devices to connect to your computer, proceed with caution. Be sure your supplier is willing to help you or at least to exchange the device if you can't make it work.

Our purpose in this section is to introduce you to some of the ideas and the vocabulary of computer communications. To begin with, let's discuss in greater detail the form in which the computer stores data.

A binary number is a string of 0's and 1's. Here is a typical example of a binary number:

101110110111010100000011111

A binary digit (that is, a 0 or a 1) is called a **bit**. A string of eight consecutive bits is called a **byte**. Here are examples of two bytes:

10011001 11100011

In the computer, all data and instructions are written in terms of binary numbers, with each byte corresponding to a character. (Except

for specialized applications, don't concern yourself with the precise manner in which characters are translated into binary.) Because the basic unit of data within your machine consists of eight bits, it is called an **8 bit computer**. Larger computers (often called **main frames**) operate with 32 or 64 bits at a time. The added efficiency thus achieved accounts, in part, for their increased speed.

There are two fundamental types of computer communications: **parallel** and **serial**. In parallel communications, a byte is transmitted all eight bits at a time. This is achieved by sending the byte over eight wires. A signal on the wire corresponds to a 1 and the absence of a signal corresponds to a 0. In serial communications, the various bits are transmitted in sequence over a single wire. Many printers utilize serial communications. In addition, serial communications are used to transmit computer data to another computer using telephone lines. The interfaces required by parallel and serial communications are quite different.

In establishing a computer communications link, there are a number of different variables which must be considered. First, there is the speed of the communications. The standard measure of communications speed is the **baud rate**. Old-fashioned teletypes communicate at 110 baud (about 12 characters per second). Data transmission rates from your computer to a printer range from 300 to 1200 baud. High speed data transmission rates range up to 9600 baud. You may set the baud rate of your RS232-C interface using a computer command.

All communications links are subject to noise caused, primarily, by static on the lines. It is essential that computer data communications be accurate. Imagine the havoc that could be created by the erroneous transmission of a few digits of a financial report! In order to guard against such errors, many data transmission links utilize an extra bit which is tacked on to each byte. This extra bit is called a **parity bit**. The value of the parity bit depends on the sum of the other bits in the byte. It is agreed in advance whether the sum of the digits in a byte (including the parity bit) will be even (even parity) or odd (odd parity). In setting the parity bit, the computer then determines the sum of the bits in a byte. Suppose that a sum is odd and the parity is even. The parity bit will then be set to 1. The receiving device checks the parity bit to determine its correctness. If an error is detected, then a retransmission is usually requested. All this happens quite automatically. However, you must adjust your transmissions to match the parity expected. This can be done using a computer command to the RS232-C interface.

Finally, it is sometimes necessary to have a **communications protocol**. In some situations, it is useful to transmit the data at a speed higher than the receiver can accept. To do this, your computer sends the data

in "bursts." Your computer can utilize the waiting time between bursts to perform other chores. At the receiving end, each burst is temporarily stored in a memory called a **buffer**. This memory holds the burst of data until the receiver has a chance to look at it. In this scheme of data transmission, it is necessary to have a pair of signals that the sender and receiver exchange. Namely, the sender must tell the receiver that more data is on the way and the receiver must tell the sender that more data may be sent. Such an exchange of signals is called a *communications protocol*. There are a number of different protocols in common use. What these protocols are is not important, but it is crucial that the sender and receiver use the **same** protocol. You may select among the most common protocols using a computer command. Note that it is necessary to use a communications protocol only in situations in which the data transmission rate is too fast for the receiver. Typically, it is not necessary to use a communications protocol with a printer at 300 baud or less. However, to get the top printing speed out of a daisy wheel printer, it is necessary to go to 1200 baud.

12.2 INFORMATION STORAGE AND RETRIEVAL

In this book we have presented the basics of file construction and maintenance. What we have said will carry you through if your requirements are reasonably modest. However, if your files become very large or you wish the ability to sort through them and compile complex management reports, you will need more elaborate programs than anything we have discussed.

There are a large number of data retrieval systems which you can consider for your particular purposes. In fact, specialized programs are just appearing which are structured for the needs of a particular profession (lawyer, doctor, or architect). If your accounting and information management needs go beyond what we have discussed (or if you don't want to bother writing your own programs), you should investigate the various packages which are commercially available.

12.3 ADVANCED GRAPHICS

Computer graphics has become an incredibly sophisticated field in only a few years. Your Apple II Computer can be used to obtain an introduction to computer graphics principles. However, you can only go so far with a black and white video display. You probably should connect your

Apple II to a color television to get a feel for the beautiful color graphics possibilities.

Many printers have a graphics mode that lets you produce hard copies of screen graphics. This is usually accomplished using "dot printing" with a high resolution of dots. To obtain even finer hard copy graphics, there are a number of plotters available which (at their most sophisticated) can faithfully produce blueprints, weather maps, and other displays.

The microcomputer user can even add a graphics tablet. This is a device that allows you to input a picture to the computer by essentially tracing the picture on a special board using an electronic "pencil." The picture is transformed into a series of dots and transmitted to the computer via a communications interface.

To survey the latest in computer devices, you should attend one of the many computer shows which take place with increasing regularity all over the country. Also, a good written source is BYTE magazine, available at many computer stores.

12.4 CONNECTIONS TO THE OUTSIDE WORLD

You may connect your Apple II Computer to the outside world! To do so you must have an RS232-C interface and a special communications device called a **modem**.

A modem converts the electronic signals of your computer into signals which may be transmitted using telephone lines. A modem is connected to your computer with the RS232-C interface. To set up a telephone connection with another computer, you first dial the number of the outside computer. Once a connection has been made, you rest the telephone receiver in the cradle* of the modem. (Your RS232-C should be turned on and waiting.) You have now established a communications link between you and the outside world.

You may use this communications link in many ways. First, you may communicate with other microcomputer users. You can also play games, exchange data, program ideas, and so forth. You may even use the computer as an "electronic mail service." In fact, this application of

*Using a direct-connect modem, you are hooked directly into a telephone line and bypass the telephone receiver entirely.

computer communications promises to revolutionize the office in the next decade. Instead of sending paper memos and printed reports, you will send such data using computer communications. If the information is to be held confidential, access will then be regulated either by passwords or encoding. Just think! No more delayed mail delivery, lost letters, or other communications problems. As a microcomputer user, you can be one of the first to use such a system.

You may use computer communications to connect your operations to a time-sharing system which you have access to. This will give you access to the greater capabilities of a larger machine as well as the program library of the time-sharing system.

Finally, you may plug your computer into any one of several information networks. Such networks normally charge a monthly fee and provide the latest stock market quotations, news, and other facts. In addition, they provide a library of programs you may use. Such information services are in their infancy and are sure to grow in number and sophistication over the next few years.

13

Where to Go From Here

We have attempted to provide you with a working knowledge of the Apple II Computer and its associated BASIC language. Of course, we have only scratched the surface of the computer science field and the applications in which your computer can be used. In this final chapter, let's say a few words about some of the subjects we didn't have a chance to discuss in depth and point out some directions for further study.

13.1 ASSEMBLY LANGUAGE PROGRAMMING

All of our programming has been carried out in the BASIC language. There is a much more primitive language which underlies your Apple II Computer, namely 6502 machine language. Actually, BASIC is **itself** a program which is written in machine language. Indeed, many complex commercial programs are written directly in machine language.

Machine language consists of the instructions which the 6502 chip can execute. These instructions tend to be much more primitive than the instructions of a higher level language such as BASIC. In a certain sense, this is unfortunate since you are forced to look at a program in very fine steps. However, the resulting programs will generally be much

more efficient and will run much more quickly than programs written in BASIC. In addition, you will understand better what is going on inside the 6502 chip in response to your instructions.

After you have become competent in BASIC, your next step can be a study of machine language. In order to help you get started, let's spend a short time discussing how machine language works. As we have previously said, the internal workings of the computer are all carried out in binary. This includes machine language commands. However, it is extremely difficult to write a program which is nothing but a long string of 0's and 1's. To ease this tremendous burden, you write machine language commands in terms of **mnemonics**. These are similar to instruction designations used in BASIC. The program, written in terms of mnemonics is called the **source code**.

The next step in preparing a machine language program is to translate mnemonics into binary. This is done using a program called an **assembler**. The resulting program is called the **object code** or **machine code**. You may list the object code but it is extremely difficult to read since it consists of an endless string of 0's and 1's. To ease this burden, computer scientists use a notational system consisting of 16 symbols, namely 0–9 and A–F. This system is called the **hexadecimal** system and may be used to list the object code of a program. Moreover, all memory addresses are specified in terms of hexadecimal notation. Because of its direct relationship to binary, hexadecimal notation is directly intelligible to the computer.

In the process of running the assembler, you must decide where in memory your program is to be stored. This is a complication that you don't worry about in BASIC programming. BASIC finds memory space and keeps track of where the various parts of the program are located. However, in machine language programming, all the internal book-keeping is your responsibility. Once your program is assembled, you are ready to load and run it.

You might wonder if machine language programming is really worth the effort described above. Probably not in the case of a program you plan to use once or twice. However, if you are planning a program which you will be using often, perhaps as a subroutine in many different BASIC programs, it will then probably be worth the invested time to write the program in machine language. First of all, your program will run much faster. Second, you will be able to make the screen, keyboard, and printer perform actions which may be clumsy or downright impossible to specify in BASIC.

