THE FIRST BOOK OF ADAM:

USING AND PROGRAMMING THE COLECO ADAM

que™

The First Book of ADAM

Using and Programming the Coleco ADAM

The First Book of ADAM
Using and Programming the Coleco ADAM

Pamela J. Roth

Que Corporation
Indianapolis

Library of Congress Catalog No.: 83-63253

ISBN 0-88022-063-5

88 87 86 85 84 8 7 6 5 4 3 2 1

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit number, the number of the book's printing. For example, a printing code of 83-4 shows that the fourth printing of the book occurred in 1983.

About the Author

Pamela J. Roth

Pamela Roth received her B.A. degree, *cum laude*, from the New College of Hofstra University in Hempstead, New York, in 1975. In 1977 she completed an M.S. in technical writing at the Rensselaer Polytechnic Institute, and in 1983 a J.D. degree from Boston's New England School of Law. After completing her B.A. degree, Ms. Roth served as associate editor for a photography magazine, where she was responsible for writing columns. As a free-lance technical writer, she has written articles for *Desktop Computing* and *The Word Guild Magazine*. Ms. Roth, who has worked as a systems analyst and documentation specialist, has documented several accounting software programs, including the NEC Astra Business Systems. She has lectured college students on writing about computers and led technical-writing workshops for data processing professionals.

Table of Contents

Illustrations

Figures

Preface

This book is for anyone who wants to learn about the two versions of the Coleco ADAM:

- Family Computer Module—designed to be attached to a ColecoVision Video Game System
- Family Computer System—designed to be a complete unit for purchasers who don't already have ColecoVision

Written so that someone with neither computer nor programming experience can learn the basics about programming on ADAM, this book covers the machine and its components, computer terminology, and programming with SmartBASIC. To aid the computer novice, the text defines terms the first time they appear. They are defined also in the book's glossary.

The chapters that follow also provide instructions for creating simple programs in SmartBASIC, an Applesoft-like programming language. Included also are brief descriptions of other software packages that are either now or will soon be available for ADAM.

For detailed information about SmartWRITER, the word processor included with ADAM, refer to the next book in this series, *The Second Book of ADAM: Using SmartWRITER.*

How to Use This Book

You may want to use a combination of the following approaches:

- Read the book cover to cover to learn how to program in SmartBASIC on your ADAM.
- Read the book to decide whether or not you want to purchase your own ADAM.
- Read only the chapters, sections, and paragraphs that answer specific questions. (Refer to the Index for help in locating what you need.)
- Read the Glossary to get an overview of terms used in the book and in the computer industry.

Throughout this book, as you learn how to program on your ADAM, you will also gain insight into programming in general, the computer industry, and other home computers.

Acknowledgments

The author wishes to thank the following people for their time, effort, support, advice, and information.

Richard L. Roth, executive vice president of InfoSoft in Norwalk, Connecticut, and lead consultant for Coleco's SmartWRITER word processing

Beverly Darwent, systems consultant at CPU Computer Center, Salem, New Hampshire

Jim Russell, manager of Markline Store, Waltham, Massachusetts

Pete Mpontsikaris, innocent bystander

I also wish to thank my Fortune 32:16 and Fortune:Word word processing for good behavior.

Trademark Acknowledgments

ADAM is a trademark of Coleco Industries, Inc.
Apple Source-Code is trademark of Apple Computer, Inc.
Applesoft is a registered trademark of Apple Computer, Inc.
ATARI is a registered trademark of Atari, Inc.
BUCK ROGERS is a trademark of the Dille Family Trust
ColecoVision is a registered trademark of Coleco Industries, Inc.
COLORFORMS is a registered trademark of Colorforms
COMMODORE is a registered trademark of Commodore Business Machines, Inc.
DONKEY KONG is a trademark of Nintendo of America, Inc.
DONKEY KONG JUNIOR is a trademark of Nintendo of America, Inc.
Dr. Seuss© copyrighted 1983 by Dr. Seuss. All rights reserved
IBM is a registered trademark of International Business Machines Corporation
Logo™ by Seymour Papert
PLANET OF ZOOM is a trademark of Sega Enterprises, Inc.
SLITHER is a trademark of Century II
SmartBASIC is a trademark of Coleco Industries, Inc.
SmartFILER is a trademark of Coleco Industries, Inc.
SMARTKEY is a trademark of Coleco Industries, Inc.
SmartLOGO is a trademark of Coleco Industries, Inc.
SmartWRITER is a trademark of Coleco Industries, Inc.
SMURF™ copyrighted 1983 by Peyo and licensed by Wallace Berrie and Co.
TEXAS INSTRUMENTS 99/4A is a registered trademark of Texas Instruments, Inc.
TIME PILOT is a trademark of Konami Industry Co., Ltd.
ZAXXON, TURBO, and SUBROC are trademarks of Sega Enterprises, Inc.

Introduction

Welcome to *The First Book of ADAM*. Before you begin, let's take a look at what is covered.

Chapter 1 covers information about home and personal computers, offering insight into Coleco's intent as it designed ADAM.

Chapter 2 describes the two versions of ADAM: the Family Computer Module, the ColecoVision add-on module; and the Family Computer System, the stand-alone unit for those who do not own a ColecoVision Game System.

Chapter 3 discusses programming on both the commercial and hobbyist levels, covering at length the steps involved in formulating program objectives and functional specifications.

The following chapters focus on programming in SmartBASIC:

Chapter 4 includes a tutorial introduction to programming in SmartBASIC; Chapter 5 discusses the SmartBASIC statements used to define input, processing, and output; Chapter 6 covers the design and appearance of the screen and printouts; Chapter 7 guides you through a sample program; Chapter 8 describes the SmartBASIC statements used to create low-resolution and high-resolution graphics; and Chapter 9 covers two more sample programs.

Chapter 10 discusses other ready-to-use software that is either currently or soon-to-be available for the ADAM, including SmartLOGO, SmartFILER, games, and educational programs.

Several appendixes offer additional assistance in using your ADAM: Appendix A covers error messages that may appear on the screen during programming; Appendix B is a summary of statements, functions, and commands; Appendix C is a list of ASCII code equivalents for use in graphics; and Appendix D is a list of reserved words to avoid when assigning names to variables.

A glossary of terms and an index of subjects are also included for your convenience.

1

Coleco's ADAM in the Home Computer Industry

Home Computers vs. Business Computers

The Coleco ADAM, advertised as a home computer, is sold through the same stores that handle other home computers, like Atari, TIMEX SINCLAIR, Commodore, and Texas Instruments. ADAM is the first home computer to include a daisy wheel printer and carry a retail price that is less than the price of most daisy wheel printers alone.

The general theory behind a home computer is that, like most other home appliances, the purchaser can take it home, read the directions, plug it in, and use it immediately to do what it was designed to do. In comparing a home computer to a home appliance, few will disagree that home computers are more complicated than most home appliances.

Home Computers

Because of their complexity, home computers offer a greater potential for return on investment than other home appliances. Designed to fulfill many needs, a home computer can be used for education, management, entertainment, typing, and even money making. Moreover, much of

This chapter was written with Richard L. Roth, executive vice president of InfoSoft in Norwalk, CT, and leading consultant for Coleco's SmartWRITER word processor.

this activity can be done with very little computer knowledge if users take advantage of the self-prompting, commercial software available for today's microcomputers.

As an educational tool, home computers can drill, or otherwise instruct, users on just about any popular subject, including most of those now offered in public education. From basic math to computer science, the list of educational programs for home computers is growing daily. In addition, of course, those who use home computers become computer literate and, if motivated, can learn to program.

As a management tool, home computers, with the appropriate programs, can help users keep track of everything from income taxes to family trees.

According to a 1982 survey of computer stores, entertainment programs now account for nearly 50 percent of the commercial programs sold. A home computer, particularly the ADAM with its many ColecoVision games, can provide endless hours of entertainment. All game programs aside, most users find the initial stages of mastering their computers a form of entertainment as well.

A home computer's versatility is most evident when an industrious user turns it into a money-making tool. As the computer slowly replaces the typewriter, so will word-processing services replace typing services. But most of all, those who dream of making money from their computers, dream of doing it through programming. All of us are familiar with at least one romantic tale of a hobbyist programmer who created a best-seller program in the basement of his home.

With an understanding of what home computers are designed to do, we have yet to cover the most persuasive argument for purchasing one. According to Future Computing, a market research firm dedicated exclusively to the personal computer market, 98 percent of American households, the same number as now have television sets, will have home computers by 1994. Why the rapid growth of computers in the home? Possibly the biggest reason that so many Americans will purchase home computers in the near future is the fear of being at a disadvantage in a computer-oriented world. Television commercials warn that children without home computers will be handicapped not only at school, where computers now abound, but in society in general, where computers now control everything from cash registers to automobiles.

For persons not motivated by fear, the result is the same. In fact, those motivated by a desire to be ahead, rather than a fear of being left behind, will probably be among the first to purchase home computers.

After a look at the uses of a home computer, you may be wondering how they differ from the uses of a business computer. How do the needs of the home computer user differ from those of the business user?

Business Computers

The professional who purchases a business computer, whether it is used in an office or a home, approaches the sale with concerns that differ from those of a buyer of a home computer.

Although the reasons for purchasing a business computer are similar to those for buying a home computer—fear of being left behind and desire to be at an advantage—the uses of a business computer are very different. While chief among the uses of a home computer are learning and entertainment, the chief uses of a business computer are information management, word processing, and financial planning.

Unlike the home consumer, who is investing in his personal future, the business consumer is making a financial investment. As a result, the business user cannot afford the luxury of learning about his computer at a leisurely pace. The business computer must be operating and paying for itself in a short amount of time.

To meet these time constraints, the professional must invest in much more than just a machine. His purchase usually includes extensive training to make sure the computer is productive from the start, and follow-up support to maintain that productivity.

It's not surprising, then, to learn that the cost of a business computer is many times that of a home computer. Since the business computer is more equipped than the home computer to handle data,the base price of the business computer is higher. Add to that the cost of training and support, and the business user pays about ten times what the home user pays to fulfill individual needs. While a basic home computer costs about $600, a basic business computer system costs approximately $6,000.

Where does ADAM fit? Is it just another home computer? Coleco thinks not. Realizing from the beginning that it was entering the race

late, Coleco attempted to produce a machine that would have more to offer than any other machine aimed at the home market.

The Evolution of Coleco's ADAM

For Coleco, entry into the home computer industry was a natural outgrowth of the home video game market. In 1983 Coleco's entrance wowed the industry. The production run of the ColecoVision Video Game System, the predecessor to the ADAM, was increased from 70,000 to 500,000 units. And they sold out. Coleco created ADAM with hopes for the same kind of success.

ADAM's Design

With people like Eric Bromley, designer of the successful hand-held Donkey Kong, on ADAM's design team, Coleco had every reason to approach the home computer market with confidence.

Bromley and others on the design team at Coleco, such as design engineer Rob Schenk and team leader Mike Levy, realized that moving from video games to computers was a large step—that there was much more involved and at stake in producing home computers than in designing, manufacturing, and distributing video games. Their research was extensive as they set out to answer several consumer product questions: What should the machine look like? What should it be able to do? What size should it be? How should it be packaged? How much wear and tear should it be able to withstand?

They decided to put together a compact, durable product that would replace something already in most homes (a typewriter), work with something in most homes (a television), and work with something else in many homes (a ColecoVision).

As a result of design decisions, all three of ADAM's components—the keyboard, memory console, and printer—are packaged in one box and fit comfortably on a standard office desk or in a computer cabinet. In terms of durability, the individual components can handle the same amount of wear and tear as other home appliances, such as stereos and tape decks.

The decision to make ADAM available in two versions, one that works with ColecoVision and one that works alone, is perhaps the wisest

decision made regarding ADAM's design. The Family Computer Module, which plugs into ColecoVision, is a tempting choice for the thousands of ColecoVision owners who may decide to purchase a computer. Being careful not to limit its market, Coleco also offers the Family Computer System, which includes its own game player and works without the support of ColecoVision. Both systems have the same capabilities.

ADAM's Software

In designing software for ADAM, Coleco concentrated on four major uses of a home computer: learning about computers and programming, typing, game playing, and learning in general.

To fulfill at least three of those uses, Coleco includes the following software along with ADAM: SmartBASIC, a programming language; SmartWRITER, a built-in word processor; and BUCK ROGERS PLANET OF ZOOM, a game. To meet the fourth need—learning in general—Coleco is preparing several educational software packages that will soon be available for ADAM.

ADAM's SmartBASIC

SmartBASIC is a programming language that Coleco says is source compatible with Applesoft BASIC, a BASIC language written for the Apple, one of the most popular microcomputers on the market. In theory, this statement means that you can type a program written in Applesoft BASIC into ADAM without having to make any changes. In reality, because Apple and ADAM differ, you will need to make some changes. For example, the width of a standard Apple screen is 40 characters, but the width of the ADAM screen is only 31 characters. You will have to change any instructions that tell the computer to print in positions 32 through 39 (assuming the first position on a line is labeled 0). For another example, consider the differences in printers. The SmartWRITER printer allows you to print up to 80 characters on a single line. However, depending on which printer you purchase for an Apple, you could print 132 or more characters on a line. Therefore, if an Applesoft program instructs the computer to output to a printer lines of more than 80 characters, you must change the code. (See Chapter 5, "The SmartBASIC Language," and Chapter 6, "Screen and Printout Design," for instructions on how to output data to the screen or the printer.)

ADAM's SmartWRITER

In its attempt to design a program that eases the transition from a typewriter to a word processor, Coleco has created a hybrid. SmartWRITER, ADAM's built-in word processor, is more a computerized typewriter than a word processor.

What makes SmartWRITER appear similar to a typewriter is its Electronic Typewriter mode, in which users can type directly from the keyboard to the printer. However, in order to edit any text written in this mode, the user must switch to the word processor mode. Here, the user has the option of editing on a full screen or in a two-line area called the roller, another effort to make SmartWRITER resemble a typewriter.

Early reactions to SmartWRITER were unfavorable. Several software reviewers, accustomed to microcomputer word processors, were confused by Coleco's typewriter-replacement approach and annoyed by the roller editing feature.

But Coleco did not design SmartWRITER for the experienced word-processor user. New users, with no preconceived notions, may find that SmartWRITER fulfills their typewriter needs.

ADAM's Games

ADAM's advanced graphics give Coleco's games an arcade quality that few other home computers can match. In his presentation at the Boston Computer Society's general meeting, Eric Bromley, head of Advanced Research and Design at Coleco, explained why: an average Atari game uses 16K of graphics, but games designed for ADAM, called Super Games, use up to 144K of graphics. As another selling point, Coleco includes the BUCK ROGERS PLANET OF ZOOM Super Game and accommodates all the ColecoVision games.

The Competition

How does ADAM stack up against the competition? Let's single out a few of the competitors—Timex/Sinclair, Texas Instruments, Atari, and Commodore—and establish a list of criteria by which to compare them. The criteria consist of items that make the machine desirable to a home consumer rather than to a business user, and they are defined as follows:

- Intended use—how the manufacturer thought the consumer would use the machine
- Storage—internal and mass storage
- Printer—whether one is included and the extent of compatible printers available
- Game device (arcade controller)—whether one is included and the quality of those available
- Display—the readability of text and impressiveness of graphics
- Word processing—whether a WP program is included, the quality of those available, and whether they take advantage of the computer's features
- Games—the quality of available games
- Expansion unit—whether it's included and, if not, whether it's available
- BASIC—whether the BASIC language included is comprehensive
- Educational software—the quality of educational software available
- Established base—the software, service, advice, and peripherals available from the manufacturer, dealers, software houses, and repair shops based on the number of machines sold and how well the machine was received
- Keyboard—the convenience of the keyboard

A Comparison of Home Computers

Table 1.1 compares the Timex/Sinclair, the TI-99/4, the Atari 400 and 800, the Commodore 64 and Coleco's ADAM against these established criteria.

Coleco ADAM v. IBM PCjr

As of this writing, IBM's home computer, code named Peanut, has finally emerged as PC*jr*. While it is premature to speculate about IBM's

Table 1.1
A Comparison of TIMEX SINCLAIR 1000, TI-99/4, Atari 400 and 800, Commodore 64, and ADAM

Criteria	*TIMEX SINCLAIR*	*TI-99/4*	*Atari 400/800*	*Commodore 64*	*COLECO ADAM*
Intended use	Teach BASIC programming	Educational	Entertainment; educational	Computer literacy; general educational	Typing; educational
Storage (internal)	2K up to 16K	16K graphics; 16K up to 64K cartridge	16K up to 48K	64K	16K graphics; 64K programs; expandable to 144K
Storage (mass)	Audio cassette manually operated; disk not available	Audio cassette manually operated; disk extra	Audio cassette manually operated; disk extra	Audio cassette manually operated; disk extra	Fully automatic tape; fast
Printer	Centronics parallel interface	Extra; wide range; mostly expensive	Extra; wide range	Extra; wide range	Included; slow; word quality
Game device	None	Extra; good	Included; some available; fair	Extra; good	Joystick included; good; other devices available
Display	Rough text; simple graphics	Good text; excellent graphics	Fair text; some graphics	Good text; good graphics	Fair text; good graphics
Word processing	Extra; minimum; does not use features	Extra; fair; does not use features	Extra; fair; does not use features	Extra; fair; does not use features	Included on a chip; uses features of the computer
Games	Fair to good	Excellent	Fair to good	Good	Arcade quality
Expansion unit	Extra	Extra; very good; expensive	None	None	Not needed; slots are built in
BASIC	Not comprehensive	Good graphics	Comprehensive	Good graphics	Applesoft equivalent; good
Educational software	Simple	Excellent	Fair to good	Fair to good	Expected quality equal to games
Established base	Large	Large	Some	Large	500,000 Video Game Systems; massive ADAM sales expected
Keyboard	Pressure sensitive	Good	400: poor 800: average	Good	Good; function keys and cursor keys

success, reactions from Arnold Greenberg, president of Coleco, indicate that Coleco is not intimidated by "Big Blue." At a 1983 Boston Computer Society general meeting, Greenberg welcomed IBM, explaining that IBM's entrance validates the home computer industry.

Part of Greenberg's confidence may be based on IBM's lack of experience with products aimed at a mass market. IBM may have a large share of other computer markets, but PC*jr* represents IBM's first major attempt at mass marketing.

Another reason, one addressed by Greenberg in Boston, is that Coleco doesn't view PC*jr* as a strong competitor. The vast home computer market consists of several different levels. IBM's PC*jr*, which retails for $669 and comes with only the system unit, keyboard, and cassette BASIC software, is aimed at the computer user who is looking for IBM compatibility. IBM is, no doubt, counting on the many potential home computer buyers who use IBM PCs at the office.

ADAM, on the other hand, whose $675 retail price includes a system unit, keyboard, printer, joysticks, and several software programs, is, according to Coleco, aimed at a different level of the home computer market. The ADAM's compatibility with ColecoVision, one of the leading computerized home entertainment systems, further serves to put the ADAM in a different league.

Needless to say, there will be purchasers, not necessarily bound by price or compatibility, who must seriously consider all the factors involved in purchasing a home computer. One of the many purposes of this book is to help those who have yet to make that important decision and want to learn more about the ADAM.

ADAM's Success in the Home Computer Market

If success could be accurately predicted, the computer industry would not be the competitive industry it is today. We can only look at the facts and leave the rest to history.

At the 1983 Consumer Electronics show in June of 1983, Coleco estimated it would have 500,000 ADAMs on store shelves prior to the Christmas season. The failure to meet that projected distribution quota undoubtedly cost Coleco some sales.

If, however, the home computer industry grows as rapidly over the next ten years as predicted, the industry has yet to experience its greatest growth spurt, and Coleco has plenty of time to recover. Despite these initial distribution problems, ADAM has all the other elements for success: a competitive price; prior success in the targeted market; compatibility with ColecoVision; bundled, easy-to-use software; and sturdy, compact components.

2

The Family Computer System and Module

ADAM is available in two versions: the Family Computer Module, which plugs into the ColecoVision Video Game System, and the Family Computer System, which also plays all ColecoVision game cartridges.

This chapter provides an overview of ADAM and a brief description of the following components:

- Keyboard, including SMARTKEYs, command keys, and cursor keypad—for typing information into ADAM
- Memory console—for processing information that you type at the keyboard and for saving (at your request) information so that you can edit or print it without retyping it
- Digital data packs—for storing up to 256,000 characters (bytes), or 125 double-spaced pages, of information
- Printer—for printing all or part of a document, program output, or program listing on letter-size (8 1/2" x 11") and legal-size (8 1/2" x 14") paper
- Your television or monitor—for displaying information
- Joysticks—for moving the cursor around while programming, word processing, or playing games

- Software—for word processing, programming, and entertainment, all included with ADAM

Overview

ADAM's most notable feature is its compactness. All the major components, except a TV, are assembled in one box: the keyboard, the memory console, and the printer. ADAM plugs into your color or black-and-white TV with the cables provided. All these components fit easily in a cabinet or on a standard-size office desk with room left over for notes, supplies, and elbows.

ADAM's second most notable feature is its durability. All its components can take the same amount of use or abuse as a home stereo. Hunt-and-peck typists will find the keyboard particularly durable as they pound away without fear of damaging it.

Another plus for ADAM is its quick response time. For example, the SmartBASIC language can be loaded from a digital data pack in less than 15 seconds. ADAM responds quickly to commands largely because its microprocessor, which controls the response time, is located on one of ADAM's chips. The microprocessor chip is linked to other chips by what Coleco calls ADAMNet. In other words, the chips work together to respond to your commands.

The entire SmartWRITER program, which includes the Electronic Typewriter and the SmartWRITER word processor, is also contained on one of these chips, allowing you to use SmartWRITER without loading the program from a digital data pack. You need to insert a digital data pack in the drive only when you want to save a document.

With the Electronic Typewriter, which appears on the screen and is ready to use as soon as ADAM is on, you can type directly from the keyboard to the printer; and you can control the format of your page. However, you cannot edit or store information. If you wish to edit or store something that you have typed on the Electronic Typewriter, you must switch to the SmartWRITER word processor.

You can switch to the word processor mode by pressing the ESCAPE/WP key. Now you can edit, store, and print information; but you cannot type directly from the keyboard to the printer.

ADAM's compatibility with the ColecoVision Video Game System and near compatibility with Applesoft BASIC give ADAM the added dimension of versatility. Both versions of ADAM accommodate all the ColecoVision game cartridges; and with slight modifications, programs written in Applesoft 8K BASIC source code can be typed into and run on ADAM.

Hardware Components

All the following hardware comes packaged with ADAM, except, of course, your own TV, which serves as ADAM's monitor (display screen).

Keyboard

The keyboard is the part of ADAM with which you will have the most contact. Through the keyboard you communicate with the other components of your computer. The keyboard consists of the following areas: standard keys, SMARTKEYs, command keys, and the cursor keys. Figure 2.1 shows the keyboard.

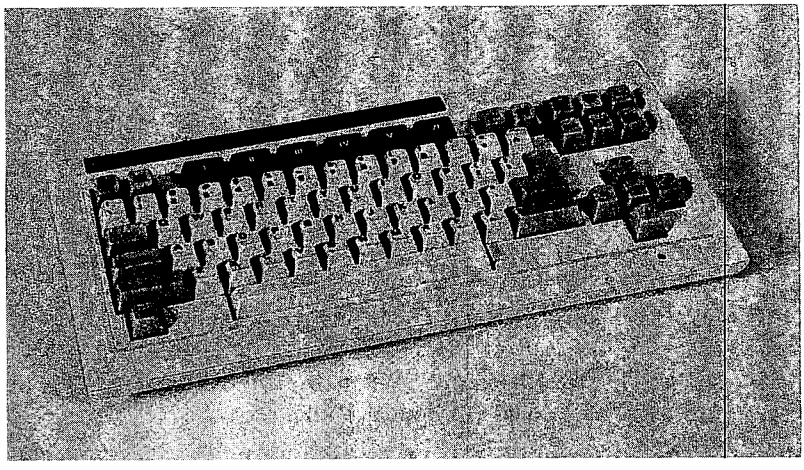


Figure 2.1
ADAM's Keyboard

Standard Keys

The white keys are the standard typing keys, making this part of the keyboard much the same as a typewriter keyboard. In addition, however, there are four extra white keys:

- Line/backslash key to the left of the exclamation point/1 key
- Twiddle (tilde)/caret key to the right of the equals/plus key
- Left bracket key to the right of the P key
- Right bracket key to the right of the left bracket key

The line/backslash and bracket keys are used for programming. The twiddle/caret key is used for accenting words.