13.2 OTHER LANGUAGES AND OPERATING SYSTEMS

BASIC is only one of several hundred different computer languages. It is only one of the possible languages which is available to run on your Apple II Computer. Mastering one or more of these other languages is another possible area for further study.

As microcomputers have become more common, many of the languages designed to run on large computer systems have been configured for microcomputers. Any list of available languages will probably be incomplete by the time this book goes to press. Nevertheless, let's mention some of the most common languages which are available for the Apple II computer.

The old standard of computing languages is FORTRAN. This is a powerful language especially useful in scientific and engineering applications.

FORTRAN is a **compiler**, while Applesoft BASIC is an **interpreter**. This is an important distinction. With an interpreter, you type in the program directly as it will be executed. In order to execute a particular instruction, the computer refers to a machine language subroutine to "interpret" the intent. With a compiler, you type in the program in much the same manner as with an interpreter. However, you must **compile** the program prior to running it. That is, you must run a special routine which translates the various typed instructions into machine language. It is the machine language version of the program which you actually run. A compiled program is much more efficient than a program written with an interpreter. Depending on the program, the compiled program will run from 5 to 50 times faster!

You may supplement your BASIC interpreter with a BASIC compiler. This will allow you to program in a language you already know (once you learn the intricacies of the compiler version) and yet achieve the efficiencies of compiled programs.

COBOL and PASCAL are two very popular languages. COBOL is probably the most commonly used language for business programs. It is designed to allow ease in preparing management and financial reports. PASCAL is an extremely powerful language which may be used for general programming. It allows you to write complex programs in very few commands.

In purchasing languages for your computer, it is important to recognize that the particular version you purchase must be compatible with

your operating system. The Apple II Computer disk operating system is Apple DOS. In order to make use of other languages or other programs, you may wish to add another operating system. The most likely candidate is CP/M, the most common microcomputer operating system. This may be accomplished by adding an additional card in one of the empty slots of the Apple II.

We hope that this book has sparked your interest in microcomputing and that you will pursue some of the suggestions for further study. Good luck!

Answers to Selected Exercises

CHAPTER 2

Section 2.2 (page 22)

1. 10 PRINT 57+23+48
20 END
2. 10 PRINT 57.83*(48.27-12.54)
20 END
3. 10 PRINT 127.86/38
20 END
4. 10 PRINT 365/.005+1.02^5
20 END
5. 10 PRINT 2^1,2^2,2^3
20 PRINT 3^1,3^2,3^3
30 PRINT 4^1,4^2,4^3
40 PRINT 5^1,5^2,5^3
50 PRINT 6^1,6^2,6^3
60 END
6. 10 PRINT "CAST REMOVAL",45
20 PRINT "THERAPY",35
30 PRINT "DRUGS",5
40 PRINT
50 PRINT "TOTAL",45+35+5
60 PRINT "MAJ MED", .8*(45+35+5)

220 ANSWERS TO SELECTED EXERCISES

```
70 PRINT "BALANCE", .2*(45+35+5)
80 END
```

```
7. 10 PRINT "THACKER", 698+732+129+487
    20 PRINT "HOVING", 148+928+246+201
    30 PRINT "WEATHERBY",379+1087+148+641
    40 PRINT "TOTAL VOTES", 698+732+129+487+148+928+246+
        201+379+1087+148+641
    50 END
```

Note that line 40 extends over two lines of the screen. To type such a line just keep typing and do not hit a carriage return until you are done with the line. The maximum line length is 255 characters.

8. -2

9. SILVER	GOLD	COPPER
327	448	1052

10.	GROCERIES	MEAT
MON	1,245	2,348
TUE	248	3,459

11. 2.3E7

12. 1.7525E2

13. -2E8

14. 1.4E-4

15. -2.75E-10

16. 5.342E16

17. 159,000

18. -20,345,600

19. -.0000000000007456

20. .0000000000000000239456

Section 2.3 (page 31)

1. 10

2. 0

3. 50

4. 9 -7 18

5. JOHN JONES AGE 38
6. 22
57
7. A can only assume numeric constants as values.
8. Nothing
9. A\$ can only assume string constants as values.
10. No line number, string constant not in quotes.
11. Nothing
12. A variable name must begin with a letter.
13. 10 LET A=2.3758:B=4.58321:C=58.11
20 PRINT A+B+C
30 PRINT A*B*C
40 PRINT A^2+B^2+C^2
50 END
14. 10 LET A\$="OFFICE SUPPLIES":B\$="COMPUTERS":C\$=
"NEWSLETTERS"
20 LET RA=346712:RB=459321:RC=376872
30 LET EA=176894:EB=584837:EC=402195
40 PRINT A\$,B\$,C\$
50 PRINT RA,RB,RC
60 PRINT EA,EB,EC
70 LET PA=RA-EA:PB=RB-EB:PC=RC-EC
80 PRINT PA,PB,PC
90 PRINT
100 PRINT "TOTAL PROFIT EQUALS",PA+PB+PB

Section 2.4 (page 41)

1. 10 LET S=0
20 FOR J=1 TO 25
30 LET S=S+J^2
40 NEXT J
50 PRINT S
60 END
2. 10 LET S=0
20 FOR J=0 TO 10
30 LET S=S+(1/2)^J
40 NEXT J
50 PRINT S
60 END

3. 10 LET S=0
20 FOR J=1 TO 10
30 LET S=S+J³
40 NEXT J
50 PRINT S
60 END
4. 10 LET S=0
20 FOR J=1 TO 100
30 LET S=S+1/J
40 NEXT J
50 PRINT S
60 END
5. 10 PRINT "N²", "N²", "N³"
20 FOR J=1 TO 12
30 PRINT J, J², J³
40 NEXT J
50 END
6. 10 PRINT "MONTH", "INTEREST", "BALANCE"
20 B=4000:P=125.33
30 FOR J=1 TO 12
40 LET I=.01B:REM I=THE INTEREST FOR MONTH
50 LET R=P-I:REM R=REDUCTION IN BALANCE FOR MONTH
60 LET B=B-R:REM NEW BALANCE
70 PRINT J,I,B
80 NEXT J
90 END
7. 10 PRINT "END OF YEAR", "BALANCE"
20 B=1000
30 FOR J=1 TO 15
40 B=B+1000+.10*B :REM ADD DEPOSIT AND INTEREST
50 PRINT J,B
60 NEXT J
70 END
8. 10 LET S=3.5E7: P=5.54E6
20 PRINT "END OF YEAR", "SALES", "PROFITS"
30 FOR J=1 TO 3
40 LET S=1.2*S: P=1.3*P
50 PRINT J,S,P
60 NEXT J
70 END

Section 2.6 (page 56)

1. 10 J=1
20 IF $J^2 > 45000$ THEN 100
30 PRINT J, J^2
40 J=J+1
50 GOTO 20
100 END
2. 10 PI=3.14159
20 R=1
30 IF $PI * R^2 \leq 5000$ THEN 40
35 GOTO 100
40 PRINT R, $PI * R^2$
50 R=R+1
60 GOTO 30
100 END
3. 10 PRINT "SIDE OF CUBE","VOLUME"
20 S=1
30 $V = S^3$
40 IF $V < 175000$ THEN 50
45 GOTO 100
50 PRINT S,V
60 S=S+1
70 GOTO 30
100 END
4. 10 FOR J=1 TO 10 : REM LOOP TO GIVE 10 PROBLEMS
20 INPUT "TYPE TWO 2-DIGIT NUMBERS"; A,B
30 INPUT "WHAT IS THEIR PRODUCT";C
40 IF $A * B = C$ THEN 200
50 PRINT "SORRY. THE CORRECT ANSWER IS ", $A * B$
60 GOTO 500 : REM GO TO THE NEXT PROBLEM
200 PRINT "YOUR ANSWER IS CORRECT! CONGRATULATIONS"
210 LET R=R+1 : REM INCREASE SCORE BY 1
220 GOTO 500 : REM GO TO THE NEXT PROBLEM
500 NEXT J
600 PRINT "YOUR SCORE IS ",R," CORRECT OUT OF 10"
700 PRINT "TO TRY AGAIN, TYPE RUN"
800 END
5. 10 FOR J=1 TO 10 : REM LOOP TO GIVE 10 PROBLEMS
15 PRINT "CHOOSE OPERATION TO BE TESTED:"
16 PRINT "ADDITION (A), SUBTRACTION (S), OR MULTIPLICATION (M)"

```

17 INPUT A$
20 INPUT "TYPE TWO 2-DIGIT NUMBERS"; A,B
21 IF A$="A" THEN 30
22 IF A$="S" THEN 130
23 IF A$="M" THEN 230
30 INPUT "WHAT IS THEIR SUM ";C
40 IF A+B=C THEN 400
50 PRINT "SORRY. THE CORRECT ANSWER IS",A+B
60 GOTO 500 : REM GO TO THE NEXT PROBLEM
130 INPUT "WHAT IS THEIR DIFFERENCE";C
140 IF A-B=C THEN 400
150 PRINT "SORRY. THE CORRECT ANSWER IS ",A-B
160 GOTO 500 : REM GO TO THE NEXT PROBLEM
230 INPUT "WHAT IS THEIR PRODUCT ";C
240 IF A*B=C THEN 400
250 PRINT "SORRY. THE CORRECT ANSWER IS",A*B
260 GOTO 500 : REM GO TO THE NEXT PROBLEM
400 PRINT "YOUR ANSWER IS CORRECT! CONGRATULATIONS"
410 LET R=R+1 : REM INCREASE SCORE BY 1
420 GOTO 500 : REM TO TO THE NEXT PROBLEM
500 NEXT J
600 PRINT "YOUR SCORE IS ";R;" CORRECT OUT OF 10"
700 PRINT "TO TRY AGAIN, TYPE RUN"
800 END

```

6. See Exercise 8.

7. See Exercise 9.

```

8. 10 INPUT "NUMBER OF NUMBERS";N
20 FOR J=1 TO N
30 INPUT A
40 IF J=1 THEN B=A
50 IF A>B THEN B=A
60 NEXT J
70 PRINT "THE LARGEST NUMBER INPUT IS",B
80 END

```

9. Replace line 50 in Exercise 8. by:

```

50 IF A<B THEN B=A
Change "Largest" to "Smallest"

```

```

10. 10 A0=5782:A1=6548:B0=4811:B1=6129:C0=3865:C1=4270
20 D0=7950:D1=8137:E0=4781:E1=4248:F0=6598:F1=7048
30 FOR J=1 TO 6
40 IF J=1 THEN A=A0:B=A1
50 IF J=2 THEN A=B0:B=B1

```



```

60 IF J=3 THEN A=C0:B=C1
70 IF J=4 THEN A=D0:B=D1
80 IF J=5 THEN A=E0:B=E1
90 IF J=6 THEN A=F0:B=F1
100 I=B-A
110 IF I>0 THEN PRINT "CITY ",J," HAD AN INCREASE OF", I
120 GOTO 200
130 IF I<0 THEN PRINT "CITY ",J," HAD A DECREASE OF",A-B
140 GOTO 300
200 IF I>500 THEN PRINT "CITY ",J," MORE THAN 500 INCREASE"
300 NEXT J
400 END

```

11. 10 PRINT "THIS PROGRAM SIMULATES A CASH REGISTER"
 20 PRINT "AT THE QUESTION MARKS, TYPE IN THE"
 30 PRINT "PURCHASE AM'TS. TYPE -1 TO INDICATE":PRINT THE
 END OF ORDER."
 40 INPUT "TYPE 'Y' IF READY TO BEGIN"; A\$
 50 IF A\$="Y" THEN 60
 55 GOTO 10
 60 HOME
 70 PRINT "ITEM":INPUT A
 80 IF A=-1 THEN 200
 85 GOTO 90
 90 T=T+A: REM T IS THE RUNNING TOTAL
 100 GOTO 70
 200 PRINT "THE TOTAL IS", T
 210 S=.05*T:REM S=SALES TAX
 220 PRINT "SALES TAX", S
 230 PRINT "TOTAL DUE", S+T
 240 INPUT "PAYMENT GIVEN",P
 250 PRINT "CHANGE DUE", P-(S+T)
 300 END
12. 10 INPUT "CASH ON HAND"; C1
 20 PRINT "INPUT ACCOUNTS EXPECTED TO BE ":PRINT " RE-
 CEIVED IN NEXT MONTH."
 30 PRINT "TO INDICATE END OF ACCOUNTS TYPE -1."
 40 INPUT "ACCOUNTS RECEIVABLE";A
 50 IF A=-1 THEN 100
 60 C2=C2+A:REM C2=RUNNING TOTAL OF ACCOUNTS RECEIV-
 ABLE
 70 GOTO 40
 100 PRINT "INPUT ACCOUNTS EXPECTED TO BE PAID IN NEXT
 MONTH."
 110 PRINT "TO INDICATE END OF ACCOUNTS TYPE -1."

```

120 INPUT "ACCOUNTS PAYABLE";A
130 IF A=-1 THEN 200
140 C3=C3+A:REM C3=RUNNING TOTAL OF ACCOUNTS PAY-
    ABLE
150 GOTO 120
200 PRINT "CASH ON HAND",,C1
220 PRINT "ACCOUNTS RECEIVABLE",C2
230 PRINT "ACCOUNTS PAYABLE",C3
240 PRINT "NET CASH FLOW",, C1+C2-C3
300 END

```

CHAPTER 3

Section 3.1 (page 69)

1. DIM A(5)
2. DIM A(2,3)
3. DIM A\$(3)
4. DIM A(3)
5. DIM A\$(4),B(4)
6. 10 DIM A\$(3),B(3,2),C\$(2)


```

20 PRINT, "RECEIPTS"
30 C$(1)="STORE #1":C$(2)="STORE #2"
50 B(1,1)=57385.48:B(1,2)=89485.45
60 B(2,1)=39485.98:B(2,2)=76485.49
70 B(3,1)=45467.21:B(3,2)=71494.25
100 PRINT C$(1),C$(2)
200 FOR J=1 TO 3
220 PRINT A$(J),B(J,1),B(J,2)
230 NEXT J
300 END

```
7. Add the instructions:


```

5 DIM D(2)
240 FOR J=1 TO 2
250 D(J)=B(1,J)+B(2,J)+B(3,J)
260 NEXT J
270 PRINT "TOTALS"
275 PRINT D(1),D(2)

```

8. Move the END to 400 and add the following instructions.
- ```

6 DIM E(3)
300 FOR J=1 TO 3
310 E(J)=B(J,1)+B(J,2)
320 NEXT J
330 PRINT
340 PRINT "PERIOD", "TOTAL SALES"
350 FOR J=1 TO 3
360 PRINT A$(J) , E(J)
370 NEXT J
400 END

```
9. 10 DIM A\$(4), B\$(5), C(5,4)
- ```

20 A$(1)="STORE #1":A$(2)="STORE #2": A$(3)="STORE #3"
21 A$(4)="STORE #4"
30 B$(1)="REFRIG.":B$(2)="STOVE":B$(3)="AIR COND."
40 B$(4)="VACUUM": B$(5)="DISPOSAL"
50 PRINT "INPUT THE CURRENT INVENTORY"
60 FOR J=1 TO 4
70 PRINT A$(J)
80 PRINT
90 FOR I=1 TO 5
100 PRINT B$(I)
110 INPUT C(I,J)
120 NEXT I
130 NEXT J
200 REM REST OF PROGRAM IS FOR INVENTORY UPDATE
210 PRINT "CHOOSE ONE OF THE FOLLOWING"
220 PRINT "RECORD SHIPMENTS(R)"
230 PRINT "DISPLAY CURRENT INVENTORY(D)"
240 INPUT "TYPE R OR D";D$
250 IF D$="R" THEN 300
260 IF D$="D" THEN 600
270 HOME:GOTO 200
300 HOME
310 PRINT "RECORD SHIPMENT"
320 INPUT "TYPE STORE#(1-4)";J
330 PRINT "ITEM SHIPPED"
340 PRINT "REFRIG=1,STOVE=2,AIR COND.=3,
      VACUUM=4,DISPOSAL=5"
350 INPUT I
360 INPUT "NUMBER SHIPPED";S
370 B(I,J)=B(I,J)-S
380 GOTO 200

```

```

600 HOME
620 FOR J=1 TO 5
630 FOR I=1 TO 4
640 PRINT A$(I),B(I,J)
650 NEXT I
660 NEXT J
670 GOTO 200
1000 END

```

Note that this program is really an infinite loop. For this type of program this is a good idea. You don't want to accidentally end the program thereby erasing the current inventory figures! End this program using the CTRL C key combination.

Section 3.2 (page 75)

1. $A(1)=2, A(2)=4, A(3)=6, A(4)=8, A(5)=10, A(6)=12, A(7)=14, A(7)=16, A(8)=18, A(9)=20$
2. $A(0)=1.1, A(1)=3.3, A(2)=5.5, A(3)=7.7, B(0)=2.2, B(1)=4.4, B(2)=6.6, B(3)=8.8$
3. $A(0)=1, A(1)=2, A(2)=3, A(3)=4, B\$(0)=""A"", B\$(1)=""B"", B\$(2)=""C"", B\$(3)=""D""$
4. $A(0)=1, B(0)=2, A(1)=3, B(1)=4, A(2)=1, B(2)=2, A(3)=3, B(3)=4$
5. $A(1,1)=1, A(1,2)=2, A(1,3)=3, A(1,4)=4, A(2,1)=5, A(2,2)=6, A(2,3)=7, A(2,4)=8, A(3,1)=9, A(3,2)=10, A(3,3)=11, A(3,4)=12$
6. $A(1,1)=1, A(2,1)=2, A(3,1)=3, A(1,2)=4, A(2,2)=5, A(3,2)=6, A(1,3)=7, A(2,3)=8, A(3,3)=9, A(1,4)=10, A(2,4)=11, A(3,4)=12$
7. Out of DATA in 30
8. Type Mismatch in 30. (Attempt to set numeric variable equal to string.) Out of DATA in 30.
9. Set F(J) equal to the Federal withholding for employee J, N(J) = the net pay, and add the following lines.


```

280 PRINT "EMPLOYEE", "WITHHOLDING", "NET PAY"
290 FOR J=1 TO 5
300 IF D(J)<=200 THEN F(J)=0
310 IF D(J)<=210 THEN F(J)=29.10
320 IF D(J)<=220 THEN F(J)=31.20