The comma (,) and period (.) keys can be used only as unshifted or lower-case keys. If you try to use these keys with the SHIFT key, a less than (<) or greater than (>) sign will appear on the screen and be printed on the page.

All the characters on the white keys are on the daisy wheel. If you are curious, you can remove the daisy wheel from the printer and try to figure out where each character is located.

For more information about daisy wheels, see the section about the printer in this chapter.

SMARTKEYs and SMARTKEY LABELs

The six black keys labeled with Roman numerals are SMARTKEYs. SMARTKEYs are softkeys, which are keys whose functions can be changed according to the situation.

SMARTKEY LABELs, shown at the bottom of your screen when you are using the Electronic Typewriter or SmartWRITER word processor, indicate how to use each SMARTKEY. When the use of a SMARTKEY changes, its LABEL changes. When the SMARTKEYs are not in use, no SMARTKEY LABELs appear on the screen.

While you are typing, the SMARTKEY LABELs show the word-processing activities that you can perform by pressing the corresponding SMARTKEY: SMARTKEY I, SMARTKEY II, SMART-

KEY III, SMARTKEY IV, SMARTKEY V, or SMARTKEY VI. SMARTKEYs are used to execute such commands as clear, delete, and print, which are initiated by pressing the CLEAR, DELETE, or PRINT keys, respectively. For example, if you press DELETE, the SMARTKEY LABELs change, providing instructions for deleting text. The SMARTKEY LABELs that appear while you type text comprise the entry level. When you press a SMARTKEY, the SMARTKEY LABELs change. To return to the entry level, press the ESCAPE/WP key.

NOTE: To begin many word-processing activities, the entry-level SMARTKEY LABELs must be on the screen. Pressing the ESCAPE/WP key ends the WP activity in progress and returns the entry-level SMARTKEY LABELs to the screen. Before pressing the ESCAPE/WP key to return to the entry level, make sure you have completed the activity or want it to end before it is completed.

Entry-level SMARTKEY LABELs for the Electronic Typewriter appear on the screen as shown in Figure 2.2.

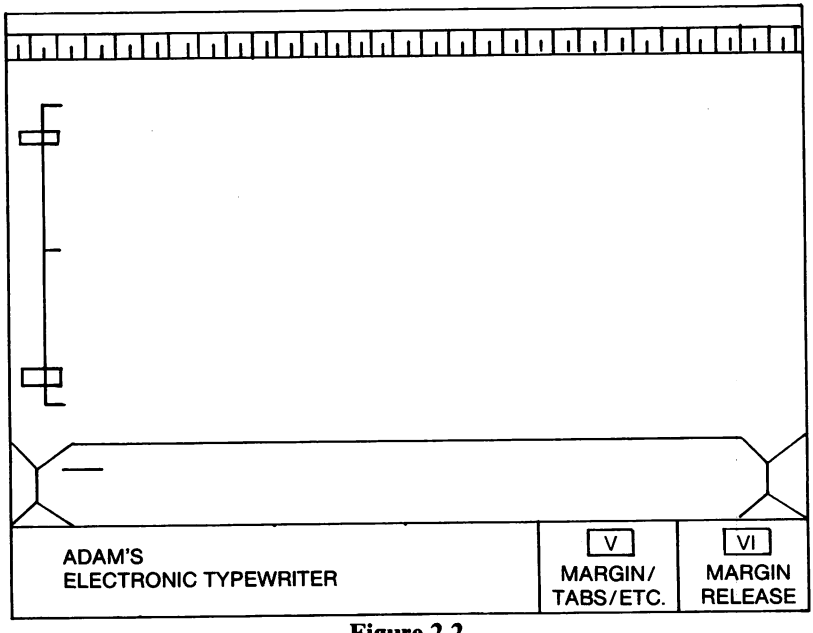


Figure 2.2
ADAM's Electronic Typewriter

Entry-level SMARTKEY LABELs for the SmartWRITER word processor appear on the screen as shown in Figure 2.3.

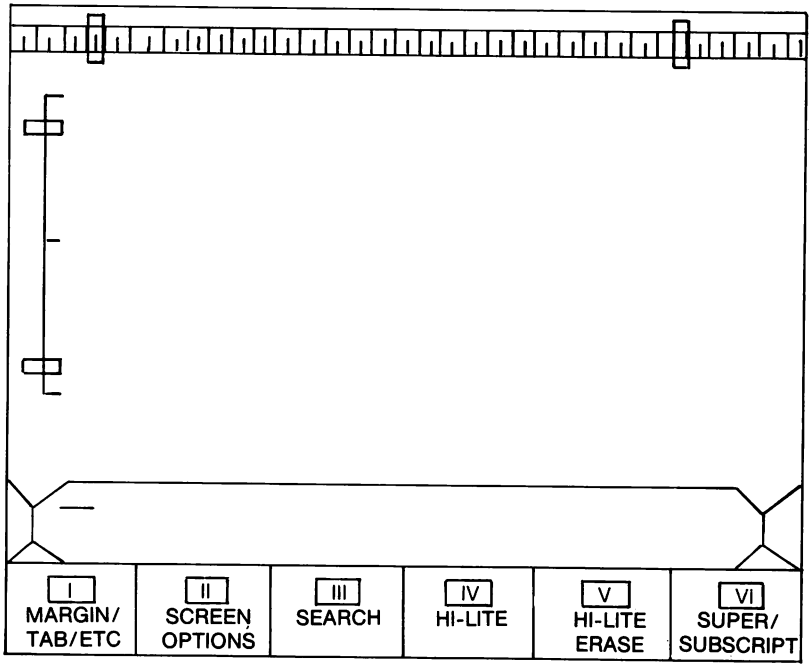


Figure 2.3
Entry-Level SMARTKEY LABELs
for SmartWRITER word processor

Command Keys

Sixteen of the gray keys are called command keys. The other five—the arrow keys and the Home key—are cursor keys, which form the cursor keypad. You are probably familiar with the TAB, SHIFT, LOCK, and RETURN command keys, commonly found on typewriters. However, you should read the descriptions of these keys carefully because, in some cases, on ADAM they function differently from the way they do on a typewriter.

The command keys and their uses while using SmartWRITER are listed in Table 2.1. As you read through it, find each key on the keyboard.

Table 2.1
ADAM's Command Keys While Using SmartWRITER

Key	*Description*
ESCAPE/WP	Changes from Electronic Typewriter mode to SmartWRITER mode. Once in SmartWRITER mode, this key is used to escape from a word-processing activity to the entry level of SMARTKEY LABELs and normal typing.
WILD CARD	This key has no function in the SmartWRITER word processor.
UNDO	Undoes the previous activity. For example, if you delete something, then realize you have deleted the wrong word, press the UNDO key to place the word where it was before you deleted it. The UNDO key works only on the activity immediately before you press it. So if you delete the wrong thing, you must press the UNDO key immediately to undo the deletion. Otherwise, the text is deleted forever. The UNDO key reverses the following activities: clear, delete, and replace.
BACKSPACE	Moves the cursor one position to the left, deleting any character in its path. This key should not be confused with the CURSOR-LEFT key, which backspaces without deleting.
MOVE/COPY	Indicates that you want to move or copy characters, words, or phrases from one part of a document to another. Pressing the MOVE/COPY key changes the SMARTKEY LABELs.
STORE/GET	Indicates that you want to store characters, words, phrases, or sentences on a digital data pack, or get the same from a digital data pack.

Table 2.1 *(continued)*

	Pressing the STORE/GET key changes the SMARTKEY LABELs, allowing you to select which digital data pack and which file to store or retrieve.
CLEAR	Indicates that you want to clear the screen or the workspace. Pressing the CLEAR key changes the SMARTKEY LABELs, allowing you to clear either the screen or the workspace.
INSERT	Indicates that you want to add a character, word, phrase, sentence, or page. You simply move the cursor to where you want new characters to appear, press the INSERT key to open space in the existing text, and type in the new text. You can also insert a page end E character and subscripted and superscripted text.
PRINT	Indicates that you want to print text. Pressing the PRINT key changes the SMARTKEY LABELs, allowing you to print only HI-LITED text, everything that appears on the screen, everything in the workspace, or everything from a particular file.
DELETE	Indicates that you want to delete a character, word, phrase, sentence, or paragraph. Pressing the DELETE key changes the SMARTKEY LABELs, allowing you to HI-LITE the text that you want to delete.
TAB	Indicates where text should be indented when you are typing a document. Pressing this key while typing or inserting indents a line to the first tab set after the cursor (position).
CONTROL	Functions only when programming the ADAM. It does not operate when you are using SmartWRITER.

Table 2.1 *(continued)*

RETURN	The RETURN key is used to leave a blank line in text and to end a paragraph. Unlike a typewriter, you do not need to press the RETURN key at the end of each line. The wraparound feature allows you keep typing without pressing the RETURN key at the end of each line.
SHIFT	Both SHIFT keys operate the same as the SHIFT keys on a typewriter.
LOCK	Operates the same as the LOCK key on a typewriter. It locks the SHIFT key so that only upper case letters appear on the screen or are printed on paper. Pressing the LOCK key again returns the standard typewriter keys to normal operation.

Cursor Keys

The cursor keys and their uses are described in Table 2.2. As you read through the table, find the keys on the keyboard. The four arrow keys and the HOME key that comprise the cursor keys are sometimes called the cursor keypad.

NOTE: The operation of the cursor keys with SmartWRITER depends on whether the screen is set for standard or moving-window format. Table 2.2 describes the operation during moving-window format. During standard format, the text scrolls in and out of the roller.

Table 2.2
Cursor Keys

Key	*Description*
CURSOR UP	Also referred to as CURSOR NORTH, this key, when pressed, moves the cursor up one position to the line above. If the cursor is at the top of the screen when you press the CURSOR UP key, the text scrolls.

Table 2.2 *(continued)*

CURSOR DOWN	Also referred to as CURSOR SOUTH, this key, when pressed, moves the cursor down one position to the line below. If the cursor is at the bottom of the screen when you press the CURSOR DOWN key, the text scrolls.
CURSOR LEFT	Also called CURSOR WEST, this key, when pressed, moves the cursor left one position. If the cursor is at the first position of a line when you press the CURSOR LEFT key, the cursor moves to the last character of the previous line. If the cursor is at the top of the screen when you press the CURSOR LEFT key, the text scrolls.
CURSOR RIGHT	Also called CURSOR EAST, this key, when pressed, moves the cursor right one position. If the cursor is at the last position of a line when you press the CURSOR RIGHT key, the cursor moves to the first position of the next line. If the cursor is at the bottom of the screen when you press the CURSOR RIGHT key, the text scrolls.
HOME	The HOME key moves the cursor to the beginning of the screen. When you press the HOME key and the CURSOR LEFT key simultaneously, the cursor moves to the first position of a line. When you press the HOME key and the CURSOR RIGHT key simultaneously, the cursor moves to the last position of a line.

Memory Console

The memory console, which resembles a double-cassette recorder, comes with two front-loading doors for loading digital data packs. Behind the left door is a digital data drive, referred to as drive A in this book. The second door is a facade. You can purchase the second digital data pack drive from your Coleco dealer for approximately $150. It is

designed so that you can install it by removing a few screws, taking away the facade, plugging the drive into the console or module, and replacing the screws.

To open a drive, use the release latch on top of the drive.

WARNING

A light on the memory console, indicating that the computer is writing to tape, also serves as a warning not to open the drive door. Opening the door during this operation will probably result in the destruction of the tape.

Figure 2.4 shows the memory console for the Family Computer System.

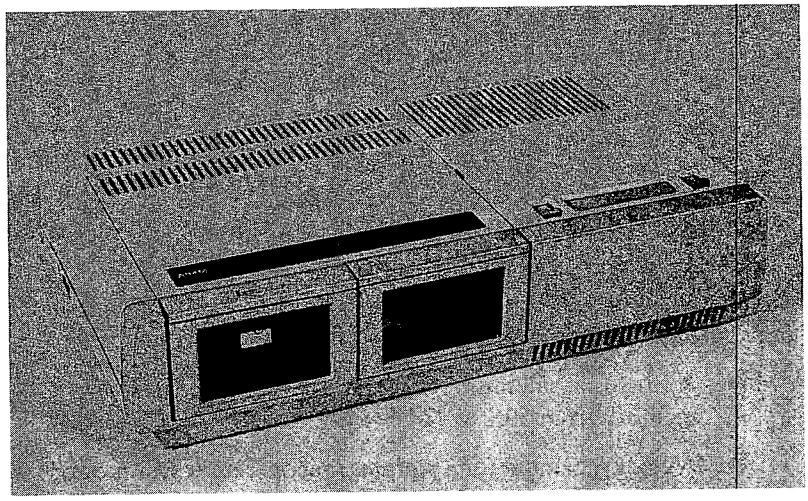


Figure 2.4
Memory Console for the Family Computer System

The memory console allows you to save programs written in SmartBASIC and documents created through SmartWRITER word processing. When you are using the SmartWRITER word processor to write, edit, or print documents, you can have digital data packs in either the first, the second, or both drives. If your memory console has only one digital data drive, you can LOAD the SmartBASIC digital data pack, remove it, and replace it with a blank or partially filled data pack for saving the programs that you write.

To LOAD SmartBASIC follow these steps:

1. Place the SmartBASIC digital data pack in drive A.
2. Pull forward the computer RESET lever on top of the console.
3. When the tape stops and the light goes out, open the drive door, remove the digital data pack, and place it in its storage box.
4. Replace the digital data pack with a blank or partially filled tape.

The memory console contains the 80K bytes of random-access memory (RAM) and 32K bytes of read-only memory (ROM). Of the 80K RAM, 16K are contained in the 9918 graphics processor chip. The other 64K RAM are on the Z80 chip. Approximately 20K of the 64K RAM are used to run SmartWRITER. The remaining 44K of RAM are used to hold text that you write. As a result, SmartWRITER responds to commands faster than many other word-processing systems.

ADAMNet, the system of processing chips in the console, is diagrammed in Figure 2.5.

Inside the memory console are three slots that allow you to plug in as many as three of the following optional boards at one time:

- Additional memory that expands ADAM's internal memory from 80K to 144K
- A clock/calendar for programs that operate at set time and date
- A switching system that you can program to switch a coffee pot or lights on and off
- A board (soon to be available) that makes ADAM CP/M compatible

Memory Module for the ColecoVision Video Game System

The Family Computer Module is a digital data storage device designed to be connected to the ColecoVision Game System. The module

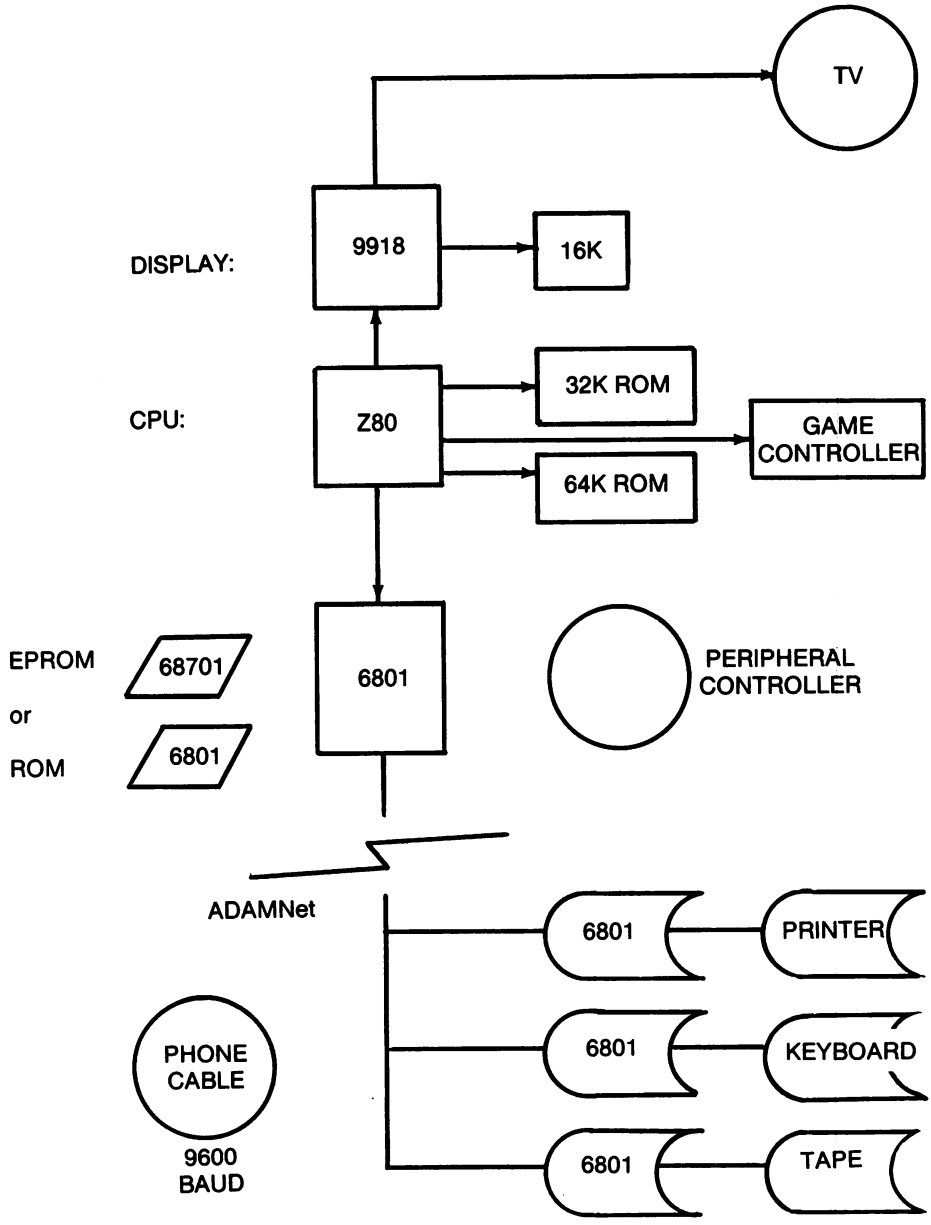


Figure 2.5
Diagram of ADAMNet

operates in the same way as the memory console that comes with the Family Computer System. The only difference is that the module does not include the game player. If you have ColecoVision, you don't need the entire system; you already have half of it. Figure 2.6 shows the Family Computer Module for the ColecoVision Video Game System.

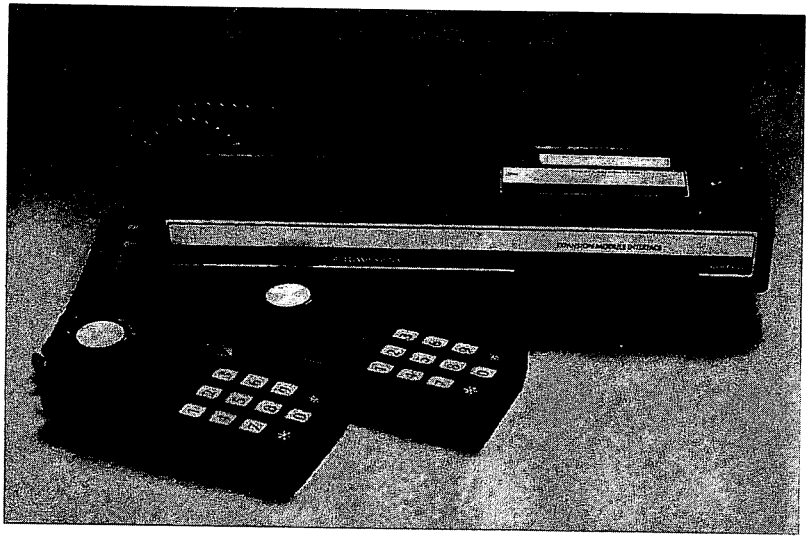


Figure 2.6
Memory Module for the ColecoVision Video Game System

Digital Data Pack

A digital data pack, which resembles an audio cassette tape, allows you to store files of up to 256,000 characters of information or 125 double-spaced pages of text. A digital data pack is shown in Figure 2.7.

The read/write speed of a digital data pack is 20 inches per second. The search speed is 80 inches per second.

To avoid the tedious task of searching for files, be sure to write the contents of the data pack on a label and attach it to the digital data pack. When the label is full, simply replace it with another self-adhering label. When labeling a digital data pack, you do not need to be as careful as with flexible diskettes, which are more fragile. Special labels are not necessary; any self-adhering label, available at stationery stores, will do.

Figure 2.7
Digital Data Pack

Additional certified digital data packs are sold by Coleco through the same distributors as the ADAM and are priced at about $10 each.

Printer

The SmartWRITER printer, which prints all or part of a document on letter-size (8 1/2" x 11") or legal-size (8 1/2" x 14") paper, consists of the following pieces and appears as shown in Figure 2.8.

- Power switch—Located next to the power cord on the back of the printer, the power switch is used to turn ADAM on and off.
- Daisy wheel—This standard Diablo-compatible wheel contains alphabet letters, punctuation marks, and numbers, and gets its name from its appearance.
- Ribbon—A standard Diablo-compatible ribbon cartridge that must be properly loaded to operate correctly
- Printer head—Moves against the ribbon and the daisy wheel, causing characters to appear on the paper

A paper tractor that holds fan-fold (also called continuous form) paper in place can be purchased for the ADAM.

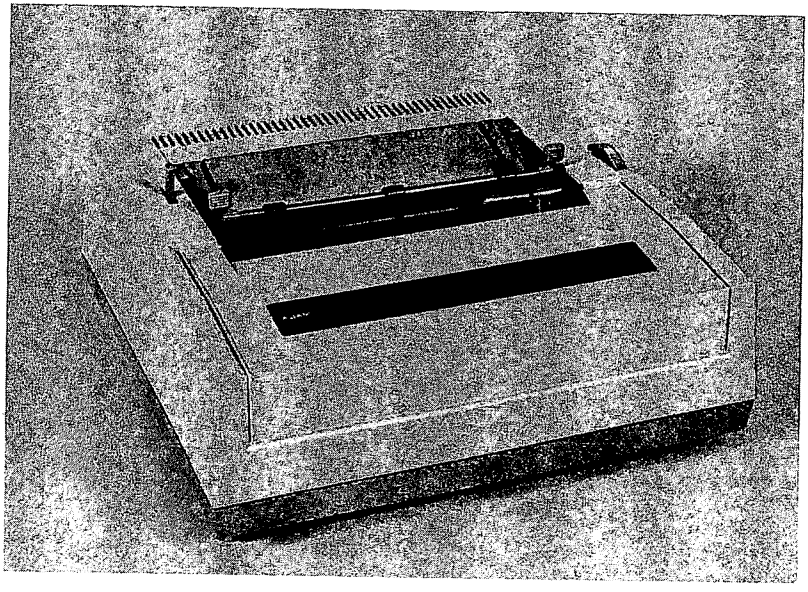


Figure 2.8
SmartWRITER Printer

When the SmartWRITER printer is on but not printing, it operates quietly because it has no fan. The carriage, which is 9 1/2 inches wide, limits you to printing 80 columns on letter- or legal-size paper. You cannot use ADAM to create tables or graphics wider than 80 columns.

Television or Monitor

ADAM is designed to be connected to your black-and-white or color television with the cables and modulator provided. Because color is used for various functions in the SmartWRITER word processor and in most graphics programs, the display results are much more effective on a color TV.

Nearly everything you type at the keyboard appears on the screen. Not all keystrokes place characters on the screen; some are commands that move the cursor and instruct ADAM to do various activities.

Figure 2.9 shows an example of a typical television to which ADAM can be connected.