```

```

330 IF D(J)<=230 THEN F(J)=33.80
340 IF D(J)<=240 THEN F(J)=36.40
350 IF D(J)<=250 THEN F(J)=39.00
360 IF D(J)<=260 THEN F(J)=41.60
370 IF D(J)<=270 THEN F(J)=44.20
380 IF D(J)<=280 THEN F(J)=46.80
390 IF D(J)<=290 THEN F(J)=49.40
400 IF D(J)<=300 THEN F(J)=52.10
410 IF D(J)<=310 THEN F(J)=55.10
420 IF D(J)<=320 THEN F(J)=58.10
430 IF D(J)<=330 THEN F(J)=61.10
440 IF D(J)<=340 THEN F(J)=64.10
450 IF D(J)<=350 THEN F(J)=67.10
500 N(J)=D(J)-E(J)-F(J)
600 PRINT B$(J),F(J),N(J)
700 NEXT J

```

10. 5 DIM A(25)
10 DATA 10,10,9,9,8,11,15,18,20,25,31,35,38,39,40,40,42,38
20 DATA 33,27,22,18,15,12
30 FOR J=0 TO 23
40 READ A(J)
50 S=S+A(J)
60 NEXT J
70 PRINT "AVERAGE 24 HOUR TEMP.", S/24
100 PRINT "TO FIND THE TEMPERATURE AT ANY PARTICULAR
HOUR"
110 PRINT "TYPE THE HOUR IN 24-HOUR NOTATION: 0-12=AM"
120 PRINT "13-24=PM"
130 PRINT "TO END THE PROGRAM, TYPE 25"
140 INPUT "DESIRED HOUR";A
150 IF A=25 THEN 200
160 PRINT "THE QUERIED TEMPERATURE WAS ";A(J);" DEGREES"
170 GOTO 100
200 END

Section 3.3 (page 81)

1. 10 PRINT "THE VALUE OF X IS ";5.378
20 END
2. 10 PRINT "THE VALUE OF X IS";TAB(22) 5.378
20 END

3. 10 PRINT "DATE";TAB(6) "QTY";TAB(10) "@";TAB(12) "COST";
20 PRINT TAB(17) "DISCOUNT";TAB(26) "COST"
30 END
4. 10 X=6.753:Y=15.111:Z=111.850:W=6.702
20 PRINT SPC(7-LEN(STR\$(X)));X
30 PRINT SPC(7-LEN(STR\$(Y)));Y
40 PRINT SPC(7-LEN(STR\$(Z)));Z
50 PRINT SPC(7-LEN(STR\$(W)));W
60 PRINT "_____"
70 S=X+Y+Z+W
80 PRINT SPC(7-LEN(STR\$(S)));S
90 END
5. 10 X=12.82:Y=117.58:Z=5.87:W=.99
20 PRINT "\$";SPC(6-LEN(STR\$(X)));X
30 PRINT "\$";SPC(6-LEN(STR\$(Y)));Y
40 PRINT "\$";SPC(6-LEN(STR\$(Z)));Z
50 PRINT "\$";SPC(6-LEN(STR\$(W)));W
60 PRINT "-----"
70 PRINT "\$";SPC(6-LEN(STR\$(X+Y+Z+W)));X+Y+Z+W
80 END
6. 10 PRINT TAB(20) "DATE";TAB(30) "3/18/81"
20 PRINT
30 PRINT "Pay to the Order of Wildcatters, Inc."
50 PRINT
60 PRINT "The Sum of";TAB(14) "*****\$89,385.00"
9. 10 INPUT "NUMBER TO BE ROUNDED";X
20 PRINT INT(X+.5)
30 END
10. Modify the program of Exercise 11 of Section 2.6 (page 57) by substituting SPC(11-LEN(STR\$(X))) into the PRINT statements. This will allow your cash register to handle numbers up to \$999,999.99.

Section 3.4 (page 88)

1. 100*RND(1)
2. 100+RND(1)
3. INT(50*RND(1)+1)
4. INT(4+77*RND(1))

5. $2 * \text{INT}(25 * \text{RND}(1) + 1)$
6. $50 + 50 * \text{RND}(1)$
7. $3 * \text{INT}(9 * \text{RND}(1) + 1)$
8. $1 + 3 * \text{INT}(7 * \text{RND}(1) + 1)$
10. Add the following instructions:
 - 132 IF C(J) > A(J) THEN 135
 - 133 GOTO 140
 - 135 PRINT "BET INVALID: NOT ENOUGH CHIPS"
 - 137 C(J) = 0
 - 139 GOTO 120
11. Change line 132 in Exercise 10 to read:
 - 132 IF C(J) > A(J) + 100 THEN 135
 - 133 GOTO 140
12. 10 PRINT "CHOOSE OPERATION TO BE TESTED"
 20 PRINT "ADD(A), SUBTRACT(S), MULTIPLY(M)"
 30 INPUT A\$
 40 A = INT(10 * RND(1)); B = INT(10 * RND(1))
 50 IF A\$ = "A" THEN 100
 60 IF A\$ = "S" THEN 200
 70 IF A\$ = "M" THEN 300
 100 HOME
 110 PRINT "WHAT IS "; A; "+" ; B; "?"
 120 INPUT C
 130 D = A + B
 140 GOTO 400
 200 HOME
 210 PRINT "WHAT IS "; A; "-" ; B; "?"
 220 INPUT C
 230 D = A - B
 240 GOTO 400
 300 HOME
 310 PRINT "WHAT IS "; A; "X" ; B; "?"
 320 INPUT C
 330 D = A * B
 340 GOTO 400
 400 IF C = D THEN 410
 405 GOTO 420
 410 PRINT "YOUR ANSWER IS CORRECT"
 415 GOTO 430
 420 PRINT "INCORRECT. THE CORRECT ANSWER IS "; D
 430 INPUT "ANOTHER PROBLEM(Y/N)?"; B\$

```
440 IF B$="Y" THEN 10
450 END
```

13. Put your names in a series of DATA statements located in lines 1000–1010.

```
5 DIM A$(10)
10 FOR J=0 TO 9
20 READ A$(J)
30 NEXT J
40 FOR J=1 TO 4
50 PRINT A$(10*RND(1))
60 NEXT J
70 END
```

Section 3.5 (page 95)

1. 10 FOR J=.1 TO .5 STEP .1
 20 GOSUB 100
 30 PRINT X
 35 NEXT J
 40 END
 100 X=5*J²-3*J
 110 RETURN

2. 1000 C(J)=100*(B(J)-A(J))/A(J)

3. 2000 M=C(1)
 2010 FOR J=2 TO 6
 2020 IF C(J)>M THEN M=C(J)
 2030 NEXT J
 2040 K=1
 2050 IF M=C(K) THEN 2100
 2055 GOTO 2060
 2060 K=K+1
 2070 GOTO 2050
 2100 RETURN

5. Let $D(J) = 4$ mean that J bets on first 12, $D(J) = 5$ that J bets on second 12, $D(J) = 6$ that J bets on third 12. In all such bets $B(J)$ will be 0. Corresponding to the new values of $D(J)$, there will be three new subroutines, starting in lines 4000, 5000, and 6000, respectively. Modify lines 121 to 125 as follows:

```
121 PRINT "BET TYPE:1=NUMBER BET,2=EVEN,3=ODD,4=1st
12"'
```



```

122 PRINT "5=2nd 12, 6=3rd 12"
123 INPUT "BET TYPE(1-6)";D(J)
124 IF D(J)>1 THEN 125
125 GOTO 130
126 INPUT "AMOUNT";C(J)
127 GOTO 180

```

Replace lines 320–330 by:

```

320 ON D(J) GOSUB 1000,2000,3000,4000,5000,6000

```

Finally, here are the three new subroutines.

```

4000 FOR K=1 TO 12
4010 IF X=K THEN 4100
4015 GOTO 4020
4020 PRINT "PLAYER";J;"LOSES"
4030 A(J)=A(J)-C(J)
4050 RETURN
4100 PRINT "PLAYER ";J;" WINS ";2*C(J);" DOLLARS "
4110 A(J)=A(J)+2*C(J)
4120 RETURN

```

The subroutines in 5000 and 6000 are identical, except for the lines:

```

5000 FOR K=13 TO 24

```

```

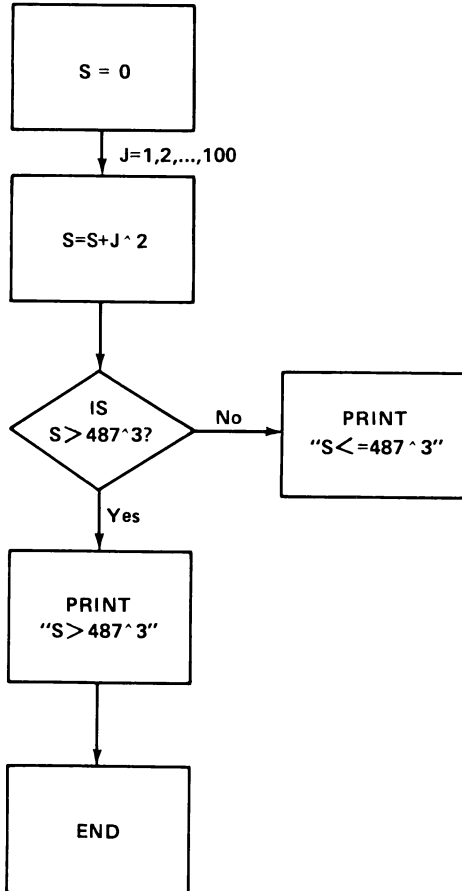
6000 FOR K=25 TO 36

```

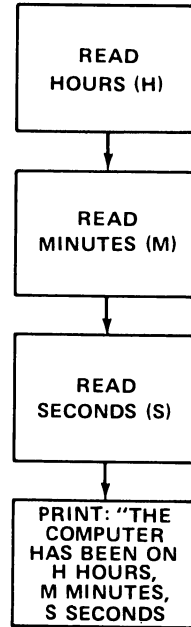
CHAPTER 4

Section 4.2 (page 102)

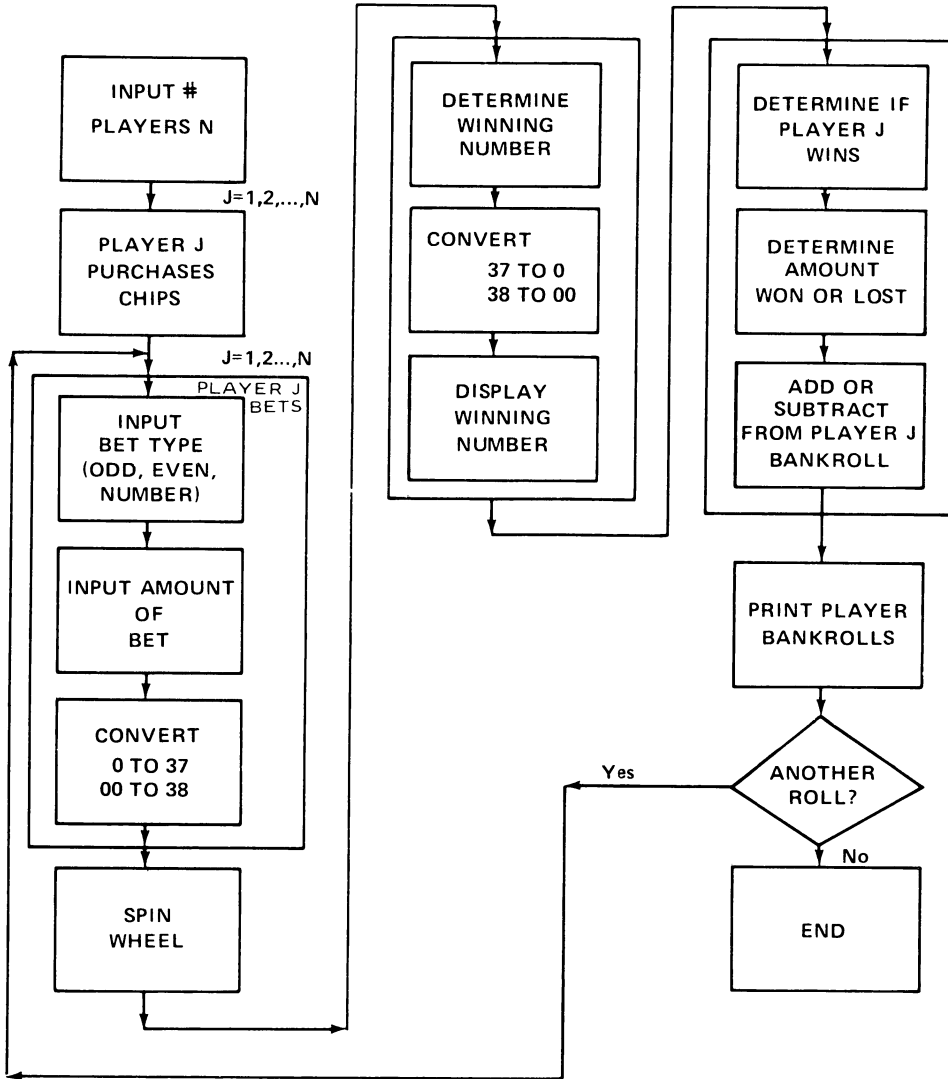
1.)



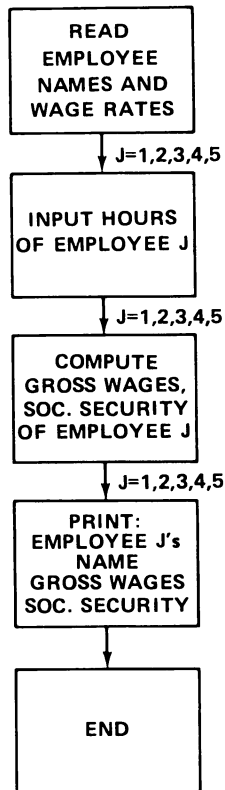
2.)



3.)



4.)

**Section 4.3 (page 106)**

1. Here are the errors:

TYPE MISMATCH in line 10: "0" should be 0

line 30: J(2 should be J²

line 80: NXT T should be NEXT T

line 90: should be deleted

line 100: ST should be S*T

line 110 quotes around "THE ANSWER IS"

2. line 30 should read: PRINT "THE FIRST N EQUALS",N
Need line 40: GOTO 200**CHAPTER 5****Section 5.3 (page 118)**1. a. 10 S=0
20 FOR J=1 TO 50

```

30 S=S+J^2
40 NEXT J
50 PRINT S
60 END

```

b. Type SAVE SQUARES

2. Type LOAD "SQUARES"
3. Type DELETE "SQUARES"

Section 5.4 (page 125)

1.

```

10 DATA 5.7,11.4,123,485,49
20 FOR J=1 TO 5
30 READ A(J)
40 NEXT J
50 PRINT CHR$(4);"OPEN NUMBERS"
55 PRINT CHR$(4);"WRITE NUMBERS"
60 FOR J=1 TO 5
70 PRINT A(J)
80 NEXT J
90 PRINT CHR$(4);"CLOSE NUMBERS"
100 END

```
2.

```

10 PRINT CHR$(4);"OPEN NUMBERS"
15 PRINT CHR$(4);"READ NUMBERS"
20 FOR J=1 TO 5
30 INPUT A(J)
40 NEXT J
50 PRINT CHR$(4); "CLOSE NUMBERS"
60 FOR J=1 TO 5
70 PRINT A(J)
80 NEXT J
100 END

```
3.

```

10 PRINT CHR$(4); "APPEND NUMBERS"
15 PRINT CHR$(4);"WRITE NUMBERS"
20 DATA 5, 78, 4.79, -1.27
30 FOR J=1 TO 4
40 READ A
45 PRINT A
50 NEXT J
60 PRINT CHR$(4);"CLOSE NUMBERS"
70 END

```
4.

```

5 PRINT CHR$(4); "OPEN NUMBERS"
10 PRINT CHR$(4); "READ NUMBERS"

```

```

20 FOR J=1 TO 9
30 INPUT A(J)
40 NEXT J
50 PRINT CHR$(4); "CLOSE NUMBERS"
60 FOR J=1 TO 9
70 PRINT A(J)
80 NEXT J
100 END

```

```

5. 10 DIM A(100),B$(100),C$(100),D(100),E$(100)
20 PRINT "TYPE CHECK DATA ITEMS REQUESTED."
30 PRINT "FOLLOW EACH ITEM BY A CARRIAGE RETURN."
40 J=J+1
50 INPUT "CHECK #";A(J)
60 INPUT "DATE";B$(J)
70 INPUT "PAYEE";C$(J)
80 INPUT "AMOUNT(NO $)";D(J)
90 INPUT "EXPLANATION";E$(J)
100 INPUT "ANOTHER CHECK(Y/N)";F$
120 HOME
130 IF F$="Y" THEN 20
200 PRINT CHR$(4); "OPEN CHECKS"
210 PRINT CHR$(4); "WRITE CHECKS"
220 FOR M=1 TO J
230 PRINT A(M):? B$(M):?C$(M):?D(M):?E$(M)
240 NEXT M
250 PRINT CHR$(4); "CLOSE CHECKS"
1000 END

```

```

6. 10 PRINT CHR$(4); "OPEN CHECKS"
15 PRINT CHR$(4); "READ CHECKS"
20 ONERR GOTO 100
30 J=1
40 INPUT A(J),B$(J),C$(J),D(J),E$(J)
50 J=J+1
60 GOTO 40
100 PRINT CHR$(4); "CLOSE CHECKS"
110 S=0
120 FOR M=1 TO J
130 S=S+D(M)
140 NEXT M
150 PRINT "TOTAL OF CHECKS IS",S
1000 END

```

CHAPTER 6**Section 6.1 (page 132)**

1. 5 GR:COLOR=1
10 HLIN 0,39 AT 18
2. 5 GR: COLOR=1
10 VLIN 0,39 AT 17
3. 5 GR:COLOR=1
10 HLINE 0,39 AT 19
20 VLIN 0,39 AT 19
4. 5 GR:COLOR=1
10 HLIN 0,39 AT 13
20 HLIN 0,39 AT 26
30 VLIN 0,39 AT 13
40 VLIN 0,39 AT 26
5. 5 GR: COLOR=1
10 VLIN 1,24 AT 30
20 VLIN 1,24 AT 31
6. 5 GR: COLOR=1
10 FOR J=0 TO 39
20 PLOT J,J
30 NEXT J
40 END
7. 5 GR: COLOR=1
10 FOR J=0 TO 39
20 PLOT J,12
30 NEXT J
40 FOR K=1 TO 3
50 PLOT 10*K,13
60 PLOT 10*K,11
70 NEXT K
80 END
8. 5 GR: COLOR=1
10 FOR J=0 TO 39
20 PLOT 19,J
30 NEXT J
40 FOR J=0 TO 4
50 PLOT 18,8*J
55 PLOT 20,8*J
60 NEXT J
70 END