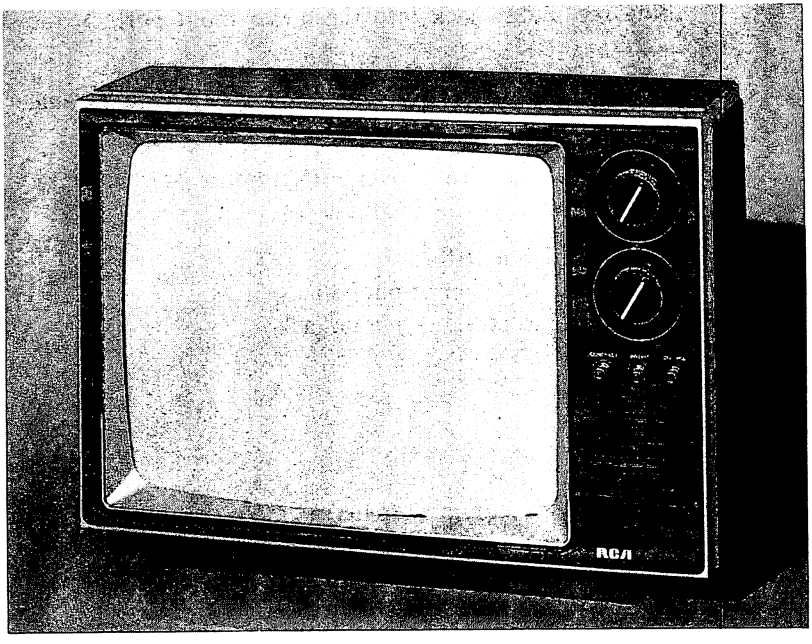


Figure 2.9
Connecting ADAM to a Television

Some television manufacturers, including Sony and Philco, are now distributing televisions that have direct hookups on the front, allowing you to hook up the computer more quickly and easily than to a connection in the rear of the television.

NOTE: In its revised documentation, Coleco provided instructions for hooking ADAM to a monitor. The instructions are outlined here for your reference:

- Plug the standard connector cable that is included with ADAM into the monitor port on the back of the console.
- You will also need a second cable, with either a 7-pin or 5-pin DIN connector at one end and a phono jack at the other end.

  Plug the DIN connector into the port marked "AUX VIDEO" on the back of the console.

Plug the phono jack into the audio input port on the monitor.

If your monitor has only a video port, you will not be able to receive sounds from ADAM.

- You may create your own DIN/phono jack cable by following these wiring instructions:

  5- or 7-pin DIN
  PIN 1 to center of phono jack
  PIN 2 to center of phono jack

Joysticks

Joysticks, included with the console package, are used to play games or move the cursor while programming or using SmartWRITER. Since they are already in the ColecoVision Video Game, joysticks are not included in the module package designed to be used with ColecoVision.

Software

Software is included for the SmartWRITER word processor, the BUCK ROGERS PLANET OF ZOOM Game, and the SmartBASIC programming language.

SmartWRITER

SmartWRITER consists of the Electronic Typewriter and SmartWRITER word processor. The Electronic Typewriter prints as you type. You can transfer what you type on the Electronic Typewriter to the SmartWRITER word processor by pressing the ESCAPE/WP key.

The SmartWRITER word processor allows you to type, edit, format, save, and print documents and parts of documents. SmartWRITER has a wraparound feature that wraps words that do not fit on a line to the next line. It also has a moving-window feature that is 35 characters wide.

BUCK ROGERS PLANET OF ZOOM Super Game Pack

The BUCK ROGERS PLANET OF ZOOM Super Game Pack uses up to 144K of memory to provide arcade-like quality graphics. Other super game packs will be available for ADAM. In the meantime, you can use all the game cartridges you have already purchased for ColecoVision. If you have the memory console, not the memory module, you can purchase Coleco game cartridges or super game cartridges. In short, the ADAM upgrades not only to a computer, but also to a super game player.

SmartBASIC

The SmartBASIC programming language is used to write programs that process data and display graphics. SmartBASIC operates the same on the memory console as it does on the memory module.

You can read more about SmartBASIC—what it is and how to use it—in several of the chapters that follow. Chapter 4 is an introduction to programming in SmartBASIC. Chapter 5 discusses the statements and functions used to input, process, and output data. Chapter 6 provides hints for screen and printout design. Chapter 7 contains a sample program described in detail that you can try in pieces or as a complete program. Chapter 8 describes statements used to create low-resolution and high-resolution graphics. Chapter 9 has more programs that you can try.

3

Introduction to Programming

This chapter covers the following topics:

- Programs and programming
- Information about your computer needed for programming
- The role of documentation
- The requirements definition
- The functional specification
- Programming as a profession

Programs and Programming

A computer cannot think or comprehend as a human can. If you want a computer to do something, you must write step-by-step instructions.

Programs

Written in a programming language, a program is a set of instructions, also called code or lines of code, that the computer can interpret and carry out. A program can instruct a computer to do one or all of the following:

- Add, subtract, multiply, and divide
- Display letters, numbers, and graphic characters on the screen
- Print results on the printer
- Store information in files
- Retrieve information from files

Programming

Programming is the process of writing a set of instructions (a program) for a computer. The programmer responsible for this step is also responsible for testing the program to make sure it runs properly and meets a set of predetermined goals.

The predetermined goals—the requirements definition and the functional specifications—are discussed at length later in this chapter.

The entire process, from goals to completion, can be either formal or informal. In the data processing department of a large business or in a commercial software development firm, the process is usually formal. Several people, including analysts, programmers, users, project leaders, and managers, may participate in writing the requirements definition and functional specifications. Sometimes a group of programs is required to meet the goals established by these people. The resulting programs are collectively called a system.

In contrast, programming on your home computer can be an informal process. The requirements definition and functional specifications can be drawn up casually, and programming can take place on a leisurely schedule. For this reason, home computer users who program for fun rather than profit are often called hobbyist programmers.

Information about Your Computer Needed for Programming

Imagine, if you will, three clock radios, all of which tell time and play music. Two may receive AM and FM stations, while the third receives only AM stations. Perhaps only one of the three plays in stereo. In addition, the method of setting the time and alarm probably differs

from clock to clock, as does the location of the switches that control the settings. The switches may be on the front of one, on the left side of the second, and on the top of the third.

Likewise, different computers have different capabilities, and even those with the same capabilities require a different set of commands to perform the same task. Consequently, part of learning to program is understanding the environment (the computer) in which you will work.

A discussion of the differences among computers raises another important issue: compatibility. Despite all their external differences, some computers are similar enough in their internal workings that programs written for one can be run on another. Not only, then, is the program said to be compatible with both machines, but the machines themselves are deemed compatible. By the same token, a computer may be designed to accommodate add-on hardware or other accessories that work on more than one machine.

Part of understanding your computer is understanding its compatibility with other hardware. Information about your computer's compatibility with other machines can be obtained from your dealer.

For the moment, let's deal with the issue of writing and editing a compatible program for your computer. What do you need to know?

- How to use the keyboard
- How much internal memory (now and the most it can be upgraded to)
- How much storage (now and the most it can be upgraded to)
- What programming languages will run on the computer
- Whether the language is in a chip or must be loaded from a cassette, diskette, or from the hard disk

How to Use the Keyboard

Chapter 2 describes the ADAM keyboard and provides an overview of how each key operates. The tutorial in Chapter 4 gives you a chance to use the keyboard to do some elementary programming and explains typical responses to your keystrokes.

How Much Internal Memory

ROM (read-only memory) contains preprogrammed instructions that are saved when the computer is off. RAM (random-access memory) is used to store data temporarily. When the power is turned off, the data in RAM is lost. Both ROM and RAM are measured in thousands of bytes, signified by K. Generally, the greater the number of bytes, the greater the capacity. The amount of internal memory (ROM and RAM) determines the amount of data that can be input and processed. Manufacturers that provide programming languages for their machines usually make sure there is enough memory in the machine to handle the programming activities built into the program statements.

For example, the SmartBASIC programming language requires approximately 16K to 20K bytes of RAM memory to interpret program lines that you type in or LOAD from a digital data pack. The size of RAM is important because the more program lines that the computer can store, the fewer times it has to read lines from the digital data pack. Reading from a digital data pack takes time and slows down the computer's response to you.

A computer is deemed "fast" when it has a lot of memory, and the programs take advantage of it.

How Much Storage

When you save a document or a program for future use, it must be stored in what is referred to as the computer's external memory. The amount of data that can be stored in one place in the external memory represents the computer's storage capacity. In the case of ADAM, all data is stored on digital data packs. Since a digital data pack can store up to 256K bytes (characters), which translates to about 125 double-spaced, typed pages or several thousand program lines, you should not have to worry about filling a digital data pack with a program you write.

What Programming Language Will Run on the Computer

The programming language that you want to use may not be available for your machine. The SmartBASIC programming language, a language compatible with Applesoft source code, comes with the

ADAM. Instructions for using it are in this book. Programming languages such as FORTRAN, C, Assembler, COBOL, Pascal, and PL/1 are not available for ADAM at this time.

Whether the Language Is in a Chip

Although the SmartWRITER word processor is contained on a chip, you must LOAD SmartBASIC from the SmartBASIC digital data pack into internal memory.

The next few sections discuss programming issues that computer professionals confront daily. While you may not deal with them on the same level, understanding how professionals tackle the various stages of programming will help prepare you for your own programming activities.

The Role of Documentation

Software documentation is any written material that explains what a program does and how it operates on a particular computer. Hardware documentation explains how to maintain and repair the hardware components of the computer. Documentation comes in many forms and is written for different audiences, such as first-time computer users, technicians, programmers, data entry operators, and business managers. Table 3.1 lists some of the various types of documentation.

The role of documentation is very important, yet extremely underrated. Documentation is an integral part of any program or system. The best-designed program in the world is useless if no one knows what it is for or how to use it. Functional specifications, programmer's guides, and user's guides explain the details of a computer system for people programming the system and others using it.

Most programmers, new and experienced, are not prepared to document their work properly. They complain that it's difficult to do and takes a lot of time, revealing an attitude that frequently leads to neglect and poorly written documentation.

This neglect, though bad for the computer industry, has a positive side effect. It has created a need for people who like to communicate. New breeds of technohumanists have evolved in the form of technical writers, technical illustrators, technical instructors, and course devel-

Table 3.1
Types of Documentation

Name	*Purpose*
Requirements Definition	Explains what the user should be able to do with the completed program
Functional Specification	Explains the basic functions used by the program to meet the requirements definition
Programmer's Guide	Explains in programming terms what the system does and how it does it
User's Guide	Explains what the system does and how it does it
Maintenance Manual	Explains in technical terms how the hardware works so that technicians can maintain and repair it

opers. These people learn how computers operate and explain what they have learned to others. Many of them come from academia, seeking new challenges or better pay. Others come directly from schools that provide programs in technical writing. Still others begin as programmers and later decide they prefer to write about computers rather than program them.

With the standardization of software and hardware, often the only difference between one product and another is the quality of support, a large part of which is documentation. Eventually, those responsible for documentation will be as important to the product as programmers and sales and field service personnel are now.

After you have gained an understanding of your computer, what is the next step in the programming process? Where do you begin?

The Requirements Definition

As mentioned earlier, a computer cannot think; it can only respond to a set of step-by-step instructions. Before writing those instructions, the programmer must have a set of user objectives. He must have a clear

notion of what the end-user will do with this program. These objectives make up the requirements definition.

In an informal atmosphere, such as your home, the requirements definition can take the form of a list or outline of the activities the user should be able to perform with the program. Later this list can be used to make sure the program has met all the user objectives.

For example, suppose you want to write a program providing practice in multiplication tables for your children in elementary school. You begin by writing what you want your children to do with the program. The list you come up with might look like the one in Figure 3.1.

MULTIPLICATION DRILL

The user of this program shall be able to do the following:

1. Practice multiplication tables from 0 to 12
2. Choose which tables to practice
3. View questions on the screen
4. Finish the selected table and choose another one or end the drill

Figure 3.1
List of Requirements

Once you have completed the requirements definition, you can begin to outline how the program will function to meet the user objectives. Your outline, no matter how sketchy, is called a functional specification.

The Functional Specification

The functional specification, often called a spec or specs, begins where the requirements definition ends. At the professional level, specs, which often include formulas and screen and printout illustrations, can run into hundreds of pages. The resulting documents are used to evaluate a program before it is produced. The evaluation process, which may take months and often leads to revisions, may result in scrapping the program.

The time required for a program project to be approved depends on many factors, the most important of which are its priority and complexity.

At its most informal level, the functional specification can be an outline that is filled in as the programmer works at the keyboard. However, a detailed set of specs has advantages even for the home computer user. For one, the specs serve as a list for checking whether all objectives have been met. For another, the specification helps the programmer respond to error messages that may appear when the program is typed in.

The functional specification for the multiplication drill mentioned earlier would describe the program in terms of input, processing, and output. The specs would include, but not be limited to, the following items:

- What the screen should look like
- What questions the student is asked
- How the computer accepts input
- How the computer does the calculations
- How the answer is shown on the screen

An outline of a program is shown in Figure 3.2.

MULTIPLICATION DRILL

Purpose: to provide practice in multiplication tables.

Input

1. Student types at keyboard
2. Programmer provides formula that computer uses to check student's answer: the table (selected by the student) multiplied by from 0 to 12 (selected by the computer as instructed by the programmer). Programmer assigns variable TABLE, variable S, and variable ANSWER.

   Formula: ANSWER = TABLE * S.
3. Computer accepts input from keyboard.

Processing

1. Computer calculates correct answer based on table student has selected and information provided by the programmer (see formula above).
2. Computer evaluates student's answer by comparing it to the correct answer.
3. Computer displays next question or redisplays current question depending on student's answer.

Output

The following prompts are displayed on the screen at one time or another while the program is running:

- Please type your name and press the RETURN key
- Do you want to continue?
- Press any key to continue
- Ok, (name of student), choose the table you want to be tested on
- Type 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, or 12 and press the RETURN key
- Type 99 when you want to finish the exercise
- What is A x B = ?
- Sorry, try again
- Good. The answer is XXXXX.
- Do you want to try another table?
- Type y for yes or n for no

Figure 3.2
Informal Functional Specifications

Programming as a Profession

Professional programmers come in all shapes and sizes, and so do their job descriptions. Some programmers sit at desks all day, writing code

lines that are entered into a computer by data entry personnel. Others type in their own code at terminals.

In some organizations, programmers are assigned tasks in addition to coding. They may, for example, participate in establishing user objectives and specs. In such companies, the programmer works closely with marketing personnel, whose responsibility is to know what their customers need. In other organizations, objectives and specs are drawn up and provided by other personnel, and the programmer simply uses his programming language skills to write the lines of code. In the computer departments of companies that produce products other than software, the jobs are similar. However, there are no marketing personnel; the programmers "market" the products to the other departments.

The size and work environment of a programming department depends on the attitude and needs of the company. For example, a large insurance company may have 500 people in its MIS (Management Information Services) department to run, repair, and program the computers. And the work day may be 8:30 to 4:30 with overtime required occasionally. On the other hand, a six-month-old computer company operating on venture capital probably has a staff of no more than five jack-of-all-trades programmers who nearly live in the three-room office and wipe the last of their lunch from their hands to their jeans as they continue working. These examples represent opposite ends of the spectrum, and there are many variations that fall in between.

If you were to listen to an interview with a programmer, you might hear the following exchange:

Interviewer: How did you decide to become a programmer?

Programmer: I was intrigued by computers, largely because they're new, and I like to tinker with things.

I enjoy breaking things into minute pieces and then putting them back together again. I also enjoy figuring out why something doesn't work. I get to do that when I debug a program. Though I wouldn't want that to be my only task.

Interviewer: So you enjoy programming.

Programmer: Yes.

Interviewer: What do you enjoy most about it?

Programmer: The best part is when the project is just beginning, and everyone is coming up with ideas about what modules to use, that is, how to handle the activities the programs will perform. I usually work on a few programs at the same time. That's because any piece of a system, even small systems, requires at least a few programs.

Interviewer: What happens when people can't agree on the best way to go, or if people are using different programming styles that conflict?

Programmer: That can be a problem. A lot of programmers see their work as an art. They want to create the program at the terminal. Some don't like to share what they're doing with other programmers. I don't agree with that. I think there's something special about programming. I enjoy doing it more than any other job I've had; but I realize, too, that when I took the job, I agreed to work on the team. And it's our job to create a unified program for the people who are going to use the system.

Interviewer: Doesn't it get difficult sometimes, though, when you feel strongly that something should be designed a certain way?

Programmer: It's frustrating when I'm asked to do something in a way I know won't work. I feel it is my responsibility to point out those things before they get into a program. It saves time and frustration, which, in the long run, the customer will pay for.

Interviewer: What if you explain the situation, and they still want you to do it that way?

Programmer: If I'm absolutely convinced, I'll create the situation that I'm talking about to show them what I mean. That usually works. Sometimes, though, I might be convinced or willing to believe that the situation I found isn't likely to occur because of the way the module fits into the program.

Interviewer: What makes one programming job more creative than another?

Programmer: When I first started programming, everything was exciting, and I felt very creative, even when I was doing mundane things like documentation.

As I became more experienced, some of the initial excitement wore off, but I still enjoyed programming. I remember when we finished that first project. I was so proud; I had done a lot of work and had even come up with a few things that impressed the senior level programmers. I felt I was being creative. That was satisfying.

That feeling continued through the next project and into the third. But somewhere in the third I began to wonder about what it would be like to work on other types of programming projects. Everything I had been doing up to that point was for accounting applications. I decided I wanted to try programming an operating system. It was almost like starting from the beginning again because writing programs for operating systems is different from writing programs for accounting applications. But it was even more exciting in a way because I could use the skills I had picked up in the previous position. One thing I've learned is that as long as I am developing new skills, I feel creative and don't get bored.

Interviewer: I get the idea you don't particularly enjoy writing documentation. Is it a necessary evil?

Programmer: It's necessary, but it isn't an evil. Documentation is extremely important. I don't like doing it because I prefer programming. I have also found that I'm not the best person to do it. There are people who specialize in gathering information about programs and systems and writing up manuals that describe how to use programs. They're usually called technical writers and documentation specialists.

Documentation is more than just writing up instructions about how to use a program—though that in itself is a tremendous effort. It involves anything that informs the programmer or the user what the program is about and how to use it. For example, comment statements in a program are extremely important because the programmer who wrote the program may not be the one who goes in later and changes or fixes it. Or, the programmer himself may go back to a program that he wrote quite a while ago. Having those comments right in the program to tell him what a particular module is supposed to do is extremely useful.

Interviewer: Well, thank you for your time. I've gained a lot of insight into what it's like to be a programmer.

Programmer: No problem. I enjoy programming and I like to talk about it.

If you are considering becoming a professional programmer, keep in mind that many companies require programmers to prepare formal requirements definitions and functional specifications.

Regardless of the formality or the purpose, though, all programs need to be planned, written, and later tested to make sure they work.

4

Programming Adam: Tutorial

This chapter provides instructions for setting up ADAM, loading SmartBASIC from the digital data pack into memory, and programming on ADAM.

If you are familiar with the concepts listed below, you may want to skip to the next chapter, which describes statements and functions of the SmartBASIC programming language.

The following exercises guide you in a step-by-step fashion through the activities involved in beginning programming. Each exercise provides instructions and explains the intended results.

- Exercise 1: Getting Started—checking connections, powering up, and loading SmartBASIC
- Exercise 2: Your First Program
- Exercise 3: Using ADAM as a Calculator
- Exercise 4: Assigning Line Numbers
- Exercise 5: Defining Input

- Exercise 6: Comparing Strings
- Exercise 7: Saving Programs
- Exercise 8: Loops
- Exercise 9: Conditions and Branches
- Exercise 10: Changing the Current Output Device

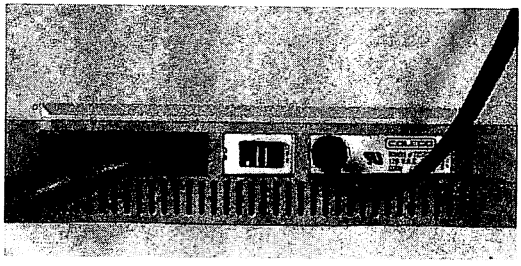
Exercise 1: Getting Started

To get started, check for the following:

- The keyboard is connected to the data pack drive.
- The television is connected to the keyboard.
- The printer is connected to the memory module or console.
- The joysticks are connected to the memory console. (They are already connected to the memory module.)
- The plug is correctly placed in an outlet.

When the connections are correct, turn the switch at the back of the printer to the ON position as shown in Figure 4.1.

Figure 4.1
Location of Power Switch



When the power comes up, the entry level of the SMARTKEY LABELs for the Electronic Typewriter appears on the screen as shown in Figure 4.2.

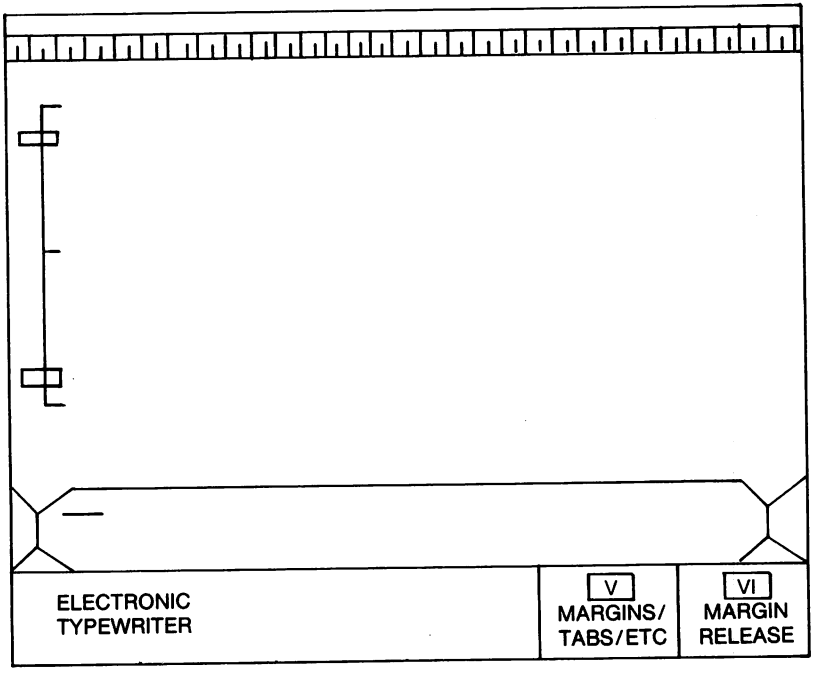


Figure 4.2
Appearance of Screen after Switching on Power

Place the digital data pack labeled "SmartBASIC" into data pack drive A. Find the RESET lever on the memory console or module. Pull the RESET lever forward. When SmartBASIC is LOADed, the screen background changes from the Electronic Typewriter to black.

Now remove the SmartBASIC digital data pack and place a blank or partially full data pack in the drive.

WARNING

If you try to remove a digital data pack while a program is LOADing, being SAVEd, or SEARCHing, the pack will be destroyed.

The exercises in this chapter are written so that you can take a break between them. If you want to stop using ADAM after an exercise, follow these steps:

1. Make sure you remove the SmartBASIC digital data pack and store it in a safe place.
2. Turn off the TV.
3. Turn off the power switch at the back of the printer.
4. Remove the plug from the wall outlet.

Exercise 2: Your First Program

Now that SmartBASIC is running, you may start using it to write programs. The definition of a program is simple—one or more instructions that tell the computer how to do something—so let's start with a simple program. Type the program shown in Figure 4.3 exactly as it appears, making sure to leave a space between PRINT and the opening quotation mark ("). Note that in SmartBASIC you must type also the closing quotation mark.

NOTE: For ease of reading, all characters in the programming lines in this and other chapters have been printed in uppercase. To make your programs run, you do not have to type them in uppercase. You may type in lowercase, which is easier to create on the ADAM keyboard. SmartBASIC will automatically convert all commands, such as PRINT or INPUT, to uppercase. The only time that you need to concern yourself with case is inside quotation marks. Characters typed inside quotation marks will appear on the screen exactly as you type them in.

```
PRINT "HELLO"
```

Figure 4.3
Program to PRINT "HELLO"

If you make a typing error, press the BACKSPACE key to move the cursor to the point at which you made the error, and retype the line from there.

Press the RETURN key. ADAM executes, that is, follows, your instructions immediately, and the screen appears as shown in Figure 4.4. This result is called immediate execution.

```
]Print "HELLO"
HELLO
]_
```

Figure 4.4
Result of Your First Program

Notice that when ADAM executes an instruction to print, it prints neither the word *PRINT* nor the quotation marks—only the word or words within the quotation marks. The word *PRINT* is a statement. A programming language consists of many statements, each of which has preprogrammed instructions for the computer.

The PRINT statement tells the computer to print the text after the next double quotation mark. Unless you have specified otherwise, ADAM will output to the screen, and the word *HELLO* will appear on the screen.

When a program line with a PRINT statement is read by the computer, the computer follows the preprogrammed instructions that tell it how to print the specified output. The screen is the default or assumed current output device. (The current output device is changed by using the PR# statement.) For this example, you are using the default output device. For more information about the PR# statement and other statements used to define output, see Chapters 5 and 8.

Exercise 3: Using ADAM as a Calculator

So far, you've seen how to use the PRINT statement to write a program. No calculations were required in the HELLO program. Now let's see how ADAM is used as a calculator.

To use SmartBASIC as a calculator, simply type in a PRINT statement that includes the equation you want solved and press the RETURN key. SmartBASIC displays the answer on the screen. Just as the PRINT statement provides instructions to the computer, so do arithmetic operators. Table 4.1 lists the arithmetic operations and the operators used to indicate them.

Table 4.1
Arithmetic Operators

Operation	*Operator*
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^ (read: to the power of)

Try entering some equations for ADAM to solve. Figure 4.5 shows examples of PRINT statements with equations and SmartBASIC responses as they appear on the screen. Use these and also enter some of your own.

```
PRINT 20 * 5
100

PRINT 300 / 6
50

PRINT 2 ^ 2
4

PRINT 5467 - 213
5254

PRINT 20342 + 196
20538
```

Figure 4.5
Examples of Equations

An equation containing more than one operator will be solved in the following preprogrammed order of precedence:

- Exponents (^)
- Multiplication (*)
- Division (/)
- Addition (+)
- Subtraction (-)

Use parentheses, (), braces, {}, or brackets, [], to override the preset rules when necessary and to make equations easier for you to follow. For example, the following equation has parentheses that instruct the computer to solve the parts in a specific order—from inside to outside:

((20342 + 196) - 3) * X

Exercise 4: Assigning Line Numbers

You do not need to assign a number to each line if you want your programs to execute immediately. However, without line numbers, lines are not retained in memory and cannot be stored on a digital data pack. If you want to enter several lines and execute them later—referred to as deferred execution—you must assign line numbers. Also, as you will see, line numbers are used in the LIST command to display or print single lines or a range of lines.

If you have worked with programs before, you are aware that line numbers are assigned by tens: 10, 20, 30, 40, 50, etc. This method allows you to add program lines without renumbering subsequent program lines. SmartBASIC has a built-in editor that arranges program lines in the correct order. This editor is especially useful when you are writing a program from a rough outline. Chances are you will be inserting many lines.

To see the difference between immediate execution (no line numbers) and deferred execution, type the following:

```
10 PRINT "HELLO"
```

and press the RETURN key.

Are you waiting for something to happen? Okay, now type RUN and press the RETURN key. The line you typed appears on the screen.

Now if you type the LIST command, the contents of memory might appear as shown in Figure 4.6.

```
]10 PRINT "HELLO"

]RUN
HELLO

]LIST

10 PRINT "HELLO"

]_
```

Figure 4.6
LISTed Memory After Lines with Numbers

Press the S key while holding down the CONTROL key to stop LISTing and temporarily suspend output to the screen. Press any key to start output to the screen again. To stop LISTing permanently, press the C key while holding down the CONTROL key. This command ends LISTing and execution of a program. To resume execution, use the CONT command; that is, type CONT and press the RETURN key.

NOTE: To stop execution of a program that is waiting for the user to respond to an INPUT statement, press C while holding down the CONTROL key and then press RETURN.

To see more, use the following steps:

1. Type

   ```
   20 PRINT "I AM FINE"
   ```

   and press the RETURN key. The screen appears as shown in Figure 4.7, and the cursor moves to the beginning of the next screen line.

```
]10 PRINT "HELLO"

]RUN
HELLO

]20 PRINT "I AM FINE"
]_
```

Figure 4.7
Lines 10 and 20

2. Insert a line between the other two without retyping any lines by using a line number that falls between these two line numbers. Type

   ```
   15 PRINT "HOW ARE YOU?"
   ```

   and press the RETURN key. The screen appears as shown in Figure 4.8, and the cursor moves to the beginning of the next screen line.

```
]10 PRINT "HELLO"

]RUN
HELLO

]20 PRINT "I AM FINE"

]15 PRINT "HOW ARE YOU?"

]_
```

Figure 4.8
Lines 10, 15, and 20

3. Type

   ```
   30 PRINT 7*8
   ```

and press the RETURN key. The display appears as shown in Figure 4.9, and the cursor moves to the beginning of the next line.

```
]10 PRINT "HELLO"

]RUN
HELLO

]20 PRINT "I AM FINE"

]15 PRINT "HOW ARE YOU?"

]30 PRINT 7*8

]_
```

Figure 4.9
Lines 10 through 30

4. Type the LIST command as follows to list only lines 10 and 20:

   ```
   LIST 10, 20
   ```

   and press the RETURN key. The screen appears as shown in Figure 4.10, and the cursor moves to the beginning of the next line.

5. Ready for something new? To remove a line, simply type the number of the line you want removed. This procedure is very easy, perhaps too easy. To remove more than one line, use the DEL command with the numbers of the first and last lines (separated by a comma) you want deleted. For example, if you wanted to delete lines 5 through 10, you type DEL 5,10. If you use the DEL command without line numbers, the LINE NUMBER RANGE EXPECTED message appears.

   Lines that you want to keep can very easily be removed accidentally, so take care when giving instructions to remove lines.

```
]10 PRINT "HELLO"

]RUN
HELLO

]20 PRINT "I AM FINE"

]15 PRINT "HOW ARE YOU?"

]30 PRINT 7*8

LIST 10, 20

10 PRINT "HELLO"
15 PRINT "HOW ARE YOU"
20 PRINT "I AM FINE"
```

Figure 4.10
LIST 10, 20

For this example, type 20 and press the RETURN key. Then type the LIST command and press the RETURN key to see the result. The screen appears as shown in Figure 4.11, and the cursor moves to the next line.

Exercise 5: Defining Input

Changing output by inserting and removing program lines is tedious and time-consuming and defeats the purpose of creating programs. There are other ways to enter data. You can instruct the computer to input, get, and read data by using INPUT, GET, DATA, and READ statements to define and redefine variables.

Variables

A variable is an element that can have many different values. Text variables, those that consist of words or phrases, are called strings. Numeric variables are used in calculations. When setting up a variable in your program, give it a name that identifies it for you.

```
]20 PRINT "I AM FINE"

]15 PRINT "HOW ARE YOU"

]30 PRINT 7*8
LIST 10, 20

10 PRINT "HELLO"
15 PRINT "HOW ARE YOU?"

20 PRINT "I AM FINE"

]LIST
10 PRINT "HELLO"
15 PRINT "HOW ARE YOU?"
30 PRINT 7*8

]_
```

Figure 4.11
LIST After DELeting Lines

Although SmartBASIC looks at only the first two characters of the variable name, you may want to use more characters to make the variable easy for you to identify. More rules for variable names are described in detail in Chapter 5, "The SmartBASIC Language."

NOTE: SmartBASIC will change all variable names to lowercase, no matter how you type them in. The only time you must be careful about uppercase or lowercase is when you are typing characters inside quotation marks.

INPUT and GET Statements

The INPUT and GET statements are placed in programs to instruct the computer to accept data from the keyboard. The INPUT statement accepts a response from the keyboard after the RETURN key is pressed. The GET statement accepts a one-character response from the

keyboard or other input device without waiting for the user to press the RETURN key.

To see how a variable is used with an INPUT statement, type NEW to clear the previously entered program lines from memory; then type the following program lines. Remember to press the RETURN key at the end of each program line. An underline (_) symbol indicates where you should press the space bar.

```
10 INPUT "WHAT IS YOUR NAME? "; NAME$
20 PRINT "HI,_"; NAME$
```

Notice that a semicolon is used in line 10, and a space, as indicated by an underline (_), is placed between the comma (,) and the quotation mark (") in line 20. Punctuation and spacing are very important in programming. The semicolon indicates that the input typed at the keyboard is the value of string variable NAME$. The space is added in line 20 to separate the word *HI* from your name when it appears on the screen.

Other punctuation marks, such as the colon, also have special meanings. The colon is used to combine instructions on one line. There are advantages and disadvantages to combining instructions. When you combine instructions, there are fewer lines for the computer to execute, so the program runs faster. On the other hand, program lines with several instructions are sometimes hard to follow, making editing difficult.

Type RUN and press the RETURN key. The display appears as shown in Figure 4.12.

```
]NEW

]10 INPUT "WHAT IS YOUR NAME? "; NAME$

]20 PRINT "HI, "; NAME$

]RUN
WHAT IS YOUR NAME?_
```

Figure 4.12
Name Program

Now type your name and press the RETURN key. ADAM responds as shown in Figure 4.13.

```
]NEW

]10 INPUT "WHAT IS YOUR NAME? "; NAME$

]20 PRINT "HI; ": NAME$

]RUN
WHAT IS YOUR NAME? BEV
HI, BEV

]_
```

Figure 4.13
Response to Name Program

To see how a variable is used with a GET statement, type NEW to clear memory. Then press the RETURN key and type the following program lines. Remember to press the RETURN key at the end of each program line.

```
10 PRINT "PRESS ANY KEY TO CONTINUE_";
20 GET A$
30 HOME
```

Type RUN and press the RETURN key. The display appears as shown in Figure 4.14.

```
]10 PRINT "PRESS ANY KEY TO CONTINUE ";

]20 GET A$

]30 HOME

]RUN
PRESS ANY KEY TO CONTINUE _
```

Figure 4.14
Continue Program

This program stops the action until the user presses a key, giving the user a minute to breathe before continuing to another part of a program. Line 20 instructs the computer to get the response. Once the response is received, line 30 is executed, instructing the computer to move the cursor to the beginning of the screen and clear it. Press any key and see what happens: the screen clears, as instructed. This and other short programs can be incorporated into longer programs.

DATA and READ Statements

DATA and READ statements work in pairs: for each READ statement, there must be a DATA statement; otherwise, the user will receive an error message. DATA statements are used to embed information that is read into variables. When the computer executes a READ statement followed by a variable, the program reads the first unread DATA statement into that variable. The next READ statement reads the next DATA statement into the next variable, and so on, in a sequential manner, until each pair has been executed.

RESTORE Statement

You can, however, use the RESTORE statement to reuse the list of DATA statements. When the computer reaches a RESTORE statement, it finds the first DATA statement. The next READ statement reads the second DATA statement, and so on. This concept is illustrated in Figure 4.15.

REM Statement

As your programs become more complex, you should include statements that explain the purpose of each line or group of lines, as necessary. The REM statement, which is short for REMARK, allows you to type comments that the computer will not execute.

Type the next program to get a feel for how REM, DATA, READ, and PRINT statements are used in a program.

```
10 REM INSTRUCTIONS TO READ DATA INTO VARIABLES THAT WILL BE
   PRINTED
20 READ ADDRESS$: REM READ STATEMENTS
30 READ CITY$: READ ST$: READ ZIP$
```

```
3010 RESTORE
3030 FOR I=1 TO 27: READ WRD$
3040 IF LEFT$(WRD$, 1) = F$ THEN I=27
3050 NEXT
3060 If WRD$="NOT FOUND!" THEN PRINT: PRINT "MAKE FIRST LETTER
     UPPERCASE!": PRINT
4000 REM PRINTING INSTRUCTIONS
4010 PRINT: PRINT "YOUR NAME AND "; WRD$: PRINT "START WITH THE
     SAME LETTER. "
4020 RETURN
5000 REM DATA
5010 DATA "APPLE", "BEAR", "CAT", "DOG"
```

Figure 4.15
Flow of a Program That Contains
READ, DATA, and RESTORE Statements

```
40 DATA "140 W. MAIN STREET ", "AMHERST, ", "NEW
   HAMPSHIRE ", "30103"
50 REM PRINT INSTRUCTIONS
60 PRINT "ADDRESS: "; ADDRESS$
70 PRINT "CITY STATE: "; CITY$; ST$
80 PRINT "ZIP CODE: "; ZIP$
```

REM statements are used in lines 10, 20, and 50 to explain the purpose of the program lines. Notice that a colon is used in line 20 to combine a READ statement and a REM statement.

NOTE: If you combine a REM statement in a line with other statements, make certain that you place the REM statement at the end of the line. Any statement after the REM statement is not executed.

Colons are also used in line 30 to combine three READ statements and variables into which data will be read onto one line. Spaces are placed in the strings following the DATA statements in line 40. Semicolons separate the string variables in lines 60, 70, and 80 to indicate that the computer should print the value of the variables next to each other. Since spaces were left in the DATA statement strings, the text will be correctly spaced when it appears on the screen.

Type RUN and press the RETURN key. The display appears as shown in Figure 4.16.

```
]10 REM INSTRUCTIONS TO READ DATA INTO VARIABLES THAT WILL BE
    PRINTED

]20 READ ADDRESS$: REM READ STATEMENTS

]30 READ CITY$: READ ST$: READ ZIP$

]40 DATA "140 W. MAIN STREET ", "AMHERST, ", "NEW HAMPSHIRE ",
    "30103"
]50 REM PRINT INSTRUCTIONS
]60 PRINT "ADDRESS: "; ADDRESS$
]70 PRINT "CITY, STATE: "; CITY$; ST$
]80 PRINT "ZIP CODE: "; ZIP$
]RUN
ADDRESS: 140 W. MAIN STREET
CITY, STATE: AMHERST, NEW HAMPSHIRE
ZIP CODE: 30103
]_
```

Figure 4.16
Using DATA, READ, and PRINT Statements

Exercise 6: Comparing Strings

To get the desired responses on the screen or printout, or to do other comparisons and calculations, you can use functions and statements to tell the computer to evaluate and compare strings. Strings are evaluated in terms of length and positions of characters within a string. You can use the LEN function to determine the length of a string and use that value later in the program. For example, you can write a short program that counts the number of words in a page by (a) letting each sentence be a string (255 characters maximum), (b) using the LEN function to count the number of words in each string, and (c) adding the values together.

A less complex example is the following program, which counts the number of characters in a sentence.

```
10 REM ASSIGN VARIABLE TO LENGTH OF SENTENCE
20 NUMBER% = LEN("STRINGS ARE FUN TO MANIPULATE. ")
30 PRINT "THERE ARE_"; NUMBER%; "_CHARACTERS"
40  PRINT "_IN THAT SENTENCE. "
```

Type NEW to clear memory and type in this program. Type RUN and press the RETURN key. The display appears as shown in Figure 4.17.

```
]RUN
THERE ARE 29 CHARACTERS
 IN THAT SENTENCE.
]_
```

Figure 4.17
Number of Characters in a Sentence

Now try a program that accepts input from the keyboard.

```
10 REM GET, COUNT, AND PRINT THE NUMBER OF LETTERS IN FIRST
   NAME
20 REM REQUEST FIRST NAME
30 INPUT "PLEASE TYPE YOUR FIRST NAME:_"; NAME$
40 REM FIND THE LENGTH OF NAME
50 L% = LEN (NAME$)
60 REM PRINT FINDINGS
70 PRINT "WELL, _"; NAME$; "=THERE ARE _" ; L%;
   "_CHARACTERS"
80 PRINT "_IN YOUR FIRST NAME. "
```

Line 30 requests the first name and assigns the variable name NAME$ to the string variable. Line 50 finds the number of characters in the name and assigns L% as the variable name of the result. Lines 70 and 80 instruct the computer to print a sentence that contains that name as typed (NAME$) and the number of characters in the name (L%).

Type RUN and press the RETURN key; type your first name and press the RETURN key. The screen appears as shown in Figure 4.18.

Answer the prompt and press the RETURN key to see the result.

```
]RUN
WELL, NAME$ THERE ARE L% CHARACTERS
IN YOUR FIRST NAME.
]_
```

Figure 4.18
Number of Characters in First Name

The LEFT$, MID$, and RIGHT$ functions are useful for reading parts of a string for later comparison to another string or part of a string. Within these functions, you can use a string or a variable string name. These functions can be used to create different code names as shown in the following programs.

First, to clear memory, type NEW and press the RETURN key. Then type the following:

```
10  REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST
    NAMES
20  REM REQUEST FIRST AND LAST NAMES
30  INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
40  INPUT "PLEASE TYPE YOUR LAST NAME:_"; LAST$
50  REM GET THE FIRST 3 LETTERS OF THE FIRST NAME
60  F$ = LEFT$(FIRST$, 3)
70  REM GET THE FIRST 3 LETTERS OF THE LAST NAME
80  L$ = LEFT$(LAST$, 3)
90  REM DEFINE CODE BY CONCATENATING STRINGS
100 CODE$ = F$ + L$
110 REM PRINT INSTRUCTIONS FOR CODE
120 PRINT FIRST$; ", _YOUR CODE NAME IS_" ; CODE$
```

Lines30 and 40 request the first and last names and assign variable names (FIRST$ and LAST$). Lines 60 and 70 find the first three letters of the first and last names.

Line 100 introduces a new concept: concatenation, which means chaining strings together. In this example the contents of string variable F$ and string variable L$ are chained and defined as string variable CODE$.

When concatenating strings, you must keep in mind how much space you want between the strings. When strings are chained, no space is left

between them. To separate strings by one or more spaces, you must construct the strings so that they have the necessary space within them. In this example no space was required between strings.

Line 120 instructs the computer how to display the code on the screen. Type RUN, answer the prompts, and press the RETURN key after each response. The screen appears as shown in Figure 4.19.

```
]NEW
]10 REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST NAMES
]20 REM REQUEST FIRST AND LAST NAMES
]30 INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
]40 INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
]50 REM GET THE FIRST 3 LETTERS OF THE FIRST NAME
]60 F$ = LEFT$(FIRST$, 3)
]70 REM GET THE FIRST 3 LETTERS OF THE LAST NAME
]80 L$ = LEFT$(LAST$, 3)
]90 REM DEFINE CODE BY CONCATENATING STRINGS
]100 CODE$ = F$ + L$
]110 REM PRINT INSTRUCTIONS FOR CODE
]120 PRINT FIRST$; ", YOUR CODE NAME IS " ; CODE$
]RUN
PLEASE TYPE YOUR FIRST NAME: BEV
PLEASE TYPE YOUR LAST NAME: DARWENT
BEV, YOUR CODE NAME IS BEVDAR
]_
```

Figure 4.19
Code Name of the First Three Letters of First and Last Names

To use the MID$ function to create a different code, change line 80 by typing

```
70 REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST
   NAME
80 L$ = MID$(LAST$, 2, 3)
```

and press the RETURN key. To see the change in the program, type LIST -120. The screen appears as shown in Figure 4.20.

```
]LIST -120
10  REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST NAME
20  REM REQUEST FIRST AND LAST NAMES
30  INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
40  INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
50  REM GET THE FIRST 3 LETTERS OF THE FIRST NAME
60  F$ = LEFT$(FIRST$, 3)
70  REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST NAME
80  L$ = MID$(LAST$, 2, 3)
90  REM DEFINE CODE BY CONCATENATING STRINGS
100 CODE$ = F$ + L$
110 REM PRINT INSTRUCTIONS FOR CODE
120 PRINT FIRST$; ", YOUR CODE NAME IS "; CODE$
]_
```

Figure 4.20
Code Program with MID$ Function

Line 80 now starts at the second character of the last name and takes that character and the next two characters as the value of variable L$. Type RUN and press the RETURN key. After you answer the questions, the screen appears as shown in Figure 4.21.

To create another code, this time using the RIGHT$ function, type

```
50 REM GET THE LAST 2 CHARACTERS OF THE FIRST NAME
60 F$ = RIGHT$(FIRST$, 2)
```

and press the RETURN key. To see the change in the program, type LIST -120. The screen appears as shown in Figure 4.22.

Line 60 now takes the last two letters of the first name and assigns them to variable string F$.

Type RUN and press the RETURN key. Answer the questions once more. The screen appears as shown in Figure 4.23.

Exercise 7: Saving Programs

You've done a fair amount of programming in this chapter. To save yourself the trouble of retyping programs, you will want to SAVE them on a digital data pack and LOAD them into memory later.

```
]LIST
10  REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST NAMES
20  REM REQUEST FIRST AND LAST NAMES
30  INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
40  INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
50  REM GET THE FIRST 3 LETTERS OF THE FIRST NAME
60  F$ = LEFT$(FIRST$, 3)
70  REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST NAME
80  L$ = MID$(LAST$, 2, 3)
90  REM DEFINE CODE BY CONCATENATING STRINGS
100 CODE$ = F$ + L$
110 REM PRINT INSTRUCTIONS FOR CODE
120 PRINT FIRST$; ", YOUR CODE NAME IS "; CODE$
]RUN
PLEASE TYPE YOUR FIRST NAME: BEV
PLEASE TYPE YOUR LAST NAME: DARWENT
BEV, YOUR CODE NAME IS: BEVARW
]_
```

Figure 4.21
Code Name of First Three Letters of First Name and Second, Third, and Fourth Letters of Last Name

```
]LIST -120
10  REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST NAMES
20  REM REQUEST FIRST AND LAST NAMES
30  INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
40  INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
50  REM GET THE LAST 2 CHARACTERS OF THE FIRST NAME
60  F$ = RIGHT$(FIRST$, 2)
70  REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST NAME
80  L$ = MID$(LAST$, 2, 3)
90  REM DEFINE CODE BY CONCATENATING STRINGS
100 CODE$ = F$ + L$
110 REM PRINT INSTRUCTIONS FOR CODE
120 PRINT FIRST$; " , YOUR CODE NAME IS "; CODE$
]_
```

Figure 4.22
Code Program with RIGHT$ Function

```
]NEW
]10 REM CODE USING FIRST THREE LETTERS OF FIRST AND LAST NAMES
]20 REM REQUEST FIRST AND LAST NAMES
]30 INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
]40 INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
]50 REM GET THE LAST 2 CHARACTERS OF THE FIRST NAME
]60 F$ = RIGHT$(FIRST$, 2)
]70 REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST NAME
]80 L$ = MID$(LAST$, 2, 3)
]90 REM DEFINE CODE BY CONCATENATING STRINGS
]100 CODE$ = F$ + L$
]110 REM PRINT INSTRUCTIONS FOR CODE
]120 PRINT FIRST$; " , YOUR CODE NAME IS "; CODE$
]RUN
PLEASE TYPE YOUR FIRST NAME: BEV
PLEASE TYPE YOUR LAST NAME: DARWENT
BEV, YOUR CODE NAME IS EVARW
]_
```

Figure 4.23
Code Name of Last Two Letters of First Name and Second, Third, and Fourth Letters of Last Name

The SAVE command saves everything in memory to a file you name on a digital data pack. If a file by that same name is already on the digital data pack, the program in memory overwrites the program on file. If there is no file by the name given, SmartBASIC creates one and writes the program into it.

To save the most recent program in memory (see Figure 4.23) to a file named CODE, make sure a blank or partially full digital data pack is in drive A and type SAVE CODE.

The screen appears as shown in Figure 4.24.

Press the RETURN key. As you can see, the program remains in memory.