Section 6.3 (page 135)

2. The entire screen will be white.

CHAPTER 7**Section 7.2 (page 145)**

3.


```

10 A$="15+48+97=160"
20 B$(1)=LEFT$(A$,2)
30 B$(2)=MID$(A$,4,2)
40 B$(3)=MID$(A$,7,2)
50 B$(4)=RIGHT$(A$,3)
60 FOR J=1 TO 4
70 B(J)=VAL(B$(J))
80 NEXT J
90 FOR J=1 TO 3
100 PRINT SPC(3-LEN(STR$(B(J))))); B(J)
110 NEXT J
120 PRINT " _"
130 PRINT SPC(3-LEN(STR$(B(J))))); B(4)
140 END
```
4.


```

10 A$="$6718.49": B$="$4801.96"
20 A1$=RIGHT$(A$,7):B1$= RIGHT$(B$,7)
30 A2=VAL(A1$): B2=VAL(B1$)
40 PRINT "$";SPC(9-LEN(A1$));A2
50 PRINT "$";SPC(9-LEN(B1$));B2
60 HTAB 2: PRINT"-----"
70 PRINT "$";SPC(9-LEN(STR$(A2+B2))); A2+B2
80 END
```

CHAPTER 9**Section 9.1 (page 177)**

1. 3.000000000
2. 2.370000000
3. 578,000.0000
4. 2.000000000

5. 3.000000000
6. -4.100000000
7. -4
8. 3500.685000
9. 217.600000
10. -5,940,000,000,000
11. 3.586950400
12. -2.34542383E10
13. -236,700,000,000,000,000,000
14. 4570000000000000000
15. 46.00000000
16. .5000000000
17. .6000000000
18. 1.600000000
19. .6666666667
20. 1.1966666667
21. 1749.99999
22. 3.33333333

Section 9.2 (page 179)

1. 10 PRINT (5.87+3.85-12.07)/11.98
20 END
2. 10 PRINT (15.1+11.9)^4/12.88
20 END
3. 10 PRINT (32485+9826)/(321.5-87.6^2)
20 END
4. 10 INPUT X%
20 IF X%<0 THEN X%=X%-1
30 PRINT X%
40 END
5. -5

242 ANSWERS TO SELECTED EXERCISES

6. 4
7. -11
8. 1.780000000
9. .0010000000000000
10. 32.65342000
11. 4.252345443E21
12. -1.234567890E-32
13. 3.283646493
14. -5.740000000

Section 9.3 (page 184)

1. 10 PRINT EXP(1.54)
20 END
2. 10 PRINT EXP(-2.376)
20 END
3. 10 PRINT LOG(58)
20 END
4. 10 PRINT LOG(9.75E-5)
20 END
5. 10 PRINT SIN(3.7)
20 END
6. 10 PRINT COS(.017453*45)
20 END
7. 10 PRINT ATN(1)
20 END
8. 10 PRINT TAN(.682)
20 END
9. 10 PRINT 57.29578*ATN(2)
20 END
10. 10 PRINT LOG(18.9)/LOG(10)
20 END
11. 10 FOR X=-5.0 TO 5.0 STEP .1
20 PRINT X, EXP(X)

- ```

30 NEXT X
40 END
12. 10 DATA 1.7, 3.1, 5.9, 7.8, 8.4, 10.1
 20 FOR J=1 TO 6
 30 READ X
 40 PRINT X, 3*X^(1/4)*LOG(5*X)+EXP(-1.8*X)*TAN(X)
 50 NEXT J
 60 END
15. 10 INPUT X
 20 PRINT "THE FRACTIONAL PART OF",X,"IS", X-INT(X)
 30 END

```

**Section 9.4 (page 185)**

- ```

1. 10 DEF FNA(X)=X^2-5*X
2. 10 DEF FNA(X)=1/X-3*X
3. 10 DEF FNA(X)=5*EXP(-2*X)
4. 10 DEF FNA(X)=X*LOG(X/2)
5. 10 DEF FNA(X)=TAN(X)/X
6. 10 DEF FNA(X)=COS(2*X)+1
7. 10 DEF FNA(X)=5*EXP(-2*X)
    20 FOR X=0 TO 10 STEP .1
    30 PRINT X, FNA(X)
    40 NEXT X
    50 END

```

Index

- Absolute value function, 183–184
- Advanced printing, 78
 - formatting numbers, 79–81
 - horizontal tabbing, 78–79
- Alphabet, of Applesoft BASIC, 15
- Apostrophe, use of, 15
- APPEND instruction, 120
- Apple II computer, features of, 3, 3n, 5, 5f, 6–7, 7f, 8
- Applesoft BASIC
 - alphabet, 15
 - displaying of, 1–2
 - vocabulary, 14
- Applesoft BASIC command(s), 42–43
 - deleting program lines, 44
 - listing a program, 43–44
 - saving a program, 45–46
- Applesoft BASIC constant(s), 15–16
- Applesoft BASIC language, *see* BASIC
- Applesoft BASIC program(s), 16
 - features of, 17–19
- Applesoft BASIC prompt, *see* Applesoft BASIC
- Applewriter, 153, 153n
- Appointment calendar
 - software for, 205
 - use of real-time clock for, 159–160
- Arithmetic operation(s), 15
 - numeric constants and, 176
 - performance of, 17–19, 175
 - see also* Mathematics
- Array
 - containing string data, 65
 - dimensioning of, 66, 68, 69
 - numeric, 65
 - recalling of, 75
 - redimensioning of, 108
 - two-dimensional, 65
 - of variables, 64–66
- Art, *see* Computer art
- ASC instruction, 141
- ASCII character codes, 138, 138f–139f, 147
 - uses of, 140–142
- Assembler, 216
 - running of, 216
- Backspace key, 8
- Backup, 116
 - making of, 114–115
- ?BAD SUBSCRIPT ERROR, 108
- BASIC, 1, 11
 - development of, 11
 - see also* Applesoft BASIC
- BASIC command(s), *see* Applesoft BASIC command
- BASIC constant(s), *see* Applesoft BASIC constant
- BASIC language, 215
- BASIC prompt, *see* Applesoft BASIC
- BASIC programs, *see* Applesoft BASIC program(s)
- Baud, 212
- Baud rate, 211
- Beginners All-Purpose Symbolic Instructional Code, *see* BASIC
- Binary digit, 210

- Binary number, 210
- Bit(s), 211
 - defined, 210
 - parity, 211
- Blind target shoot game, 162, 162f, 163, 164f, 164, 165f, 165–166
- Boldface, use of word processor for, 152
- Book-keeping systems, software for, 205
- BREAK key, use of, 51–52
- Budget, home, VISICALC for, 197–205
- Buffer, defined, 211–212
- Bug, *see* Error
- Byte, defined, 210
 - see also* Apple II computer, Personal computer
- Computer age, 1
- Computer art, 134
 - devices for, 135
 - forms, 134–135
- Computer communications, 209
 - avoiding of errors, 211
 - devices for, 209–211
 - establishing of link, 211
 - to outside world, 213–214
 - protocol, 211–212
 - types of, 211
- Computer games, 157, 205
 - blind target shoot, 162–167
 - telling time, 157–162
 - tic tac toe, 167–173
- Computer graphics, 127
 - advanced, 212–213
 - computer art, 134–135
 - modes
 - high resolution, 134
 - low resolution, 128, 129f, 130f, 131–132
 - text, 128
- Computer language(s), 5
 - see also* BASIC
- Computing, early, 2
- Constant(s), 15
 - numeric, *see* Numeric constants
 - string, *see* String constants
 - see also* Applesoft BASIC constants
- CONT command, 36, 37
- Copy(ies), *see* Backup
- Cost of computers, 2
- CPU, use of, 4, 5
- CTRL C key, 36, 52, 105
- Current disk drive, 117
 - changing of, 117
 - use of, 117
- Cursor, 7
 - moving of, 98, 99–100, 163
- Cursor position, 8, 48
- Daisy wheel printer, 59, 212
- Damage to computer, 8, 12
- Data
 - copies of, 5
 - inputting, 70–75
 - storage of, 210
- Data file(s)
 - reading and writing of, 110
 - procedure for, 118–125
 - storage of, 110
 - uses of, 109
- Data item(s), 71, 110
 - reading of, 110
- Data statement(s)
 - data items in, 71, 72
 - errors in, 74
- C key, 36, 52
 - see also* CTRL C key
- Calculation(s), performance of, 17
- “CALCULATION DONE”, 47
- ? CAN'T CONTINUE ERROR, 108
- Cassette, 4
 - positioning of, 75
 - recalling array from, 75
 - saving a program, 45
- Cassette recorder, use of, 5
- CATALOG command, use of, 117–118
- Centering, use of word processor for, 152
- Central processing unit, *see* CPU
- Character codes, ASCII, 138, 138f–139f, 140–142, 147
- Children, program for, 122–125
- 6502 chip, 215
- Circuit board(s), 112, 146
- CLEAR command, use of, 68
- Clearing function, 8
- Clock, *see* Real-time clock
- CLOSE instruction, 119
- COBOL, 217
- Color(s), selecting of, 130, 131
- COLOR instruction, 131
- Comma, use of, 20, 78, 110, 121, 123
- Command(s), interpreting of, 12
 - see also* Specific type
- Command mode, 14, 52
- Commercially available programs, *see* Software
- Communications, *see* Computer communications
- Computer
 - connection to external devices, 209
 - cost of, 2
 - damage to, 8, 12
 - features of, 4–5, 11–12
 - early, 2
 - modern, 2
 - large, 211
 - “main-frame,” 12
 - starting of, 113–114
 - use, areas of, 1
 - views on, 1, 11

- form of, 70–71
 - rereading of, 74
 - use of, 71–72
 - for appointment calendar, 159
- Debugging, 12, 104
 - procedure for, the trace, 104–105
- Decision making, use of computer for, 47–51
- DEF FN instruction, 185
- DEL command, 44, 160
- Delay(s), creating of, use of loops for, 39–40
- DELETE command, 118
- Deleting program lines, 44
- Delimiters, 110
- DIM statement
 - inserting of, 69
 - use of, 66–67
- Dimension statement, see DIM statement
- Disk controller board, 112
- Disk controller card, insertion of, 111, 112
- Disk drive
 - description of, 112
 - insertion of diskette into, 112
 - installation of, 111, 112
 - use of, 111
- Disk operating system, see DOS
- Diskette(s), 4
 - care of, 112
 - DOS, 113, 114–115
 - floppy, 112
 - insertion of, 112
 - parts of, 111f
 - reading of, 112
 - reading and writing data files on, 118–125
 - saving a program on, 45
 - use of, 112–113
 - writing on, 112
- Diskette drive, 110
 - current, see Current diskette drive
 - function of, 5
- Diskette file(s), 110, 154
 - use of, 5
- Diskette filer, 113
- Diskette system, use, for first time, 114–115
- Division operation, 17, 22
- ? DIVISION BY ZERO ERROR, 107
- Do-it-yourself word processor, 153–156
- Document
 - draft version of, 154
 - saving of, 154
- DOS, 45, 118, 206
 - catalog, 117–118
 - current disk drives, 117
 - erasing files from, 118
 - features of, 116
 - reading of, 113–114
 - renaming a file, 118
 - saving and loading programs, 116–117
- DOS diskette, 113
 - copy of, 114–115
- Dot-matrix thermal printer, 59
- Doubly subscripted variable, 65
- Draft version(s), production of, 154
- Dry cleaners, simulation of, 189–192
- Duplicate copies, see Backup
- Editing, 154
 - of program lines, 97
 - process, 98
 - use of word processor for, 137–138
- “Electronic brain,” computer as, 1, 11
- Electronic pen, 135
- Electronic pencil, 213
- Electronics, of computer, 11
- END instruction, 12, 16, 17, 51, 71
- Erasing files, 118
- Error(s)
 - analysis of, 106
 - avoiding of, 122, 211
 - correcting of, 12, 17, 151, 154
 - see also Editing
 - finding of, see Debugging
 - see also Typing errors
- Error message(s), 8, 75, 106
 - examples, 107–108
 - printing of, 106
- ESC key (escape), 98
- ESCAPE sequences, 147
- EVEN bets, 90
 - subroutines corresponding to, 91
- Execute mode, 14
- Exponential format, use of, 16
- Exponential function(s), 179, 181–182
- Exponentiation, operation of, 21–22
- External device(s), see Peripheral device(s)
- File(s)
 - erasing of, 118
 - opening of, 118, 121
 - random access, 125
 - renaming of, 118
 - sequential, 125
 - see also Data file, Program file
- Financial planning program, see VISI-CALC
- Financial statement(s), preparing of, 80
- Floppy diskette(s), 112
- “Floppy disks,” 5
- Flow charting, 101, 101f, 102, 103f
 - rules, 102
- Flowchart, defined, 101
- FNF function, 185
- FOR statement, 33, 60
 - NEXT without, 108

Form letters, use of string manipulation
 for, 147–149
 Formatting numbers, 79–81
 ? FORMULA TOO COMPLEX ERROR, 108
 FORTRAN, 217
 Frustrations, see Programming frustra-
 tions

 Gambling, use of computer for, 83–88
 Games, see Computer games
 Global search and replace, use of word
 processor for, 152
 GOSUB statement, 89, 94
 RETURN statement without, 108
 GOTO instruction, 98, 105
 deleting of, 99–100
 uses of, 47, 48–51
 GR command, 128
 features of, 128
 Graphics, see Computer graphics
 Graphics block, 134
 Graphics pad, use of, 135
 Graphics tablet, adding of, 213
 Greatest integer function, 183–184

 Hard copy(ies), 4, 59
 production of, 213
 Hexadecimal system, 216
 HGR command, 128
 High resolution graphics mode, 127, 128
 features of, 134
 HOME, 8, 37
 Home budget, VISICALC for, 197–205
 Homework, data file for, 122–125
 Horizontal tabbing, 78–79

 Identification of program, 45
 IF statement, 47, 60
 use of, 48–51, 53, 55
 ?ILLEGAL QUANTITY ERROR, 108, 141
 Infinite loop(s)
 avoidance of, 60
 stopping of, 51, 52
 Information
 flow of, 113
 storage and retrieval, 112, 212
 Information networks, connection to,
 214
 Input device, see Keyboard
 INPUT operation, 53
 INPUT statement, 52–56, 70
 form of, 121
 use of, 119
 Input unit, function of, 4
 Inputting data, 70–75
 Instruction(s)
 acceptance of, 12
 addition of, 13
 interpreting of, 14
 repeating of, 89

 INT function, 183
 use of, 84
 INT statement, 80
 Integer number constant
 defined, 175
 storage of, 175
 Integer type of variable, 178
 Interface, defined, 209–210
 RS232-C Interface, 210, 211, 213
 Interfacing
 defined, 210
 microcomputer, 210
 Interpreter, Applesoft BASIC as, 217

 Justification, use of word processor for,
 153–154

 Keyboard, 4, 15, 113
 features of, 6–7, 7f

 LEFT instruction, 144, 145
 Legal variable names, 28–29
 LEN instruction, 80
 use of, 141
 LET statement, 25, 27, 29, 58, 67
 deletion of, 58
 Letters, see Form letters
 Light pen, 135
 Line(s)
 multiple statements on, 30
 spacing of, 20
 Line number(s), 17
 Line numbering, features of, 17
 LIST command, 13, 43, 44, 106
 variations of, 43–44
 LIST key, 99
 Listing a program, 43–44
 LOAD command, 45
 LOAD RETAIN command, 45
 Loading programs, 116–117
 LOG function, 179, 180, 181
 Logarithmic functions, 179, 181–182
 Loop(s), 83
 containment of, 60
 defined, 33
 execution of, 51
 indications of, 101, 102
 performance of, 161
 uses of, 34–35, 55
 to create delays, 39–40
 see also Infinite loops
 Low resolution graphics mode, 127, 128
 color selection for, 128, 130
 line drawing with, 131–132
 screen layout for, 128, 129f, 130f

 Machine code, 216
 Machine language, instructions for,
 215–216
 preparing of, 216

Main frame(s), 211
 Main-frame computer(s), 2
 Mathematical functions, 179
 evaluation, 180
 greatest integer, absolute value and related functions, 183–184
 logarithmic and exponential, 181–182
 powers, 182–183
 trigonometric, 180–181
 Memory, 137
 function of, 4
 out of, 108
 types of, 4–5
 Microminiaturization, 2
 Microcomputer, 213
 interfacing, 210
 use of, 137, 217
 in word processing, 137
 Microprocessor, 6502, 4
 MID instruction, 144, 145
 Mnemonics, 216
 Mode(s), 14
 command, 14
 execute, 14
 Modem, function of, 213
 Multiple statements, on single line, 30
 Multiplication operation, 17, 22

 Names, giving of, to numbers and words, 25–26, 27f, 27
 Negative X, use of, 84
 NEW command, 13, 14, 42, 46
 NEXT instruction, 50, 60
 ? NEXT WITHOUT FOR ERROR, 108
 NO-TRACE, 105
 Null string, 68
 Number
 formatting, 63, 79–81
 giving names to, 25–26, 27f, 27
 rounding, 38
 Numeric array(s), 65
 Numeric constant(s), 21, 110
 defined, 15
 integer, 175
 real, 176
 specification of, 16
 type of, determining of, 176
 Numeric data, 71
 conversion to string data, 144–145
 manipulating of, 15
 Numeric variable(s), setting of, 68
 Numerical output, 79–81

 Object code, 216
 ODD bets, 90
 subroutines corresponding to, 91
 ONERR GOTO statement, 122
 use of, 160

 OPEN the file command, 118
 Operating system, function of, 113
 ?OUT OF DATA ERROR, 108
 ?OUT OF MEMORY ERROR, 108
 Output, adapting to screen, 36
 see also Hard copy
 Output device(s), 4
 Output of programs, production, 59
 Output unit, function of, 4
 Outside world, connection to, 213–214
 ?OVERFLOW ERROR, 107