NOTE: Whenever you want to LOAD—that is, copy—a program into memory from the digital data pack, simply type LOAD and the FILE NAME, then press the RETURN key.

```
]130  INPUT "PLEASE TYPE YOUR FIRST NAME: "; FIRST$
]140  INPUT "PLEASE TYPE YOUR LAST NAME: "; LAST$
]150  REM GET THE LAST 2 CHARACTERS OF THE FIRST NAME
]160  F$ = RIGHT$(FIRST$, 2)
]170  REM GET THE SECOND, THIRD, AND FOURTH LETTERS OF THE LAST NAME
]180  L$ = MID$(LAST$, 2, 3)
]190  REM DEFINE CODE BY CONCATENATING STRINGS
]100 CODE$ = F$ + L$
]110 REM PRINT INSTRUCTIONS FOR CODE
]120 PRINT FIRST$; ", YOUR CODE NAME IS "; CODE$
]RUN
PLEASE TYPE YOUR FIRST NAME: BEV
PLEASE TYPE YOUR LAST NAME: DARWENT
BEV, YOUR CODE NAME IS: EVARW
]SAVE CODE
]_
```

Figure 4.24
SAVE Command

Exercise 8: Loops

Sometimes you will want to repeat the same activity for different values of a variable string or number. FOR and NEXT statements are used to create loops in which an activity is repeated for each indicated value. For this example use the NEW command to clear memory; then type the following program. Type your name where it says "YOUR NAME. "

```
10  REM PROGRAM THAT REPEATS YOUR NAME
20  REM ASSIGN VARIABLE NAME TO STRING
30  N$ = "YOUR NAME"
40  REM INITIALIZING LOOP
50  L = 0
60  FOR L = 1 TO 6
70  REM PRINT INSTRUCTIONS
80  PRINT N$; SPC(4) N$
90  INVERSE
100 PRINT N$; SPC(4); N$
110 NORMAL
```

```
120 REM FIND NEXT
130 NEXT: REM IF LOOP REPEATED 6 TIMES, THEN END
140 END

]RUN
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
YOUR NAME YOUR NAME
```

Line 30 allows you to type your name or someone else's into this example. If you prefer, you can delete lines 20 and 30 and substitute "YOUR NAME" for N$ in lines 80 and 90 (which you would change to 60 and 70 if you delete 20 and 30). Line 50 initiates the loop at 0 (zero). Line 60 tells the computer how many times to repeat the loop.

Lines 80 and 90 provide the instructions that will be repeated. In this case, they are print instructions that tell the computer to

- print your name
- leave four spaces
- reverse the display
- print your name
- leave four spaces
- return to normal display

A new function, introduced as SPC(4), is used in this program. The SPC(x) function is a numeric function that places x number of spaces on the screen or moves the printer head x spaces, depending on which device is the current output device. (See PR# statement in Chapter 5 for more information.)

If you want to see what "; SPC(x)" in line 80 does, try running the line without those statements.

Line 150 instructs the computer to run the loop for the next value of L. Implicit in the NEXT statement is the instruction to end the loop if the instructions have been followed for all values of L. Line 160 is an instruction to end the program. Even though a program ends when it runs out of data, the END statement acts as a safety device to make sure the program stops.

Type RUN and press the RETURN key. Depending on the length of your name, the screen will appear similar to the one shown in Figure 4.25.

```
BEV DARWENT     BEV DARWENT
BEV DARWENT
    BEV DARWENT
    BEV DARWENT
BEV DARWENT     BEV DARWENT
BEV DARWENT
    BEV DARWENT
    BEV DARWENT
BEV DARWENT
BEV DARWENT     BEV DARWENT
    BEV DARWENT
    BEV DARWENT
BEV DARWENT     BEV DARWENT
BEV DARWENT
    BEV DARWENT
    BEV DARWENT
BEV DARWENT     BEV DARWENT
BEV DARWENT
    BEV DARWENT
    BEV DARWENT
BEV DARWENT     BEV DARWENT
    BEV DARWENT
    BEV DARWENT
```

Figure 4.25
Name Loop

SAVE the program by typing SAVE LOOP and pressing the RETURN key.

Exercise 9: Conditions and Branches

Sometimes you want the computer to evaluate input and, depending on the results, follow or not follow subsequent instructions. In addition to arithmetic operators, relational and logical operators are available for evaluating input. Tables 4.2 and 4.3 show relational and logical operators.

Table 4.2
Relational Operators

Operator	*Operation*
=	Equal to
<	Less than
>	Greater than
<=, =<	Less than or equal to
>=, =>	Greater than or equal to
<>, ><	Not equal to

Table 4.3
Logical operators

Operator	*Operation*
AND	Both true
OR	Either or both true
NOT	Is false

You can instruct the computer to branch to another program line (ON . . . GOTO and IF . . . THEN) or to a group of program lines, called a subroutine (GOSUB and ON . . . GOSUB). The difference between branching to another program line and branching to a subroutine is that with a subroutine the RETURN statement is used to

send the computer back to the line following the GOSUB or ON . . . GOSUB statement. When the computer is sent to another program line by the IF . . . THEN statements, or GOTO or ON . . . GOTO statements, it does not return to the line following the branch point; that is, the RETURN statement is not used with these statements.

Subroutines are useful for creating a simple, well-structured program. You decide what you need the program to do, write several lines of instructions for each activity, use REM statements to label them as subroutines, and use the GOSUB and ON . . . GOSUB statements to instruct the computer to find the first line of the subroutine.

The following program evaluates the input and, depending on the result, instructs the computer to do various activities. This program shows the use of the GOSUB statement to branch to other program lines, execute a subroutine, and then return (the RETURN statement) to the line following the GOSUB statement. Before trying this program, SAVE the program that is in memory to a file named LOOP if you want it on a digital data pack. If you have already SAVEd or don't want to, then type NEW and press the RETURN key.

```
10 REM INSTRUCTIONS FOR NAME GAME
20 GOSUB 1000:REM REQUEST NAME
30 GOSUB 2000:REM GET FIRST LETTER
40 GOSUB 3000: IF WRD$="NOT FOUND!" GOTO 20: REM FIND
   MATCHING WORD
50 GOSUB 4000:REM PRINTING INSTRUCTIONS
60 GOSUB 5000:REM DATA
70 END
1000 REM INSTRUCTIONS FOR NAME GAME
1010 INPUT "PLEASE TYPE YOUR NAME: "; NAME$
1090 RETURN
2000 REM GET THE FIRST LETTER OF THE NAME
2010 F$=LEFT$(NAME, 1)
2090 RETURN
3000 REM COMPARE FIRST LETTER OF THE NAME WITH FIRST LETTER
3010 RESTORE
3030 FOR I=1 TO 27: READ WRD$
3040 IF LEFT$(WRD$, 1)= F$ THEN I=27
3050 NEXT
```

```
3060 IF WRD$="NOT FOUND!" THEN PRINT: PRINT "MAKE FIRST
     LETTER UPPERCASE!": PRINT
3070 RETURN
4000 REM PRINTING INSTRUCTIONS
4010 PRINT: PRINT "YOUR NAME AND "; WRD$: PRINT "START WITH
     THE SAME LETTER. "
4020 RETURN
5000 REM DATA
5010 DATA "APPLE", "BEAR", "CAT", "DOG"
5020 DATA "ELEPHANT", "FAWN", "GOAT", "HORNET"
5030 DATA "INSECT", "JAM", "KITE", "LION"
5040 DATA "MOUSE", "NIGHT", "OXEN", "PORCUPINE"
5050 DATA "QUAIL", "RHINOCEROS", "SENTENCE"
5060 DATA "TAIL", "UNIVERSAL", "VIRUS", "WALRUS"
5070 DATA "XRAY", "YOKE", "ZEBRA", "NOT FOUND!"
```

The name game gets the first letter of the computer user's name, finds a word that starts with the same letter, and prints the word in a space. Lines 20 through 60 instruct the computer to go to the subroutines. The subroutine that begins on line 1000 instructs the computer to prompt the user for his or her name. The subroutine that begins on line 2000 gets the first letter of the name and assigns it to a string variable named **F$**.

The subroutine that begins on line 3000 is a loop that instructs the computer to READ each DATA statement into a string variable named **WRD$** (line 3030). Then the computer tests the data. If the first letter of the data is the same as the first letter in the name, the computer sets the loop index to the end, which **RETURN**s the computer to the beginning of the statement after GOSUB 3000. If the first letter of the data is not the same as the first letter in the name, the computer ignores the second part of the conditional statement (**IF . . . THEN**) and executes the next value of the loop. If the first letter is not a capital letter, then an error message is returned.

The subroutine that begins on line 4000 instructs the computer to print a sentence with the name retrieved in subroutine 2000 and the word returned in subroutine 3000.

Type RUN and press the RETURN key; answer the prompts and press the RETURN key. The screen appears as shown in Figure 4.26.

```
3030 FOR I=1 TO 27: READ WRD$
3040 IF LEFT$(WRD$, 1) = F$ THEN I=27
3050 NEXT
3060 IF WRD$="NOT FOUND!" THEN PRINT: PRINT "MAKE FIRST LETTER
     UPPERCASE!": PRINT
4000 REM PRINTING INSTRUCTIONS
4010 PRINT: PRINT "YOUR NAME AND "; WRD$: PRINT "START WITH THE
     SAME LETTER. "
4020 RETURN
5000 REM DATA
5010 DATA "APPLE", "BEAR", "CAT", "DOG"
5020 DATA "ELEPHANT", "FAWN", "GOAT", "HORNET"
5030 DATA "INSECT", "JAM", "KITE", "LION"
5040 DATA "MOUSE", "NIGHT", "OXEN", "PORCUPINE"
5050 DATA "QUAIL", "RHINOCEROS", "SENTENCE"
5060 DATA "TAIL", "UNIVERSAL", "VIRUS", "WALRUS"
5070 DATA "XRAY", "YOKE", "ZEBRA", "NOT FOUND!"
```

Figure 4.26
Name Game

To SAVE this program, make certain that a digital data pack is in drive A, type SAVE GAME, and press the RETURN key.

Exercise 10:
Changing the Current Output Device

Up to this point you have saved a number of programs. Table 4.4 lists the programs and the file names to which you were instructed to save them.

You may have assigned file names other than the ones suggested in Table 4.4. To find out what is on the digital data pack in drive A, type CATALOG and press the RETURN key.

To get a printout of what is on the digital data pack in drive A, use the following steps:

Table 4.4
List of Programs

Program Name	*File Name*
Code Name of Last Two Letters of First Name and First Three Letters of Last Name	CODE
Name Loop	LOOP
Name Game	GAME

1. Check that the daisy wheel, ribbon, and paper are correctly loaded.
2. Type PR# 1 to direct output to the printer, and press the RETURN key.
3. Type CATALOG and press the RETURN key.

Now select something from the list that you want to print on your printer or view on the screen. Since you have already changed the output to the printer to get a printout, simply type LIST [FILE NAME] and press the RETURN key.

If you want to view the program on the screen, you must change the PR# statement again. To do that, type PR# 0 to direct output to the screen, and press the RETURN key. When you type LIST and press the RETURN key, the last program in memory appears on the screen. To display a program that has been saved to a digital data pack, you must first load it into the computer's memory. Type LOAD LOOP and press the RETURN key.

Be careful when using the LOAD command. Whatever you LOAD from a digital disk pack replaces the program currently in internal memory.

Now when you type LIST and press the RETURN key, the screen appears as shown in Figure 4.27.

```
]LIST
10  REM PROGRAM THAT REPEATS YOUR NAME
20  REM ASSIGN VARIABLE NAME TO STRING
30  N$ = "YOUR NAME"
40  REM INITIALIZING LOOP
50  L = 0
60  FOR L = 1 TO 6
70  REM PRINT INSTRUCTIONS
80  PRINT N$; SPC(4); N$
90  INVERSE
100 PRINT N$; PRINT SPC(4); N$
110 NORMAL
120 REM FIND NEXT
130 NEXT: REM IF LOOP REPEATED 6 TIMES, THEN END
140 END
```

Figure 4.27
LISTing of Name Loop on the Screen

Summary

In this chapter, you were given introductory information about checking connections, powering up, and loading SmartBASIC; running your first program; using ADAM as a calculator; assigning line numbers; defining input; comparing strings; saving programs, loops, conditions, and branches; and changing the current output device.

To learn more about these topics and other SmartBASIC statements, commands, and functions, continue on to Chapters 5, 6, 7, 8, and 9.

5

The SmartBASIC Language

In the previous chapters you were introduced to programming in general and given a chance to do some introductory programming. This chapter begins by bridging the gap from understanding what programming is to actually thinking in a programming language. The remainder of the chapter is a discussion of SmartBASIC, an Applesoft-compatible programming language. You can translate any program written in Applesoft to SmartBASIC without changing the logic. By the end of this chapter, you will begin to think and speak in SmartBASIC. Many of the program lines used as examples in this chapter are excerpts from programs in Chapters 7 and 9, where you can find more detailed explanations, if needed.

The following topics are discussed in this chapter:

- Viewing a program
- Putting it all together
- Turning specifications into code
- Defining SmartBASIC statements and functions (except graphics—see Chapter 8)

Viewing a Program

You know what a program is, why it's necessary, and that there are different levels of programming languages. Now let's take a look at how a program is structured.

Programming instructions fall into three broad categories: where to get information (input), what to do with that information (processing), and where to put the information (output).

Figure 5.1 shows a diagram of input, processing, and output.

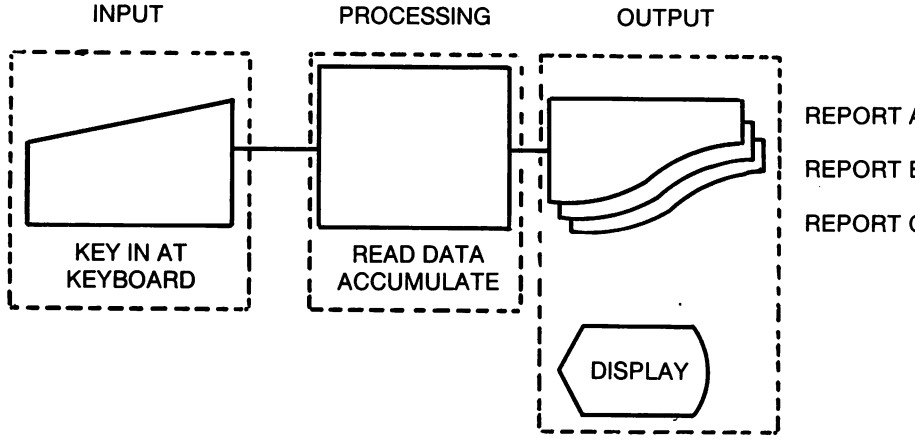


Figure 5.1
Input, Processing, and Output

This simplistic view of how a program works is often called the "black box" concept because the actual processing of the program's instructions takes place inside the computer and is invisible to the programmer. This simple explanation leaves most beginners wondering what really goes on in there. The following discussion attempts to demystify these activities.

Input

Before the computer can process data, it must be instructed as to where the data is coming from. In other words, you must define the source of the input, the source being a device such as the keyboard, a joystick, or a

digital data pack. Each of these devices is part of ADAMNet. When you define the source of the input, you are actually telling the computer from which part of ADAMNet to retrieve information. In addition, you can instruct the computer to use information that you provide in the program.

After you have instructed the computer to accept information from the keyboard, you must then indicate what user prompts to display on the screen and what type of data is to be input.

Process

Once you have defined where the information is coming from and what it is, you must instruct the computer how to interpret it. One way to do this is to set up conditions. If a specified condition is present, the computer follows a specified instruction, or path; if a different condition is present, the computer follows a second specified path. If neither condition is present, the computer follows a third path, according to your instructions.

Through the use of certain statements that, in essence, create a loop, you can repeat the previous set of instructions. Using other statements, you can create a subroutine, a set of instructions that can be repeated any place in the program without retyping the entire set of instructions. As you become adept at programming, you will recognize the best ways to instruct a computer to process information.

Output

After the information has been processed, the computer must have instructions about what to do with it. In this part of a program, you provide instructions for how information appears on the screen, how it appears as a printout, or in what form it goes to a storage device.

Use of the Program

Whether you are testing the program for the first time or using it for the tenth time, it will do the following:

- Accept data input from the specified input device or display an error message

- Test for specified conditions. If the condition is met, the information is processed according to the instructions provided. If the condition is not met, the information is processed according to other instructions. The program then accepts the next data input and tests it.
- Perform other specified activities, such as displaying results on the screen, displaying a menu, printing hard copy, and sending results to a file on a storage device
- End the program as instructed or when it runs out of data

In other words, the computer does exactly what you instruct it to do. This is true even though the computer does something you didn't instruct it to do, or doesn't do something you thought you instructed it to do. In programming, either a deliberate or an accidental omission of one or more characters in a program line amounts to an instruction that will make the computer perform or fail to perform in a certain way.

Each of the programming activities mentioned in the previous chapter is listed in Table 5.1 with the corresponding SmartBASIC statement. Later in this chapter, you will learn how to use these statements in program lines. You will also learn that just as there are different ways of saying the same thing in English, there are different ways of accomplishing the same task in SmartBASIC. You may want to experiment with these statements as you go along.

Table 5.1
English to SmartBASIC Translation at a Glance

English Activity	*SmartBASIC Statement*
Source of Input	
Expansion slot that input device is connected to	IN#
Data from input device	INPUT
Getting any data that comes from the input device	GET

Table 5.1 *(continued)*

English Activity	SmartBASIC Statement
Data in the program, supplied by the programmer	DATA
Reading data supplied by the programmer	READ
Rereading data supplied by the programmer	RESTORE
Defining Input	
Numbers, strings, variables, and constants used to manipulate numbers:	
Generates numbers between 0 and 1	RND (random numbers)
Creates functions to avoid repeating instructions	DEF FN (Creating new functions)
Provides the absolute value of the number or operation that you provide in parentheses (x)	ABS (absolute value)
Determines whether the argument provided in parentheses (x) is positive (1), negative (-1), or zero (0)	SGN (sign)
Used to produce a whole number from the argument	INT (integer)
Calculates the positive square root of the argument	SQR (square)
Calculates the sine of the argument (x)	SIN (sine)
Calculates the cosine of the argument	COS (cosine)

Table 5.1 *(continued)*

English Activity	SmartBASIC Statement
Calculates the tangent of the argument (x)	TAN (tangent)
Calculates the arc tangent of the argument (x)	ATN
Calculates the exponential value of the argument (x)	EXP
Calculates the logarithm of the argument (x)	LOG
Functions used to manipulate strings:	
Counts the number of characters in a string, where "s" is the string	LEN (length)
Takes the specified number of characters from the left side (beginning) of string, where "s" is the string and # is the number of characters	LEFT$
Takes the specified number of characters from the middle of the string, where "s" is the string, "p" is the starting position, and # is the number of characters	MID$
Takes the specified number of characters from the right side (end) of the string	RIGHT$
Changes a numeric value into a string, where "s" is the numeric value	STR$
Changes a string to a numeric value, where "s" is the string	VAL

Table 5.1 *(continued)*

English Activity	SmartBASIC Statement
Changes an ASCII code to a character, where "a" is the ASCII code	CHR$
Changes one character to an ASCII code, where "c" is the character	ASC
Arrays and Dimensions:	
Defines and allocates spaces for one or more arrays	DIM
Processing Instructions	
Arithmetic operators:	
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^
Relational operators:	
Equal to	=
Less than	<
Greater than	>
Less than or equal to	<=, =<
Greater than or equal to	>=, =>
Not equal to	<>, ><
Logical operators:	
Both true	AND

Table 5.1 *(continued)*

English Activity	SmartBASIC Statement
Either or both true	OR
Is false	NOT
Branches and loops:	
Directs the program to execute the instructions indicated by the line number following the statement	GOTO
Directs the computer to look for a specified subroutine	GOSUB
Instructs the computer to look to the line following the GOSUB or ON and GOSUB statements for the next instruction	RETURN
Used to branch to another instruction	ON . . . GOTO
Directs the computer to look for a specific subroutine based on the value of the expression	ON . . . GOSUB
Used with RETURN to send the computer to the previous GOSUB or ON and GOSUB statements	POP
Used to test for a particular condition	IF . . . THEN
Used together to create loops	FOR and NEXT
Designing Output	
Designate the output device to which data will be sent	PR#
Send data to output device	PRINT

Table 5.1 *(continued)*

English Activity	SmartBASIC Statement
Instruction to return the screen to a text screen from a low-resolution or high-resolution screen	TEXT
Clear text from screen and move cursor to beginning of first screen line	HOME
Place the specified number of spaces	SPC
Move the cursor the specified number of spaces in from the left margin	TAB
Move the cursor the specified number of spaces in from the left side of screen	HTAB
Move the cursor to a specific screen line	VTAB
Display the current cursor position	POS
Reverse appearance of the screen	INVERSE
Return INVERSEd screen text to normal	NORMAL
Change the speed at which text appears on screen	SPEED=

Putting It All Together

Similar to a part of speech in the English language, each SmartBASIC statement has a basic purpose, but every statement can be combined

with other statements to fulfill additional purposes. Before you create a program, you need to understand these basic concepts.

This chapter concentrates on the statements and the relationships between statements in several program activities. (To get an idea of how statements operate together in completed programs, see the discussion of the sample programs in Chapters 7 and 9.) As you read the description of each statement in this chapter, keep in mind the factors involved:

- The user, the person using the program
- You, the person creating the program
- The computer, which accepts the information from the person using it and is instructed on what to do with it by the program
- The peripheral devices that the computer interacts with during a program, such as digital data pack drives (in the memory console and module) and the printer

Turning Specifications into Code

You must follow several steps to turn functional specifications into code. As you become experienced at programming, you will find the sequence that is best for you. To begin, you might want to use the following steps:

1. Create components that make up the program.
   - Declare variables
   - Define constants
   - Create formulas using variables and constants
   - Decide whether or not there are conditions
   - Decide which details are repeated and how many times they are repeated
   - Create subroutines as needed
2. Use the following questions to determine where the pieces fit:

- When will data be input?
- When should data be printed or displayed?
- When should data be tested?
- What if a condition is met?
- What if a condition is not met?

3. Use statements to construct the program, combining the statements if desired.
4. Check the logic of your program, making certain that each instruction has a subsequent instruction, and that individual instructions are accurate.
5. Save the program to a digital data pack file and LOAD a copy into memory to experiment with.
6. Run the program.
7. Make changes as needed, being especially careful when deleting lines; it is very easy to delete more lines than you intended.
8. Repeat until the program operates as you want it to.
9. Save the revised program to the file that you created before,to a different file, or to a new file on a digital data pack.

Defining SmartBASIC Statements and Functions

To use this section effectively, read through the material once quickly to get an overview; then look at each area in more detail, particularly the tables that list and describe SmartBASIC statements. After a few readings, you should be able to fit the pieces together and gain an understanding of how SmartBASIC works.

Some activities, called functions, require even fewer specifics than do statements. In addition, SmartBASIC will do certain things or assign certain values, called default values or defaults, unless you tell it

otherwise. Examples of functions and defaults are given throughout this chapter.

Crossing the Language Barrier

Thinking in a computer language is easier than it sounds. English, the language you use every day, is more complicated than SmartBASIC. To make the English language easier to remember, you use abbreviations; you may even at times speak in an abbreviated manner. You can remember the parts of the language, put them together, and interpret the language when someone speaks to you. You can use that ability to learn, use, and interpret SmartBASIC and any other programming language.

Defining the Source of Input

As you know, a computer does only what it is instructed to do. One of the first instructions you must provide tells the computer where to look for input. Input can come from several sources: the keyboard, a joystick, a file on a digital data pack, or the program itself.

The following statements can be used to instruct ADAM to accept information from input devices, from storage devices, or from within the program:

IN#
INPUT
GET
READ and DATA
RESTORE

IN#

The IN# statement is used to identify the expansion slot from which you want the computer to retrieve information. Input and storage devices, including the keyboard, data pack drive unit, television, and modem, are connected to expansion slots. For a complete description of ADAM's input and storage devices, see Chapter 2. The acceptable values for this statement are listed in Table 5.2.

Table 5.2
Acceptable Values for IN# Statement

Value	*Meaning*
IN# 0	Receive information from keyboard
IN# 1	Receive information from expansion slot 1
IN# 2	Receive information from expansion slot 2
IN# 3	Receive information from expansion slot 3

The keyboard is the default input device, and unless you have changed the default, you do not need to use the IN# statement. The default device or specified device is referred to as the current input device.

The three lines of code in Figure 5.2 show how you might use the IN# statement in a program.

`320 IN# 3`	Directs the computer to accept information from the device attached to expansion slot 3
`330 GET C$`	Reads one character from the device connected to expansion slot 3. The `GET` statement is defined below. `C$` is an integer (see the discussion of variables and integers).
`340 IN# Ø`	Directs the computer to accept information from the keyboard

Figure 5.2
Example and Explanation of the IN# Statement

INPUT

The INPUT statement instructs the computer to accept data entered in response to a prompt, a question mark, or information typed at the location of the cursor.

Once the data is entered in response to a prompt, it is accepted, evaluated, and used as indicated by other instructions in the program.

Figure 5.3 shows an example of an INPUT statement directing the computer to do the following:

- Display a message asking whether the user wishes to continue
- Assign the variable string A$
- Evaluate the value of A$. If A$ is not "Y", then go to line 10. Otherwise, execute the next program line (probably 530).

```
510 INPUT "Do you want to continue?"; A$
520 If A$ <> "Y" THEN 10
```

Figure 5.3
Example of INPUT Statement

GET

The GET statement instructs the computer to read one character from the input device, which is usually the keyboard or a file stored on a data pack (see the IN# statement). The computer reads the character as soon as the user types it—without waiting for the user to press the RETURN key.

You must use a variable in the GET statement when you want the computer to accept any input from the keyboard or other input device (see the discussion about variables).

You might want to use the statement when you are directing the user to verify a choice. You first ask the user to make the choice, and then you ask "Are you sure?" to give him an opportunity to change his mind, as shown in Figure 5.4.

READ, DATA, and RESTORE

READ and DATA statements are used to read information that is contained in the program. READ specifies that the source is the program, and DATA indicates the input. More than one set of

```
320 PRINT "SELECT 0, 1, 2, OR 3"
330 PRINT "0 = elephant"
340 PRINT "1 = lion"
350 PRINT "2 = tiger"
360 PRINT "3 = bear"
370 GET ANIMAL$
380 PRINT "Are you sure? (Type y for yes or n for no)"
390 GET ANSWER$
400 IF ANSWER$ = "N" THEN 320
```

Figure 5.4
Examples of GET Statements

information can follow a single DATA statement. Each set, however, must be separated by a comma. You can assign a variable in which DATA is READ (see line 3030 in Figure 5.5).

The first time you instruct the computer to execute the READ statement, the first group of characters in the DATA statement, up to the comma, is read as the value of the first character in the READ statement.

The second time the READ statement is executed, the computer reads the second group of data from the DATA statement or looks for the next DATA statement to read from.

You cannot use a READ statement to instruct the computer to reread a previously read DATA statement or group within a DATA statement. However, you can use the RESTORE statement to instruct the computer to read the DATA statements from the beginning into a different READ statement.

The RESTORE statement allows you to indicate that another READ statement is to read the DATA statements in the program. An example of how you might use READ, DATA, and RESTORE statements appears in Figure 5.5.

These statements are useful when you have information that you want repeated in a program. However, to write a program that evaluates input from a user or file, you will want to create expressions that use variables and constants (see "Defining Input").

```
3000 REM COMPARE FIRST LETTER OF THE NAME WITH FIRST LETTER OF WORD
3010 RESTORE
3030 FOR I = 1 TO 27: READ WRD$
3040 IF LEFT$(WRD$, 1) = F$ THEN I = 27
3050 NEXT
3060 IF WRD$="NOT FOUND!" THEN PRINT: PRINT "MAKE FIRST LETTER
     UPPERCASE!": PRINT
3070 RETURN
4000 REM PRINTING INSTRUCTIONS
4010 PRINT: PRINT "YOUR NAME AND "; WRD$: PRINT "START WITH THE
     SAME LETTER. "
4020 RETURN
5000 REM DATA
5010 DATA "APPLE", "BEAR", "CAT", "DOG"
```

Figure 5.5
Examples of READ, DATA, and RESTORE Statements

Defining Input

Much of the work of programming involves defining the input in terms the computer can understand. The following SmartBASIC rules and functions are used extensively in the defining process.

- Constants and variables
- Variable names
- Real numbers and real variable names
- Integers and integer variable names
- Numeric functions
- Strings and string variable names
- Arrays and dimensions

Constants and Variables

At first, the terminology may be confusing. One way to distinguish a constant from a variable is to separate the value from the name. The

rules for naming constants and variables are the same; however, the value of the constant remains the same, and the value of the variable changes. Values are assigned by using the equals (=) symbol.

Variable Names

The computer uses variables to identify real numbers, integers (whole numbers), and strings (words). You must assign a name to each variable according to the following rules:

- All variable names start with a letter.
- Integers end with a % (percent) symbol.
- Strings end with a $ (dollar) symbol.
- No variable name can be more than 239 characters.
- No variable name can start with the first two letters of a reserved word or contain a reserved word. (Refer to Appendix D when naming variables.)

NOTE: SmartBASIC looks at only the first, second, and last characters of each variable name. To SmartBASIC, NUMBER% and NUMERAL% are the same, whereas NUMBER and NUMBER% are different.

Real Numbers and Real Variable Names

Real variables can include fractions and negative values. The default value of a real variable is 0 (zero) until it is assigned a different value. Valid and invalid examples are shown in Table 5.3.

Integers and Integer Variable Names

Integer variables are whole numbers. If you assign a number with a decimal as an integer variable, SmartBASIC truncates the fraction to the next lowest whole number. If the number is positive, SmartBASIC ignores everything after the decimal point. If the number is negative, SmartBASIC finds the next lowest whole number instead of just dropping the fraction.

Table 5.4 shows examples of what happens when real numbers are assigned as integer variables.

Table 5.3
Valid and Invalid Variable Names

Name	*Comment*
L	Valid
L2	Valid
COST	Invalid: reserved words CONT, COS
LOWCOST	Invalid: reserved words LOAD, LOG, CONT, COS
PRICE	Valid: reserved word PR# requires #
TEST	Invalid: reserved word TEXT
TITLE	Valid
SCREEN	Valid: reserved word SCRN requires a ((left parenthesis symbol) after N.

Table 5.4
Result of Real Numbers Assigned as Integer Variables

Assignment Statement	*Value*
A% = 62.139	62 is the assigned value of integer variable A%.
PROFIT% = -193.22	-194 is the assigned value of integer variable PROFIT%

Numeric Functions

Functions are previously programmed operations. A value is interpreted according to the function to which you assign it. In SmartBASIC, there are eleven numeric functions, plus a function that allows you to create your own function. The functions are defined briefly in this section.

RND

The RND function by itself generates numbers between 0 and 1. You can use it to produce random whole numbers by multiplying RND by an integer whose value is 10 or greater. The RND function is probably one of the most commonly used functions. It is used whenever a program calls for the generation of random numbers or the choosing of an activity at random. Figure 5.6 shows an example of the RND function.

```
100 REM **GENERATE 10 RANDOM NUMBERS **
120 FOR A = 1 TO 10
130 X = RND (1) * 10
140 X = X + 1
150 PRINT X
160 NEXT
```

Figure 5.6
Example of RND Statement

This program uses the FOR . . . NEXT loop (see the FOR and NEXT statements in this chapter) to repeat the instructions ten times and the INT function to produce whole numbers or real numbers (numbers with fractions).

DEF FN

With the DEF FN function you can create your own functions to avoid repeating instructions. The format is DEF FN name (numeric value). Once you have defined the function and given it a name, you simply use the FN and the name whenever you need to refer to the function.

The function that you define must be limited to one line, or a maximum of 239 characters. Only the first two characters are significant in distinguishing one function name from another.

The remaining functions are described in Table 5.5

Strings and String Variable Names

A string consists of any combination of letters, numbers, and punctuation. The variable name for that string must conform to the rules for

Table 5.5
Arithmetic Functions

Function	*Purpose*
ABS (x)	Provides the absolute value of the number or operation that you provide in parentheses (x)
SGN (x)	Determines whether the argument provided in parentheses (x), is positive (1), negative (-1), or zero (0)
INT (x)	Used to produce a whole number from the argument. The fractional part of the argument is ignored. This function does not round to the nearest whole number; it simply creates a whole number by lopping off the fraction. The result is the next lowest integer. For example, the value of INT (6.9) is 6, not 7. This is similar to what happens when you assign a real number as the value of an integer.
SQR (x)	Calculates the positive square root of the argument (x). An attempt to find the square root of a negative argument results in an error message.
SIN (x)	Calculates the sine of the argument (x), which must be in radians rather than degrees
COS (x)	Calculates the cosine of the argument (x), which must be in radians rather than degrees
TAN (x)	Calculates the tangent of the argument (x), which must be in radians rather than degrees
ATN (x)	Calculates the arc tangent of the argument (x), the result of which is in radians rather than in degrees
EXP (x)	Calculates the exponential value of the argument (x)
LOG (x)	Calculates the logarithm of the argument (x). An attempt to find the logarithm of a zero or negative number results in an error message.

general variable names and string variable names. The difference between string variables and numeric variables is that the user creates comparisons but not calculations with string variables.

You can use a string variable name such as ANSWER$ to accept the value of an answer, which might be "yes" or "no." You can then test for the presence of "yes" or "no" and instruct the computer to give a response based on the value of the string variable.

SmartBASIC supplies several functions that allow you to instruct the computer to identify and manipulate strings. These functions are described in Table 5.6.

Table 5.6
Functions Used to Identify Strings

Function	*Purpose*
LEN(s)	Counts the number of characters in a string, where "s" is the string
LEFT$(s,#)	Takes the specified number of characters from the left side (beginning) of string, where "s" is the string, and # is the number of characters
MID$(s,p,#)	Takes the specified number of characters from the middle of the string, where "s" is the string, "p" is the starting position, and # is the number of characters
RIGHT$(s,#)	Takes the specified number of characters from the right side (end) of the string
STR$(n)	Changes a numeric value into a string, where "n" is the numeric value
VAL(s)	Changes a string to a numeric value, where "s" is the string
CHR$(a)	Changes an ASCII code to a character, where "a" is the ASCII code
ASC(c)	Changes one character to an ASCII code, where "c" is the character. An attempt to find the code for a null string will result in an error message.

Arrays and Dimensions

Sometimes the information that is entered into a computer can be divided into groups of related items; each group is called an array. The items in an array can be real numbers, integers, or strings. You may want to group related information into a multiple-dimension array.

Figure 5.7 shows an example of arrays.

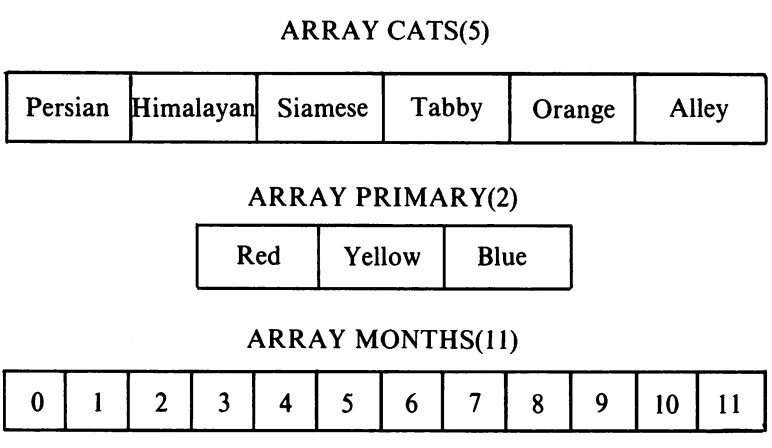


Figure 5.7
Examples of Arrays

Before creating an array, you must indicate its size so that the computer can allocate space for the elements in the array. The DIM statement, short for dimension, is used for this instruction. SmartBASIC numbers the elements from 0 (zero). For example, the dimensions for the array called PRIMARY with three elements is stated in Figure 5.8.

Processing Instructions

Up to this point, you have seen how statements and functions instruct the computer to accept and define data. In this section, you will learn how the computer evaluates and processes data.

```
DIM PRIMARY(2)
```

Figure 5.8
Dimension of Array PRIMARY

The first part of this section discusses expressions and operators used to instruct the computer to accumulate data and test it for the existence of specified conditions or values. The second part discusses the statements that direct the computer to look elsewhere (branch) in the program for further instructions:

GOTO
GOSUB
RETURN
ON . . . GOTO
ON . . . GOSUB
POP
IF . . . THEN

The third part discusses how the FOR and NEXT statements create loops that allow different data to be processed in the same way.

Using Expressions and Operators

Rather than make decisions, a computer compares and combines values. Expressions and operators are used to instruct the computer how to evaluate data. The result of an evaluation can trigger another evaluation or an instruction to execute other parts of the program. An expression is a formula that can consist of variables, constants, and operators. An operator is a symbol used to instruct the computer how to compare and combine values, as follows:

- Arithmetic operators—raise to a specified power (exponentials), add, subtract, multiply, divide
- Relational operators—compare to find the truth
- Logical operators—combine expressions and find the truth of both, one, or neither

Arithmetic operators are shown in Table 5.7.

Table 5.7
Arithmetic Operators

Operation	*Operator*
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^

Relational operators are shown in Table 5.8.

Table 5.8
Relational Operators

Operation	*Operator*
Equal to	=
Less than	<
Greater than	>
Less than or equal to	<=, =<
Greater than or equal to	>=, =>
Not equal to	<>, ><

Logical operators are shown in Table 5.9.

Table 5.9
Logical Operators

Operation	*Operator*
Both true	AND
Either or both true	OR
Is false	NOT

Figure 5.9 shows examples of how to use operators to instruct a computer to compare and combine variables and constants.

```
10 S >= 5 or S <> 100
20 TABLE * P = ANSWER
```

Figure 5.9
Examples of Expressions

Line 10 indicates that the value of variable "S" is greater than or equal to 5 or not equal to 100. Line 20 specifies that the value of variable "TABLE" multiplied by the value of variable "P" equals the value of variable "ANSWER. " ANSWER can then be displayed on the screen by using a PRINT statement.

Branching to Other Lines or to Subroutines

After the computer follows an instruction, it executes the next program line, unless you instruct the computer otherwise. The following statements can be used to direct (branch) the computer to a line other than the next line.

GOTO

The GOTO statement allows you to direct the program to execute, that is, carry out, the instructions indicated by the line number after the statement. This is called unconditional branching because you are requesting that the program branch to another instruction without testing for the presence or absence of a condition.

GOSUB

The GOSUB statement, also an unconditional statement, allows you to direct the computer to look for a specified subroutine. This statement differs from the GOTO in that it allows you to send the computer back to the statement that follows the GOSUB statement.

RETURN

The RETURN statement is placed at the end of a subroutine and indicates that the computer should look to the line following the

GOSUB or ON and GOSUB statements for the next instruction. Figure 5.10 shows an example of how the GOSUB and RETURN statements are used.

```
40  REM SCREEN TITLE
50  GOSUB 650
.
.
.
650 REM **TITLE SUBROUTINE**
660 HOME
670 TITLE$ = "MULTIPLICATION DRILL"
680 HTAB 15 - LEN(TITLE$)/2
690 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
700 RETURN
```

Figure 5.10
Example of GOSUB and RETURN Statements

ON . . . GOTO

The ON and GOTO statements are used to branch to another instruction, depending on the value of the arithmetic expression between the ON and GOTO portions of the statement. If the value is 0 or is greater than any of the numbers listed after GOTO, then the next statement on the same line of the program is executed.

ON . . . GOSUB

The ON and GOSUB statements allow you to direct the computer to look for a specific subroutine based on the value of the expression. The ON and GOSUB statements, though similar to the ON and GOTO statements, differ in that the ON and GOSUB statements allow you to send the computer back to the next part of the ON and GOSUB statements or the statement that follows the ON and GOSUB statements. Figure 5.11 shows an example of the ON and GOSUB statements.

POP

The POP statement is used with the RETURN statement to send the computer to an instruction different from the one it would normally go

```
2120 REM **CLEAR SCREEN AND DISPLAY TITLE BY USING TITLE
     SUBROUTINE AND GOSUB STATEMENT**
2130 ON 12 GOSUB 4000
2140 REM **INSTRUCTIONS FOR DISPLAYING TEST QUESTION AND
     PROCESSING STUDENT'S ANSWER**
2150 VTAB 15: PRINT "WHAT IS_"; TABLE; "_X_"; S; "_=_"; : INPUT ";
     ANSWER
2160 IF ANSWER<>TABLE*S THEN VTAB 17: PRINT "Sorry, try again":
     FOR W=0 TO 1000: NEXT: VTAB 17: PRINT GOTO 2150
2170 VTAB 17: PRINT "Good. The answer is_; " ANSWER; "."
2180 PRINT "Press any key to continue";
2190 GET A$
2200 NEXT: RETURN
```

Figure 5.11
Example of the ON and GOSUB Statements

to when it finds a RETURN statement. Instead of sending the computer to GOSUB or ON and GOSUB statements, POP sends the computer to the previous GOSUB statement or ON and GOSUB statements.

Suppose there are two subroutines. Using the POP statement after the second routine has been carried out returns the computer to the first subroutine instead of the second subroutine. An example of this is shown in Figure 5.12.

```
1080 REM RANDOMLY PICK ONE
1090 IF (N = 0) THEN R$ ="NONE": GOTO 1180
1100 N1=RND(1)*N*1.05: REM PICK WITH 5% FREE
1110 IF (N1>N) THEN R$ = "FREE": GOTO 1180
1120 RESTORE
1130 FOR N=1 TO N1
1140 READ T$: REM LOOP PAST IMPROPER ENTRIES
1150 IF LEFT$(T$, 1)<>S$ THEN 1140
1160 NEXT N
1170 R$=MID$(T$, 2, 255): REM PICK OFF SELECTOR
1180 RETURN
```

Figure 5.12
Use of the POP Statement

IF . . . THEN

The IF . . . THEN statement is used when you want to test for a particular condition and execute an instruction given in the same line. If the condition is met, that is, if it is true, the instruction after THEN is carried out. If the condition is not met, or is false, the instruction after THEN is ignored and the computer executes the next program line. Figure 5.13 shows an example of the IF . . . THEN statement.

```
300 REM PRINT MEALS
310 PR# P9
320 PRINT: PRINT "MEAL NUMBER "; I
330 IF (M$(1)<>"FREE") THEN GOTO 400
340 PRINT "PLEASANT SURPRISE: ": PRINT TAB (14)"GO OUT TO DINNER!"
350 GOTO 460
```

Figure 5.13
Example of the IF . . . THEN Statement

Creating and Using Loops: FOR and NEXT

The FOR and NEXT statements are generally used together to create loops. A loop consists of several instructions that are repeated for each input. The loop continues until all the information fed into the loop has been tested and acted upon. Then the NEXT statement feeds the next set of information into the loop. Figure 5.14 provides an example of how to use FOR and NEXT statements.

```
200 REM SELECT FOOD
210 FOR I = 1 TO 12
220 FOR J = 1 TO 6
230 S$ = MID$("AEVSDB", J, 1): REM SELECT COURSE
240 GOSUB 1000: REM FIND A COURSE
250 IF (R$="FREE") THEN M$(1)="FREE": GOTO 300
260 M$(J)=R$
270 NEXT J
```

Figure 5.14
Example of FOR and NEXT Statements

Defining Output

In addition to providing instructions about how to interpret the information, you must state what to do with the results. This section describes the statements used to create output instructions, such as what to do with the results of calculations, where to print prompts on the screen, and where to move the cursor. For hints about designing the appearance of the text screen and printouts, see Chapter 6.

The following statements and functions are described in this section:

PR#
PRINT
TEXT
HOME
SPC
TAB
HTAB
VTAB
POS
INVERSE
NORMAL
SPEED=

PR#

The PR# statement, structured like the IN# statement, is used to identify the part of ADAMNet to which you want the computer to send information. Output and storage devices, including the printer and monitor, are connected to expansion slots. For a complete description of ADAM's input and storage devices, see Chapter 2. The acceptable values for this statement are listed in Table 5.10.

If you have not changed the default output device, you do not need to include the PR# statement. SmartBASIC will output to the screen, which is the default. The default or specified device is referred to as the current output device.

The three lines of code in Figure 5.15 show how you might use the PR# statement in a program.

Table 5.10
Acceptable Values for PR# Statement

Value	*Meaning*
PR# 0	Send information to the screen
PR# 1	Send information to expansion slot 1 (printer)
PR# 2	Send information to expansion slot 2

100 PR# 1	Directs the computer to send information to the device attached to expansion slot 1, probably the printer
110 PRINT C%	Prints the value of C%. C% is an integer (See the discussion of variables and integers.)
120 PR# 0	Directs the computer to send information to the screen

Figure 5.15
Examples and Explanation of the PR# Statement

PRINT

The PRINT statement is used to send output to the current output device, the one that was most recently designated by the PR# statement.

Figure 5.16 shows examples of PRINT statements.

The following statements can be used to indicate how you want text to appear on the screen.

TEXT

The TEXT statement is used to end the graphics mode and begin the text mode. If you haven't used the graphics mode, this statement is not required. For more information about the low-resolution and high-resolution graphics screens, see Chapter 8.

```
400 PRINT "APPETIZER:"; TAB(12); M$(1)
410 PRINT "ENTREE:"; TAB(12); M$(2)
420 PRINT "VEGETABLE:"; TAB(12); M$(3)
430 PRINT "SIDE DISH:"; TAB(12); M$(4)
440 PRINT "DESSERT:"; TAB(12); M$(5)
450 PRINT "BEVERAGE:"; TAB(12); M$(6)
460 PR #0
470 NEXT I
500 END
```

Figure 5.16
Examples of PRINT Statements

HOME

The HOME statement is used to clear the screen and move the cursor to the upper-left corner of the text window. If you have defined a screen that is partially for text and partially for graphics, only the part defined as text is cleared, and the cursor is moved to the upper-left corner of that part of the screen.

You can also move the cursor to the upper-left corner of the screen without clearing it by using the HTAB statement with the VTAB statement as follows:

```
VTAB 1 : HTAB 1
```

SPC

The SPC function is placed in PRINT statements to put spaces before the text that is displayed on the screen or printed on a printer. The spaces are placed in relation to the position of the cursor at the time the instruction is given. For example, if the instruction before the SPC function leaves the cursor at position 4, then instruction `SPC (10) HELLO` tells the computer to print `HELLO` at position 14.

If you instruct the computer to leave more spaces than remain on the line, the computer will wrap the text and the spaces to the next line.

Figure 5.17 shows how you might use the `SPC` function.

```
10  REM PROGRAM THAT REPEATS YOUR NAME
20  REM ASSIGN VARIABLE NAME TO STRING
30  N$ = "YOUR NAME"
40  REM INITIALIZING LOOP
50  L = 0
60  FOR L = 1 TO 6
70  REM PRINT INSTRUCTIONS
80  PRINT N$ SPC(4); N$
90  INVERSE
100 PRINT N$; SPC(4); N$
110 NORMAL
120 REM FIND NEXT
130 NEXT: REM IF LOOP REPEATED 6 TIMES, THEN END
140 END
```

Figure 5.17 Example of SPC Function

TAB

The TAB function is placed in PRINT statements to move the cursor a specific number of positions from the left side of the screen. If you instruct the computer to tab over more positions than there are on the line, the computer will wrap the text and the spaces to the next line.

Figure 5.18 shows an example of how you might use the TAB function.

```
400 PRINT "APPETIZER:"; TAB(12); M$(1)
410 PRINT "ENTREE:"; TAB(12); M$(2)
420 PRINT "VEGETABLE:"; TAB(12); M$(3)
430 PRINT "SIDE DISH:"; TAB(12); M$(4)
440 PRINT "DESSERT:"; TAB(12); M$(5)
450 PRINT "BEVERAGE:"; TAB(12); M$(6)
460 PR #0
470 NEXT I
500 END
```

Figure 5.18
Example of TAB Function

HTAB

The HTAB statement is used to move the cursor a specified number of positions in from the left side of the screen. Unlike the TAB function, the HTAB statement can move the cursor to the left or to the right.