 Paper output, see Hard copy
 Parallel communications, 211
 Parenthesis, use of, 63
 Parity bit, 211
 PASCAL, 217
 Pen
 electronic, 135
 light, 135
 Pencil, electronic, 213
 Peripheral device(s), 113
 connection to, 209
 Personal computer(s)
 areas for use, 2–3
 features of, 2
 Personal computing, defined, 2
 Personnel department, data files for, 109, 110
 Picture, tracing of, 135
 PLOT command, 131, 163
 Point plotting, 134
 Pointer
 location of, 71–72
 resetting of, 74
 PRINT instruction, 12, 13, 16, 18, 19–20, 29, 39, 58, 59, 78, 119, 120, 140, 147
 tab command and, 78, 79
 typing of, additional lines for, 79
 use of, 80
 variables in, use of, 26
 Print item, 79
 Print position(s), 78
 Print zone(s), 20, 78, 79
 Printable characters, ASCII character codes for, 138f–139f
 Printer controls, 146–147
 Printer interface card, 59
 Printing, see Advanced printing
 Professionals, software for, 205
 Program
 changing of, 13
 commercially available, see Software
 copies of, 5
 identifying of, 45
 listing of, 43–44
 output, production of, 59
 remarks in, 30

- saving and loading of, 45–46, 116–117
 - setting of, 12–13
 - writing of, checklist for, 60–61
- Program execution, ending of, 106
- Program files, 110
- Program lines
 - deleting of, 44
 - editing of, 97–100
- Programming frustration, easing of, 97
 - editing program lines, 97–100
 - error messages, 106–108
 - errors and debugging, 104–105
 - flow charting, 101–103
- Quantity, illegal, 108
- Question mark, use of, 58
- Quotation marks, use of, 15, 71
- RAM, 4, 14, 43, 45, 113, 134, 137, 175, 176
 - erasure from, 5, 13, 46
 - features of, 5, 13
 - use of, 5
- Random access files, 125
- “Random access memory,” see RAM
- Random number(s), generation of, 84, 85
- Random number generator, 83, 189
- “Read only memory,” see ROM
- READ statement
 - errors in, 74
 - function of, 71, 72
- Read-write window, 112
- READY instruction, 162
- Real numeric constant
 - arithmetic with, 176–177
 - defined, 176
- Real-time clock, 157
 - reading of, 158
 - setting of, 158–161
- Real variable, 178
- RECALL instruction, use of, 75
- Recording tape, 5
- Rectangles, 135
- ?REDIM'D array, 108
- REENTER message, 53
- REM statement, 30
- Remarks in program, 30
- Renaming a file, 118
- Repetitive operation, performance of, 32–38
- REPT key, 98
- RESET key, 6
- RESTORE statement, 74
- RESUME statement, use of, 160
- RETURN command, 21, 44, 53, 89, 110, 118, 147, 148, 153, 154
- RETURN key, 8, 13, 13n, 36, 42, 44, 99, 114, 115, 142, 199, 200, 201, 203
 - function of, 118
- ?RETURN WITHOUT GOSUB ERROR, 108
- RIGHT instruction, 144, 145
- RND function, 83, 134, 189
 - use of, 83–84, 84–85
- ROM, 4, 113
 - features of, 5, 116
- RUN again command, 13
- RUN command, 13, 14, 21, 33, 43, 104, 105, 106
- Running the program, 12, 13
- SAVE command, 45, 116, 117
- SAVE RETAIN command, 45
- Saving a program, 45–46, 116–117
- Scientists, programming for
 - defining your own functions, 184–185
 - integer and real constants, 175–177
 - mathematic functions, 179–184
 - variable types, 177–179
- Screen, printing on, 59
- Scrolling, 8, 151
- Search
 - global, 152
 - visual, 151
- Semicolon, use of, 78, 79
- Sequential files, 125
- Serial communications, 211
- SGN function, 183
- Shape(s), defining and displaying of, 134
- Shape tables, 134
- Shortcuts, in typing programs, 58
- Simulation, computer-generated, 187–189
 - of dry cleaners, 189–192
- SIN function, 179, 180
- Single line, multiple statements on, 30
- Slot number, 112
 - indication of, 112
- Software
 - buying of, guidelines for, 205–207
 - VISICALC, 197, 197n, 198–205
- Source code, 216
- Space(s), 110
- Spelling correction, use of word processor for, 153
- SQR(X) function, 182–183
- Square root function, see SQR(X) function
- Starting the computer, 113–114
- STOP instruction, 75
 - use of, 37
- STORE instruction, 75
- STR, 80, 145
- String
 - dissecting of, 144
 - length of, computing of, 141
 - operations performed on, 142
 - quantity of characters in, 142
 - relations among, 142–144
- String array, 143
- String constant(s), 20, 110
 - defined, 15

- String data, 71
 - array containing, 65
 - conversion of numeric data to, 144–145
- String manipulation
 - ASCII character codes, 138, 138f–139f, 140–142
 - numeric data and, 144–145
- ?STRING TOO LONG ERROR, 108
- String value(s), 65
- String variable(s), 29
 - setting of, 68
- Subroutine(s)
 - assembling of, 91–94
 - defined, 89
 - use of, 89, 90f, 90–91
- Subscript(s), 63
 - bad, 108
 - use of word processor for, 152
- Subscripted variables
 - array of, 64–66
 - defined, 64
 - doubly, 65
 - use of, 63, 64
- Superscript(s), use of word processor for, 152
- Syntax error, 106, 107
- System unit, slots of, 112

- TAB command, 78, 79
- Tabbing, horizontal, 78–79
- Tabular data, working with, 63–70
- TAB function, 180, 181
- Telephone directory, data file for, 119–120, 121
- TEXT command, 128
- Text mode, 127, 128
 - features of, 128
- TH, 189, 191
- THEN statement, 47, 60
 - execution of, 47–48
 - uses of, 48–51, 53, 55
- THUNDERCLOCK PLUS, 157, 157n
- Tic tac toe game, 167, 168, 168f, 169f, 169–173
- Time, telling of, 157
 - reading real-time clock, 158
 - setting clock, 158–161
- Time-hours, see TH
- Time-minutes, see TM
- Time-sharing system, connection to, 214
- TM, 189, 191
- Trace feature, use of, 104–105
- Trace off, see No-trace
- Tracing, 135
- Trigonometric function, 179, 180–181
- TV screen, see Video display
- Two-dimensional array, 65
- Type designator, 178
- ?TYPE MISMATCH ERROR, 108

- Typewriter
 - compared to keyboard, 6, 7
 - use of microcomputers as, 137
- Typing errors, correction of, 8
- Typing programs, shortcuts, 58

- ?UNDEF'D FUNCTION ERROR, 107
- ?UNDEF'D STATEMENT ERROR, 107
- Underscoring, use of word processor for, 152

- VAL instruction, 145
- Value(s)
 - assigning of, to variables, 67, 70, 71, 72, 121
 - printing of, 78
- Variable(s), 25
 - definition of, 63
 - handling of, 27
 - insufficient supply, 63
 - subscripted, see Subscripted variables
 - types of, 177–178
 - use of, 26
 - value of, 25, 27
 - assigning to, 67, 70, 71, 72, 121
- Variable names, legal, 28–29
- Video display, 4, 113, 137
- Video display controller board, 112
- Video display worksheet, 135
- Video monitor, see Video display
- VISICALC, 197, 197n
 - application of, for home budget, 197–205
 - worksheet for, 199, 200f
- Visual search, 151
- Vocabulary, of computer, 11, 14

- Warehouse, data files for, 109
- "WINGSPAN," 71
- Word processing, 137–138
 - printer controls and form letters, 146–149
 - strings, 142–145
 - manipulating, 138–142
- Word processing systems, 205
- Word processor
 - Applewriter, 153, 153n
 - computer as, 150–153
 - defined, 137
 - do-it-yourself, 153–156
 - use in editing, 137–138
- Words
 - giving names to, 25–26, 27f, 27
 - printing of, 19–21
- Worksheet, video display, 135
- WRITE command, 110, 120
- Write protect notch, function of, 112
- Writing of programs, checklist for, 60–61

- Zero, division by, error, 107

BASIC for the APPLE II PROGRAMMING AND APPLICATIONS

Larry Joel Goldstein, PhD • Martin Goldstein

Now for the first time ever, a text specifically designed for novices, potential buyers, and existing owners of the Apple II. Here is the book that gives readers a thorough yet refreshingly informal introduction to programming in BASIC computer language. This book contains all the information you need to know about the Apple II, from turning it on to programming it and why!

In this book you will find:

- a clear, concise outline of what a computer is and how it works
- a thorough introduction to BASIC language with helpful tips on easing programming frustrations
- immediate applications to business, graphics, games, and word processing
- comprehensive tables, charts, appendices, and much more.

CONTENTS

A First Look at Computers • Getting Started in Applesoft BASIC • More About Applesoft BASIC • Easing of Programming Frustrations • Your Computer as a File Cabinet • An Introduction to Computer Graphics • Word Processing • Computer Games • Programming for Scientists • Computer-Generated Simulations • Software You Can Buy • Some Other Applications of Your Computer • Where to Go From Here • Index

Larry Joel Goldstein is a Professor of Mathematics at the University of Maryland, College Park, Maryland. Involved in the design and application of computers since 1958, Goldstein is known for his clear, straightforward writing style, which makes learning technical subjects easy, fun, and informative for thousands of his readers.

Martin Goldstein is currently the President of Goldstein Associates, a consulting firm in West Palm Beach, Florida. His career was in chemical engineering.



0-89303-189-5