Figure 5.19 shows an example of how you can use the HTAB statement.

```
570  REM **INSTRUCTIONS FOR DISPLAYING PROMPT AND RESPONDING TO
     STUDENT'S ANSWER**
580  VTAB 15: PRINT "WANT TO TRY ANOTHER TABLE?"
590  HTAB 10: PRINT "TYPE Y FOR YES OR N FOR NO"
600  GET B$
610  IF B$ = "Y" THEN 340
620  B$ = "N" GOTO 10
4000 REM **SCREEN TITLE**
4010 HOME
4020 TITLE$=" MULTIPLICATION DRILL "
4030 HTAB 15-LEN(TITLE$)/2
4040 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
4050 RETURN
```

Figure 5.19
Example of HTAB Statement

VTAB

The VTAB statement is used to move the cursor to a specific line. The first line of the screen is 1, and the last line is 24.

Figure 5.20 shows an example of how you might use the VTAB statement.

POS

The POS function provides the current number of positions in from the left edge of the screen. It is used in a PRINT statement to display the cursor position on the screen or print the cursor position on paper.

```
180 REM **INSTRUCTIONS FOR CONTINUE PROMPT**
190 PRINT: PRINT "Do you want to continue?"
200 PRINT "PRESS Y FOR YES AND N FOR NO"
210 GET A$: IF A$= "N" OR A$="N" THEN PRINT: PRINT "OK BYE. . . ": END
220 HTAB 1: VTAB 22
230 PRINT "OF COURSE YOU DO_"; NAME$; "!"
250 VTAB 23
260 INVERSE
270 PRINT " PRESS ANY KEY TO CONTINUE ";
280 NORMAL
290 GET Z$
```

Figure 5.20
Example of the VTAB Statement

INVERSE

The INVERSE statement is used to display text in black-on-white instead of the standard white-on-black. The inverse of the standard scheme depends on how your monitor handles shading. You must use the NORMAL statement to return the appearance to its standard state.

Figure 5.21 shows an example of how you might use the INVERSE statement.

```
70  REM PRINT INSTRUCTIONS
80  N$; SPC(4); N$
90  INVERSE
100 PRINT N$; SPC(4); N$
110 NORMAL
120 REM FIND NEXT
130 NEXT: REM IF LOOP REPEATED 6 TIMES, THEN END
140 END
```

Figure 5.21

NORMAL

The NORMAL statement is used to turn off the INVERSE statement. Any text printed after the NORMAL statement is executed will appear in the screen's standard mode.

SPEED=

The SPEED= statement allows you to set the speed at which characters are sent to the screen, a printer, or to another output device. The slowest speed is 0 (zero) and the fastest speed is 255. SmartBASIC's default speed is 255.

Figure 5.23 shows an example of how you might use the SPEED= statement.

```
10 REM JAZZY NAMES
20 INPUT "WHAT IS YOUR NAME?_"; NAME$
30 SPEED= 100: PRINT "HELLO_"; NAME$
40 SPEED= 255: REM SET SPEED BACK
```

Figure 5.22
Example of SPEED= Statement

6

Text Screen and Printout Design

This chapter provides tips for designing the appearance of the text screen and printouts, including rules to remember when creating them. For information about low-resolution and high-resolution graphics screens, see Chapter 8.

The following screen design topics are discussed:

- Adjusting for the width of the screen
- Using white space
- Creating the illusion of more than one screen
- Using screens to introduce programs
- Centering text
- Using prompts
- Reversing text
- Placing screen instructions in a program as a subroutine

The following printout design topics are discussed:

- Considering width and length of paper
- Changing the output device
- Centering text

Introduction to Screen Design

The appearance of the screen determines how the user reacts to the program. You should consider the following when designing a screen:

- What does the user need to know about the program?
- What is the purpose of the program? Entertainment? Programming tool? Educational tool?
- Should the user have a choice of whether or not to continue?
- Does the user have several options from which to choose?
- Are there prerequisites for using the program?
- Should the text scroll off the screen, or should the screen be cleared and the cursor moved to another location?
- Should certain information be highlighted?

The answers to these questions can be listed in your requirements definition. Your functional specification explains how the requirements are carried out and also includes drawings of sample screens on screen design forms.

Adjusting for the Width of the Text Screen

As you design, you must be aware of how much room you have. You can instruct ADAM to place up to 31 characters on a line. However, if you use the TAB, HTAB, or SPC(x) functions, the text and space that exceed 31 characters wrap to the next line. Figure 6.2 shows the result of a program line that states:

```
10 PRINT TAB 5; PRINT "NOW IS THE TIME FOR ALL
GOOD PROGRAMMERS TO COME TO THE AID OF THEIR USERS. "
```

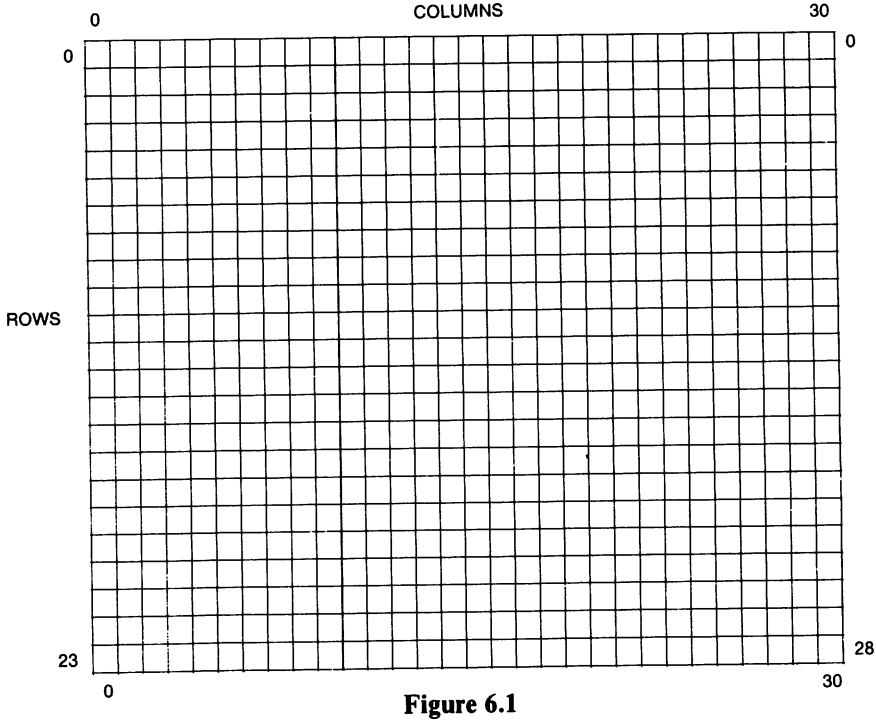


Figure 6.1
Screen Design Form for a Text Screen

One way to avoid wrapping text onto the next line is to break each sentence into parts. A good place to divide a sentence is where it breaks naturally, such as at a comma or the end of a phrase—someplace where the reader takes a breath. Figure 6.3 shows how the same sentence can be divided comfortably into parts.

The instructions required to create the screen in Figure 6.3 are as follows:

```
10 REM INSTRUCTIONS TO PRINT
20 PRINT "NOW IS THE TIME"
30 PRINT "FOR ALL GOOD PROGRAMMERS"
40 PRINT "TO COME TO THE AID"
50 PRINT "OF THEIR USERS"
```

```
]LIST
10 PRINT TAB (5); "NOW IS THE
TIME FOR ALL GOOD PROGRAMMERS T
O COME TO THE AID OF THEIR USER
S. "

]RUN
     NOW IS THE TIME FOR ALL GOO
D PROGRAMMERS TO COME TO THE AI
D OF THEIR USERS
```

Figure 6.2
Appearance of Text on the Screen

```
NOW IS THE TIME
FOR ALL GOOD PROGRAMMERS
TO COME TO THE AID
OF THEIR USERS
```

Figure 6.3
Appearance of Revised Program on the Screen

Using White Space

White space is a term used by graphic designers to describe the areas left blank in a layout. Just as a cluttered work area creates stress for the worker, so does a cluttered screen create stress for the viewer. Through the use of white space, you can design an appealing screen that is easy on the viewer's eyes. To create white space on a screen, you can use the following statements and functions:

- The PRINT statement by itself to skip lines
- The VTAB statement to print text on specific lines and leave others blank
- The HTAB statement and the TAB function to indent text

- The SPC function to leave spaces between words and phrases on a line

You need not use all these statements each time you create a screen. Use enough to create a screen that's comfortable to look at.

Creating the Illusion of More Than One Screen

You can instruct ADAM to place text on up to 24 lines, labeled 0-23. If you instruct ADAM to print text on the twenty-fourth or later line of the screen, the computer will respond with the error message "Illegal Quantity Error."

To avoid the scrolling, you can use the HOME statement to instruct the computer to clear the screen and begin displaying text at the top of the screen again.

NOTE: Unlike the CLEAR or NEW commands, the HOME statement does not clear memory.

By using the HOME statement and other statements to rebuild the screen, you can create the illusion of more than one screen. This technique is useful when you want to differentiate between parts of a program, such as an introductory description, activity selections (menus), and the body of the program.

Using a Screen to Introduce a Program

An example of a screen that introduces a program appears in Figure 6.4. This is a slightly different version of the Welcome Screen described in Chapter 7. Both are examples of ways to design a screen that introduces a program.

The screen in Figure 6.4 above consists of a title (line 2), a greeting (line 4), a brief description (lines 6 and 7), and two prompt lines (lines 10 and 11). The title and greeting are centered. The first line of the description is printed at the beginning of the screen line, and the second line of the description is on line 7. Two lines are skipped after the description. Both the first and second lines of the prompt are printed at the beginning of the line.

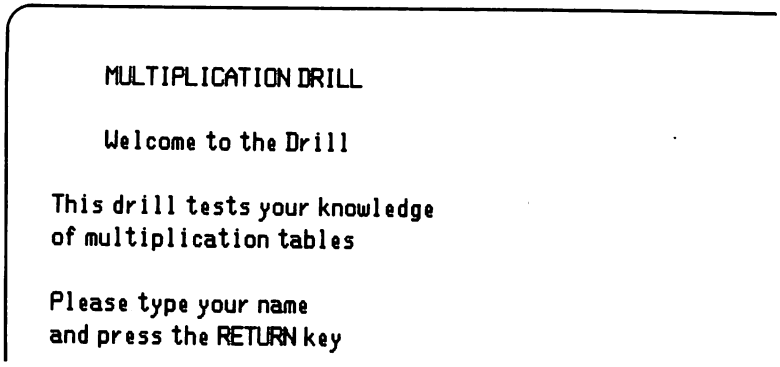


Figure 6.4
Introductory Screen

The statements that instruct the computer to display these lines appear in Figure 6.5.

Program lines 60 and 70 contain a formula used to center text on a program line; the formula is similar to one used to center text on a typewriter page. That is, you divide the length of the text to be centered by 2 and subtract that amount from the midpoint of the line. In this example the formula appears on line 70. The pieces of the formula are as follows:

- HTAB 15 represents the midpoint of the 31-character line.
- TITLE$ is the variable set in line 60 to represent the value of MULTIPLICATION DRILL.
- LEN(TITLE$) is the length of the value of TITLE$ and equals 20.
- LEN(TITLE$) / 2 is one-half the length of the title and equals 10.

When the computer executes lines 60 and 70, it applies the value of TITLE$ to the formula and determines that the title must begin at position 5 of a screen line in order to be centered. You can use this formula to center any text on a screen or printed page.

If you want the user to respond to the text displayed on the screen, you must use an INPUT statement rather than a PRINT statement. In addition, you must define the limits of a correct response, as required.

```
10  REM **MULTIPLICATION DRILL PROGRAM**
20  REM **COPYRIGHT 1983 BY PAMELA J. ROTH**
30  REM **GIVES STUDENTS PRACTICE IN ANSWERING THE 0 THROUGH 12
    MULTIPLICATION TABLES**
40  REM **SCREEN TITLE**
50  HOME
60  TITLE$ = "MULTIPLICATION DRILL"
70  HTAB 15 - LEN (TITLE$) / 2
80  VTAB 2:PRINT TITLE$
90  REM **INTRODUCTION TO MULTIPLICATION DRILL**
100 VTAB 4:HTAB 5
110 PRINT "Welcome to the Multiplication Drill"
120 VTAB 6:PRINT "This drill tests you knowledge"
130 PRINT "of multiplication tables"
140 REM **INSTRUCTS STUDENT TO TYPE NAME**
150 REM **NAME DISPLAYED INVERSE**
160 REM **ASSIGNS VARIABLE NAME$ FOR LATER USE**
170 VTAB 10: PRINT "Please type your_"
180 INVERSE: "name_"
190 NORMAL
200 INPUT "and press the RETURN key"; NAME$
```

Figure 6.5
Program Lines for Introductory Screen

(For more information, see the description of the INPUT statement in Chapter 5.)

Reversing

Other statements that you can use to enhance text on the screen include the INVERSE, SPEED=, and NORMAL statements. Again these features should be used sparingly; otherwise, they can distract the user and have the same effect as a cluttered screen (not enough white space). An example of how to put these statements into a short program appears in Figure 6.6.

The effect of this program cannot be satisfactorily illustrated in a figure. So if you are at your ADAM, use the following steps to run this program:

```
10 REM JAZZY NAMES
20 INPUT "WHAT IS YOUR NAME?_; NAME$
30 PRINT "HELLO_"; SPEED= 100
40 PRINT NAME$
60 PRINT: INVERSE: PRINT "WELCOME TO SmartBASIC": Normal
70 SPEED = 255: REM RETURN SPEED TO NORMAL
```

Figure 6.6
Program that Reverses and Slows Down Text

1. Make sure you have SAVEd what you want in memory, because you are about to clear the computer. (Type SAVE, the FILE NAME, and press the RETURN key.)
2. Type NEW and press the RETURN key.
3. Type the program shown in Figure 6.6.
4. Type RUN and press the RETURN key.
5. Respond to the prompt in the program and press the RETURN key.

Using Subroutines to Place Screen Instructions in a Program

If you are creating several screens for a program, you will probably want to repeat the screen title and, possibly, other information. To save yourself a lot of time, create a subroutine with the text that appears on every screen, and use the GOSUB statement or ON . . . GOSUB statement to direct the computer to execute the program lines in the subroutine. Figure 6.7 shows the program lines for the Introductory Screen with subroutines.

Subroutine 1000 directs the computer to display a centered screen title and then RETURN to the end of the GOSUB 1000 statement for the next instruction (line 40 to line 50). Subroutine 2000 directs the computer to set up the Introductory Screen and then RETURN to the end of the GOSUB 2000 statement for the next instruction (line 50 to line 60). As you will

```
10   REM **MULTIPLICATION DRILL PROGRAM**
20   REM **COPYRIGHT 1983 BY PAMELA J. ROTH**
30   REM **GIVES STUDENTS PRACTICE IN ANSWERING THE 0 THROUGH 12
     MULTIPLICATION TABLES**
40   GOSUB 1000:REM **SCREEN TITLE**
50   GOSUB 2000:REM **INTRODUCTION TO MULTIPLICATION DRILL**
60   END
1000 REM **TITLE SUBROUTINE**
1010 HOME
1020 TITLE$ = "MULTIPLICATION DRILL"
1030 HTAB 15 - LEN(TITLE$) / 2
1040 VTAB 2:PRINT TITLE$
1050 RETURN
2000 REM **INTRODUCTION TO MULTIPLICATION DRILL**
2010 VTAB 4:HTAB 5
2020 PRINT "Welcome to the Multiplication Drill"
2030 VTAB 6:PRINT "This drill tests your knowledge"
2040 PRINT:"of multiplication tables"
2050 REM **INSTRUCTS STUDENT TO TYPE NAME**
2060 REM **NAME DISPLAYED INVERSE**
2070 REM **ASSIGNS VARIABLE NAME$ FOR LATER USE**
2080 VTAB 10:PRINT "Please type your_"
2090 INVERSE: "name_"
2100 NORMAL
2110 INPUT "and press the RETURN key"; NAME$
2120 RETURN
```

Figure 6.7
Program Lines with Subroutines

see in Chapter 7, there is a great deal more to this program. For now, these lines should give you an idea of how subroutines are used in a program to create the appearance of the screen.

Introduction to Printout Design

Designing a printout is similar to designing a screen; you don't want to clutter the page. Of course, in a printout you cannot use the INVERSE and NORMAL statements to create special effects.

You must consider the maximum width of the print line (80 characters) and the number of lines on a page (60 on letter size, 78 on legal size). The maximum number of characters on the SmartWRITER printer is 80 characters. If you have planned to create spreadsheets and other documents with widths greater than 80 characters, you will need to adjust your plans. Table 6.1 lists the program statements and indicates which ones can be used to design screens and printouts.

Table 6.1
Screen and Printout Design Statements

Statement	*Screen Design*	*Printout Design*
PR# x	x = 0	x = 1
PRINT	Displays output on screen when PR# = 0	Prints output on the printer when PR# = 1
TEXT	Displays text screen	Not applicable
HOME	Clears screen and moves cursor to first screen position	Not applicable
SPC (x)	Moves the cursor x spaces	Moves the print head x spaces
TAB x	Moves the cursor to position x	Moves the print head to position x
HTAB x	Moves the cursor to position x	Moves the print head to position x
	(HTAB can move the cursor or print head backwards; TAB cannot.) ,	
VTAB x	Moves the cursor to line x	Moves the print head to line x
POS x	Displays position of the cursor	Not applicable

INVERSE	Reverses screen background	Not applicable
NORMAL	Reverts display from INVERSE to normal	Not applicable
SPEED=	Normal (and maximum) speed is 255. Any other speed slows down display. The slowest speed is 0.	Print speed is 120 words per minute normally. Any other speed slows the print head.

Just as you can instruct the computer to center text on the screen, you can instruct it to center text on a page. For example, to center the following line on a page:

```
This is an example of centered text.
```

you could use the instructions shown in Figure 6.8.

```
10 REM **INSTRUCTIONS TO CENTER TEXT ON PAGE**
20 PR# 1:REM CHANGE OUTPUT DEVICE TO PRINTER
30 LINE$ = "This is an example of centered text. "
40 HTAB 40 - LEN(LINE$) / 2
50 VTAB 3:PRINT LINE$
60 PR# 0
```

Figure 6.8
Program Lines Used to Center Text on a Page

Line 20 changes the current output device to the printer. Line 30 assigns variable name LINE$ to the text (string) you want centered. Line 40 is the formula used to find the starting position of the text. Line 50 indicates the line on which to print the text—in this case, line 3. The second part of line 50 tells the printer to print the value of LINE$ at the indicated location. And, finally, line 60 changes the current output device to the screen again.

For more information about planning the appearance of the screen, see the description of SmartBASIC graphics in Chapter 8.

7

Sample Program: A Multiplication Drill

This chapter guides you through the logic of a program written in SmartBASIC and explains techniques for programming, including the following:

- Designing the appearance of the screen
- Describing program lines and subroutines
- Instructing the computer to accept user responses
- Using logical operators to test for conditions
- Using loops and subroutines to repeat operations with different sets of input

The program, a multiplication drill for elementary-school-age children, is presented so that you can type in a section and test it—wherever possible—before you run the entire program. A complete set of instructions for LOADing SmartBASIC and SAVEing the sample program is also included.

LOADing SmartBASIC

Before typing the program, you must LOAD SmartBASIC by following these instructions:

1. Insert the SmartBASIC digital data pack into drive A. If you have inserted the tape correctly, the drive door will close easily.
2. To LOAD SmartBASIC into memory, find the RESET lever on top of the console or module and pull it forward. When SmartBASIC is LOADed, the screen is dark, and the cursor appears as a right bracket (]). If SmartBASIC does not LOAD after 30 seconds, pull the COMPUTER RESET lever forward again.

   NOTE: Do not RESET once you have LOADed SmartBASIC. If you do, you will reLOAD SmartWRITER, and then to use SmartBASIC, you must reLOAD SmartBASIC from the digital data pack by following the instructions above.
3. After SmartBASIC is LOADed, remove the SmartBASIC digital data pack, put it in its box, and then insert a blank digital data pack into the drive.

SAVEing the Sample Program

To SAVE the sample program after you have entered it, follow these steps:

1. Make sure a digital data pack is in the drive.
2. Type SAVE DRILL (or select a file name other than DRILL) and press the RETURN key. When the digital data pack stops running, the file is SAVEd.

WARNING

Any attempts to remove the digital data pack while it is running will probably destroy the tape. The pack may be safely removed once the console light is off.

The Sample Program: A Multiplication Drill

The sample program is based on the requirements definition in Chapter 3. The requirements are repeated in Figure 7.1, so you don't have to flip back through the pages to find them.

MULTIPLICATION DRILL

This program must allow me or anyone using it to do the following:

1. Practice multiplication tables from 0 to 12
2. Choose which table I want to practice
3. Change my mind and choose a different table before the drill begins
4. End the table that I selected and select another one
5. Have the computer display the questions on the screen

Figure 7.1
Requirements Definition for a Computerized Multiplication Drill

To provide these activities, the multiplication drill program instructs the computer to do the following:

- Display instructions on the screen for the student
- Allow the student to choose a multiplication table from 0 to 12
- Process the student's response
- Display a question for the student to answer
- Compare the student's answer with the correct answers calculated by the computer
- Display messages in response to answers
- Display the next question

- Continue until all the questions in a multiplication table are asked
- Allow the student to select another table to be tested on
- Provide an opportunity for the student to end the program, but not before he or she has completed at least one table

Screen Design

The appearance of the screen—that is, which lines the title, questions, and responses appear on and how many spaces they should be indented—is considered and designed during the functional specification stage of programming.

For this program, two screen designs are used to instruct the student and respond to the student's answers. The first screen, called the Welcome Screen, welcomes the student to the program. The second screen, called the Work Screen, is where the student does the work. Figures 7.2 and 7.3 show the screens as they appear in the functional specification for the program. Compare those figures to Figures 7.4 and 7.5, which show the screens as they appear when the program is running.

Note that the functional specification shows the complete screens. However, when the program is running, the screens show all the text only when the conditions for doing so are met.

To design the screens for the sample Multiplication Drill program, the programmer must imagine a student at the keyboard using the program. The skill required to envision a completed program as it is being designed is acquired over time. Using their imaginations is what many programmers enjoy about programming and why many of them refer to programming as an art.

The listing for the program appears as shown in Figure 7.6. Note that the _ (underline) symbol indicates where you must press the space bar to leave a space.

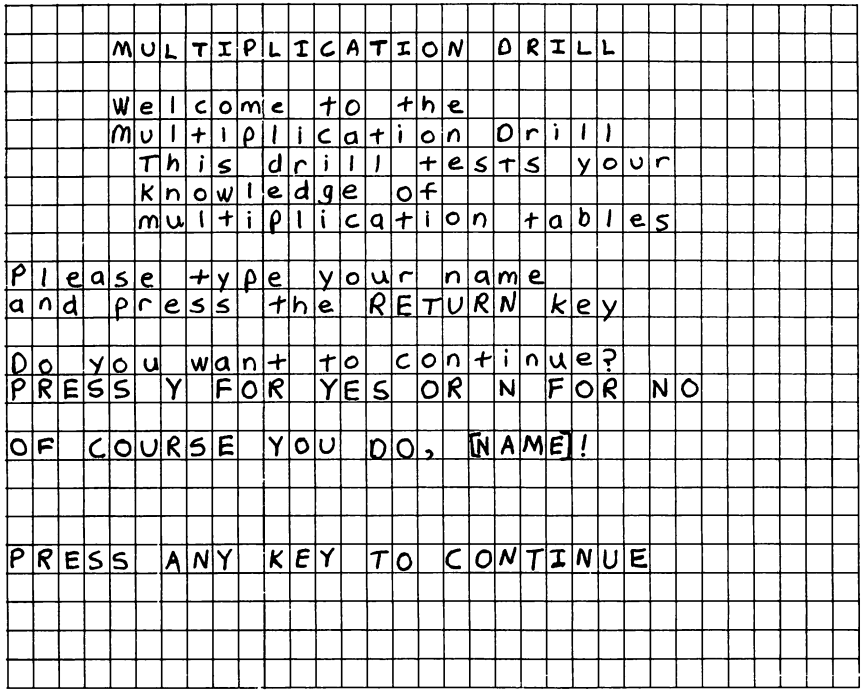


Figure 7.2
Functional Specification for
Multiplication Drill—Welcome Screen

REM statements are used throughout the program to describe the purpose of each statement. In addition to the REM statement, the following statements are used at least once in the sample program: HOME, HTAB, VTAB, RETURN, GOSUB, PRINT, INPUT, INVERSE, NORMAL, GET, FOR, NEXT, IF, THEN, and GOTO.

Figure 7.7 shows the lines that introduce the title and author and describe the purpose of the program.

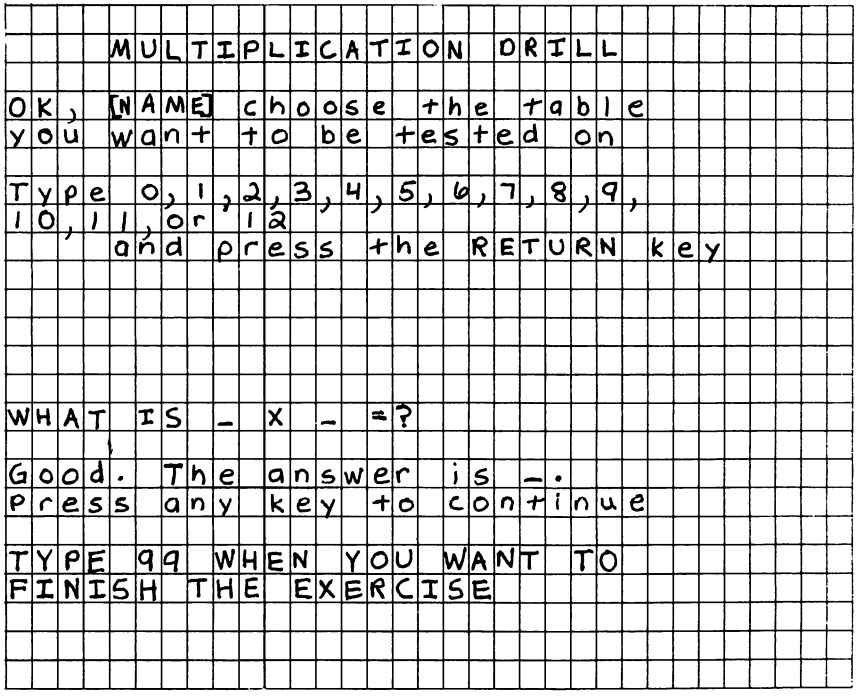


Figure 7.3
Functional Specification for
Multiplication Drill—Work Screen

Line 10 contains the REM statement that signals the computer not to execute this line. This feature allows the programmer to place comments in the program that describe the purpose of and reasoning for the statements that follow. The programmer can use the entire 239 characters allowed in a program line and place as many REM statements as desired to provide a complete description of the program.

The next sections explain the program according to the activities that occur on the Welcome Screen and Work Screen.

```
          MULTIPLICATION DRILL

      Welcome to the
      Multiplication Drill
       This drill tests your
       knowledge of
       multiplication tables

Please type your name
and press the RETURN key

Do you want to continue?
PRESS Y FOR YES OR N FOR NO

OF COURSE YOU DO, [NAME!]

PRESS ANY KEY TO CONTINUE
```

Figure 7.4
Multiplication Drill—Welcome Screen

```
       MULTIPLICATION DRILL

OK, [NAME] choose the table
you want to be tested on
Type 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, or 12
      and press the RETURN key

WHAT IS _ X _ = ?
Good. The answer is __. [Sorry, try again.]
Press any key to continue

TYPE 99 WHEN YOU WANT TO
FINISH THE EXERCISE
```

Figure 7.5
Multiplication Drill—Work Screen

```
660 HOME
670 TITLE$ = "MULTIPLICATION DRILL"
680 HTAB 15 - LEN(TITLE$)/2
690 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
700 RETURN
```

Figure 7.6
Program Listing for Multiplication Drill

```
10 REM MULTIPLICATION DRILL PROGRAM
20 REM COPYRIGHT 1983 BY PAMELA J. ROTH
30 REM GIVES STUDENTS PRACTICE IN ANSWERING THE 0 THROUGH 12
   MULTIPLICATION TABLES
```

Figure 7.7
REM Statements to Introduce the Program

Welcome Screen

If the Welcome Screen looks familiar, that's because it is a slightly different version of the Introductory Screen described in Chapter 6. Here you will see how to use it in a complete program.

The first thing that the student sees on the Welcome Screen is a screen-title line identifying the program. The screen title is centered on the second line of the screen. Instructions for placement of the screen title have been placed in a subroutine because they will be used repeatedly.

Figure 7.8 shows the lines in the order that the program executes them.

The REM statements in lines 40 and 650 describe the purpose of the lines that appear after the statements.

In line 50 the programmer instructs the computer to find the subroutine that begins at line 650. The computer responds to the GOSUB statement by executing line 650 and each line after it until the computer executes a line with a RETURN statement. The RETURN statement in line 700 instructs

```
40  REM SCREEN TITLE
50  GOSUB 650
650 REM **TITLE SUBROUTINE**
660 HOME
670 TITLE$ = "MULTIPLICATION DRILL"
680 HTAB 15 - LEN(TITLE$)/2
690 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
700 RETURN
```

Figure 7.8
Screen Title Subroutine and Use of GOSUB Statement

the computer to return to the place in the program where it branched and execute the next line, which is line 60. Line 60 is not shown here because this discussion concerns only the subroutine used to instruct the computer to display the screen title. Line 60 is discussed in the next section.

The HOME statement in line 660 clears the screen and places the cursor at the beginning of screen line 1. Before instructing the computer to display anything on the screen, make sure the screen is clear. If you don't and try to overwrite a 25-character line, for example, with a 15-character line, characters from the longer line will remain on the screen.

The first position of the screen can also be represented as HTAB 1: VTAB 1; however, this instruction does not clear the screen. The HTAB statement and the number after it represent a position or column on a screen line. The VTAB statement and the number after it represent a line on the screen.

To center the title, the programmer instructs the computer to use a formula similar to the one used to center text on a typewriter line: divide the length of the text to be centered by 2 and subtract that amount from the midpoint of the line. In this example, the formula appears on line 680. The pieces of the formula are as follows:

- HTAB 15 represents the midpoint of the 31-character screen line

- TITLE$ is the variable set in line 450 to represent the value of MULTIPLICATION DRILL
- LEN(TITLE$) is the length of the value of TITLE$ and equals 20
- LEN(TITLE$) / 2 is one-half of the length of the title and equals 10

Therefore, HTAB 15 - LEN(TITLE$) / 2 is 5. When the computer executes lines 670 and 680, it applies the value of TITLE$ to the formula and determines that the title must begin at position 5 of a screen line in order to be centered. You can use this formula to center any text on a screen or printed page.

NOTE: When using this formula to center text on a printed page (printout), remember that the maximum length of a line printed on the SmartWRITER printer is 80 characters. Therefore, the midpoint of a printed line is 40. In the HTAB 15 statement, substitute 40 for 15.

This formula can be used in other languages and on other machines. All you need to do is translate the SmartBASIC statements into the statements used in the other languages and substitute the number that represents the correct screen width (or line length).

Line 690 has four instructions separated by colons. The first instruction, VTAB 2, tells the computer to go to screen line 2, and the second instruction tells it to switch to inverse video. The third instruction directs the computer to print the title according to the instructions received so far, and, finally, the NORMAL command sets the video back to normal.

The programmer wanted to (a) welcome the student to the program and (b) give a brief description of it. The lines used to instruct the computer to display this information appear in Figure 7.9.

The REM statement in line 60 describes the purpose of these lines.

The VTAB and HTAB statements separated by a colon in line 70 are two separate instructions that tell the computer to move the cursor to position 8 on screen line 6. Whenever you want to combine statements on a program line, you must use a colon to indicate that each part of the line is to be executed separately.

```
60 REM INTRODUCTION TO MULTIPLICATION DRILL
70 VTAB 6: HTAB 8
80 PRINT "Welcome to the": TAB(5); "Multiplication Drill"
90 PRINT: PRINT "This drill tests your knowledge": PRINT
   "knowledge of multiplication tables"
```

Figure 7.9
Introduction to the Multiplication Drill

According to line 80, the computer prints "Welcome to the Multiplication Drill. " Line 90 instructs the computer to print "This drill tests your knowledge of multiplication tables" on screen lines 8 and 9.

Notice that line 70 tells the computer on which line to print the text, but lines 80 and 90 do not. SmartBASIC tells the computer to execute each PRINT statement on the next screen or page line, so the programmer did not need to provide an instruction.

Next, the programmer wanted the student to begin interacting with the program and also wanted to add variety to the program. She incorporated both of these desires in the "Please type your name" prompt. The lines used to instruct the computer are shown in Figure 7.10. Remember that a REM statement is used to describe the purpose of a line.

```
110 REM **INSTRUCTS STUDENT TO TYPE NAME**
120 REM **NAME DISPLAYED INVERSE**
130 REM **ASSIGNS VARIABLE NAME$ FOR LATER USE**
140 PRINT: PRINT "Please type your_"; : PRINT "NAME"; :NORMAL
150 PRINT " and press"
170 INPUT "the RETURN key "; NAME$
```

Figure 7.10
Instructions for Name Prompt

The first part of line 140 instructs the computer to move the cursor to screen line 10. The second part begins a print instruction. The _ (underline) symbol is used to indicate where you must press the space bar to leave a space. A space rather than this symbol appears when the space bar is pressed. A space must be left; otherwise, the words *your* and *name* will appear as "yourname."

The INVERSE statement in line 140 reverses the coloring (on color monitors) or shading (on black-and-white or black-and-green monitors) of every string in each subsequent PRINT statement until the computer reaches a NORMAL statement. To be certain that the word *name* is displayed on the same line as the words on program line 150, a semicolon is placed after it.

The NORMAL statement in line 140 instructs the computer to return to normal display. If you would like to see what inverse and normal look like without typing the entire program, type in lines 140, 150, 160, and 170. Remember to press the RETURN key at the end of each line. Then type RUN and press the RETURN key after line 170.

In the first part of line 170—up to the semicolon—the programmer uses the INPUT statement instead of the PRINT statement to tell the computer to accept input in response to the prompt. The INPUT statement causes text between the quotation marks to be printed on the line after the last executed statement.

In the second part of line 170, the programmer uses the semicolon (;) to indicate that the input received from the keyboard is to be read as the value of the NAME$ variable.

After the student types his or her name and presses the RETURN key, the computer stores the name as the value of the variable NAME$. The NAME$ variable is used later in the program.

The next group of lines, 180 through 290, instructs the computer to display the "Do you want to continue?" prompt and respond to the student's answer. These lines are shown in Figure 7.11.

Line 190 instructs the computer to display the "Do you want to continue?" prompt on screen line 14. Line 200 tells the computer to display "PRESS Y FOR YES OR N FOR NO" on the next screen line, which is screen line 15.

```
180 REM **INSTRUCTIONS FOR CONTINUE PROMPT**
190 PRINT: PRINT "Do you want to continue?"
200 PRINT "PRESS Y FOR YES OR N FOR NO";
210 GET A$
220 HTAB 1 VTAB 22
230 PRINT "OF COURSE YOU DO, _"; NAME$; "!"
250 VTAB 23
260 INVERSE
270 PRINT "PRESS ANY KEY TO CONTINUE"
280 NORMAL
290 GET Z$
```

Figure 7.11
Instructions for the Continue Prompt

In line 210, the GET statement and the A$ variable tell the computer to receive the response—one keystroke—without waiting for the student to press the RETURN key.

Notice that regardless of the student's answer, Y or N, the computer gives the same response. This is not an oversight by the programmer, just some fun. The instructions for the response are described in the next paragraph.

In lines 220 and 230, instructions for the following activities are given:

- Move the cursor to screen line 22 (VTAB 22)
- Display the text in quotation marks and insert the value of the variable NAME$, which is whatever the student answered in response to the NAME$; "!")

Lines 250 through 290 instruct the computer to display the "PRESS ANY KEY TO CONTINUE" prompt on line 270 in inverse and then return to normal screen display (line 280). In line 290, the GET Z$ instruction is used to accept that response.

The Work Screen

The programmer instructs the computer to clear the screen and calls in the screen title subroutine. Figure 7.12 shows the program lines in the order in which the computer executes them.

```
300 REM **INSTRUCTIONS FOR WORK SCREEN**
310 REM **CLEAR SCREEN AND DISPLAY TITLE BY USING TITLE SUBROUTINE
    AND GOSUB STATEMENT**
320 GOSUB 660
650 REM **TITLE SUBROUTINE**
660 HOME
670 TITLE$ = "MULTIPLICATION DRILL"
680 HTAB 15 - LEN(TITLE$)/2
690 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
700 RETURN
```

Figure 7.12
Screen Title Subroutine

The computer executes line 330 after executing the subroutine. The first prompt on the Work Screen asks the student to select a multiplication table and appears as shown in Figure 7.13.

Lines 330 through 370, which instruct the computer to display the table selection prompt, are shown in Figure 7.14.

Line 340 instructs the computer to use the value of the NAME$ variable in the prompt that it is to display on line 4. Lines 350, 360, and 370 instruct the computer to display the text on screen lines 8, 9, 20, and 21.

Lines 380 and 390, shown in Figure 7.15, test for student responses.

According to line 380, the computer continues to line 400 if the student's response is greater than or equal to 0 and less than or equal to 12.

If the student's response is between 12 and 99 or greater than 99, the "Please choose a times table from 0 to 12" prompt appears on screen line 14 as instructed in program line 390 and shown in Figure 7.16.

```
OK, Bev, choose the table
you want to be tested on

Type Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9,
1Ø, 11, or 12

    and press the RETURN key

TYPE 99 WHEN YOU WANT TO
FINISH THE EXERCISE
```

Figure 7.13
Table Selection Prompt

```
330 REM **INSTRUCTIONS TO DISPLAY TABLE SELECTION PROMPT**
340 VTAB 4: HTAB1: PRINT "OK, _"; NAME$; "_choose the table": PRINT
    "you want to be tested on. "
350 VTAB 8: PRINT "Type Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, ": PRINT TAB(6);
    "1Ø, 11, or 12"
360 VTAB 2Ø: PRINT "TYPE 99 WHEN YOU WANT TO": PRINT "FINISH THE
    EXERCISE. "
370 VTAB 11: INPUT "and press the RETURN key "; table
```

Figure 7.14
Table Selection Instructions

```
380 ON TABLE>=Ø AND TABLE<=12 GOTO 4ØØ: IF table=99 THEN RUN
390 VTAB 14: PRINT "Please choose a times table": PRINT "from Ø to
    12": GOTO 350
```

Figure 7.15
Instructions to Test for Conditions

```
OK, Bev, choose the table
you want to be tested on

Type Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9,
1Ø, 11, or 12
     and press the RETURN key

Please choose a times table
from Ø to 12

TYPE 99 WHEN YOU WANT TO
FINISH THE EXERCISE
```

Figure 7.16
Please Choose a Times Table Prompt

If the student's response is 99, then the computer is instructed by line 380 to start at the beginning of the program.

The programmer uses the student's answer in a FOR . . . NEXT loop to test the student's knowledge of the times table he selected. Lines 400 through 530, which are the heart of this program, are shown in Figure 7.17.

The FOR statement in line 430 indicates that the instructions in the loop should be followed for the values of S, which equal Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1Ø, 11, and 12; for a total of 13 passes through the loop. Line 450 instructs the computer to execute the screen title subroutine, which again clears the screen and centers the title on the second screen line.

Line 470 instructs the computer to display a question and assigns the variable name ANSWER to the response. The correct answer is composed of the value of the table (whichever table the student chose to be tested on) multiplied by the value of S (which is determined by how many times the FOR . . . NEXT loop has been executed).

Line 480 tests the answer. If the student answers incorrectly, the instructions in line 480 are followed, and the display appears as shown in Figure 7.18. The "Sorry, try again" message appears, and the

```
400 REM **INSTRUCTIONS FOR GIVING QUESTIONS AND RESPONDING TO
    ANSWERS**
410 REM **INITIALIZING FOR . . . NEXT LOOP**
430 FOR S =0 TO 12
440 REM **CLEAR SCREEN AND DISPLAY TITLE BY USING TITLE SUBROUTINE
    AND GOSUB STATEMENT**
450 GOSUB 660
460 REM **INSTRUCTIONS FOR DISPLAYING TEST QUESTION AND
    PROCESSING STUDENT' S ANSWER**
470 VTAB 15: PRINT "WHAT IS_"; TABLE; "_X_" S; "="; INPUT ""; ANSWER
480 IF ANSWER <> TABLE * S THEN VTAB 17: PRINT "Sorry, try again":
    FOR W=0 TO 1000: NEXT: VTAB: PRINT: GOTO 470
500 VTAB 17: PRINT "Good. The answer is_"; ANSWER; "."
510 PRINT "Press any key to continue_";
520 GET A$
530 NEXT
530 NEXT
```

Figure 7.17
FOR . . . NEXT Loop Instructions

question is redisplayed. The GOTO 470 statement, which means go to program line 470, instructs the computer to display and test the answer to this question again.

If the student answers correctly, the instructions in line 500 and 510 are followed, and the display appears as shown in Figure 7.19.

```
WHAT IS 3 X 4 = ? 7

Sorry, try again
```

Figure 7.18
Appearance of Work Screen in Response to Incorrect Answer

```
WHAT IS 3 X 4 = ? 12

Good. The answer is 12.
Press any key to continue
```

Figure 7.19
Appearance of Work Screen in Response to Correct Answer

When the student presses a key to continue, the loop begins again, and the next value is placed in the question that appears on the screen. The loop continues until 13 questions have been correctly answered; then the student is asked if he or she wants to try another table.

Lines 540 through 630, which instruct the computer to display the "Do you want to try another table?" prompt and respond to the answer, are shown in Figure 7.20.

Line 560 instructs the computer to execute the screen-title subroutine that clears the screen and displays the title on screen-line 2. Lines 580 through 600 use the structure discussed earlier to provide instructions to

```
540 REM **INSTRUCTIONS TO SELECT ANOTHER TABLE AFTER COMPLETING
    ONE**
550 REM **CLEAR SCREEN AND DISPLAY TITLE USING SUBROUTINE AND
    GOSUB STATEMENT**
560 GOSUB 660
570 REM **INSTRUCTIONS FOR DISPLAYING PROMPT AND RESPONDING TO
    STUDENT' S ANSWER**
580 VTAB 15: PRINT "Want to try another table?"
590 PRINT "TYPE Y FOR YES OR N FOR NO_";
600 GET B$
610 IF B$ = "Y" OR B$ = "Y" GOTO 340
620 IF B$ = "N" OR B$ = "N" GOTO 10
630 GOTO 560
640 END
```

Figure 7.20
Instructions to Select Another Table

the student and accept the response. Lines 610, 620, and 630 evaluate the response. If the student answers Y, the computer is sent to line 340, where it displays a prompt that asks the student to select a table.

If the student answers N, the computer is sent to the beginning of the program. When the student sees the Welcome Screen again, he or she can let someone else use the program, LOAD another program, RESET SmartWRITER, play a game, or turn ADAM off for the day.

If the student answers something other than Y or N, the computer is sent to line 560, which contains instructions to clear the screen and display the prompt again.

You've been through the entire program. Another version appears next to show you how the same program looks under a more structured approach.

Variation on a Theme

As mentioned earlier, different program instructions can be used to perform the same tasks. The multiplication drill, for example, does the same activities when structured as shown in Figure 7.21.

```
10   REM **MULTIPLICATION DRILL PROGRAM**
20   REM **COPYRIGHT 1983 BY PAMELA J. ROTH**
30   REM **GIVES STUDENTS PRACTICE IN ANSWERING THE 0 THROUGH 12
     MULTIPLICATION TABLES**
40   GOSUB 4000: REM **SCREEN TITLE**
50   GOSUB 1000: REM **WELCOME SCREEN**
60   GOSUB 4000: REM **SCREEN TITLE**
70   GOSUB 2000: REM **WORK SCREEN**
80   GOSUB 4000: REM **SCREEN TITLE**
90   GOSUB 3000: REM **SELECT ANOTHER TABLE
100  IF B$="Y" OR B$="Y" GOTO 60
110  END
1000 REM **WELCOME SCREEN**
1010 REM **INTRODUCTION TO MULTIPLICATION DRILL**
1020 VTAB 4: HTAB 8
```

```
1030 PRINT "Welcome to the": PRINT TAB(5); " Multiplication
     Drill"
1040 PRINT: PRINT "This drill tests your knowledge": PRINT "of
     multiplication tables. "
1060 REM **INSTRUCTS STUDENT TO TYPE NAME**
1070 REM **NAME DISPLAYED INVERSE**
1080 REM **ASSIGNS VARIABLE NAME$ FOR LATER USE**
1090 PRINT: PRINT "Please type your_";
1100 INVERSE: PRINT "NAME"; : NORMAL
1110 PRINT "_and press"
1120 INPUT "the RETURN key_"; NAME$
1130 REM **INSTRUCTIONS FOR CONTINUE PROMPT**
1140 PRINT: PRINT "Do you want to continue?"
1150 PRINT "PRESS Y FOR YES OR N FOR NO";
1160 GET A$: IF A$="N" OR A$="N" THEN END
1170 VTAB 22: HTAB 1
1180 PRINT "OF COURSE YOU DO, _"; NAME$; "!"
1200 VTAB 23
1210 INVERSE
1220 PRINT "PRESS ANY KEY TO CONTINUE_";
1230 NORMAL
1240 GET Z$: RETURN
2000 REM **WORK SCREEN**
2010 REM **INSTRUCTIONS TO DISPLAY TABLE SELECTION PROMPT**
2020 VTAB 4: PRINT "OK_, "; NAME$; "_choose the table": PRINT "you
     want to be tested on"
2030 VTAB 8: PRINT "Type 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ": PRINT TAB(6);
     "10, 11, or 12"
2040 VTAB 20: PRINT "TYPE 99 WHEN YOU WANT TO": PRINT "_FINISH THE
     EXERCISE"
2050 VTAB 11: INPUT "and press the RETURN key "; table
2060 ON TABLE>=0 AND TABLE<=12 GOTO 2110: IF TABLE = 99 THEN RUN
2070 VTAB 14: PRINT "Please choose a times table": PRINT "from 0 to
     12": GOTO 2030
2080 REM **INSTRUCTIONS FOR GIVING QUESTIONS AND RESPONDING TO
     ANSWERS
2090 REM **INITIALIZING FOR . . . NEXT LOOP**
2100 S=0
2110 FOR S=0 TO 12
```

```
2120 REM **CLEAR SCREEN AND DISPLAY TITLE BY USING TITLE
     SUBROUTINE AND GOSUB STATEMENT**
2130 GOSUB 4000
2140 REM **INSTRUCTIONS FOR DISPLAYING TEST QUESTION AND
     PROCESSING STUDENT'S ANSWER**
2150 VTAB 15: PRINT "WHAT IS_"; TABLE; "_X_"; S; "_=_"; :INPUT "";
     ANSWER
2160 IF ANSWER<>TABLE*S THEN VTAB 17: PRINT "Sorry, try again":
     FOR W=0 TO 1000: NEXT: VTAB 17: PRINT: GOTO 2150
2170 VTAB 17: PRINT "Good. The answer is_"; ANSWER; "."
2180 PRINT "Press any key to continue";
2190 GET A$
2200 NEXT: RETURN
3000 REM **SELECT ANOTHER TABLE**
3010 REM **INSTRUCTIONS FOR DISPLAYING PROMPT AND RESPONDING TO
     STUDENT'S ANSWER**
3020 VTAB 15: PRINT "Want to try another table?"
3030 PRINT "TYPE Y FOR YES OR N FOR NO";
3040 GET B$
3050 IF NOT (B$ = "Y" OR B$ ="Y" OR B$ ="N" OR B$ = "N" THEN PRINT:
     GOTO 3020
3060 RETURN
4000 REM **SCREEN TITLE**
4010 HOME
4020 TITLE$ = "MULTIPLICATION DRILL"
4030 HTAB 15 - LEN(TITLE$)/2
4040 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
4050 RETURN
```

Figure 7.21
Program Listing for Multiplication Drill

That's it; you've been through the entire program! If you are interested in going through two more programs, see Chapter 9.

8

SmartBASIC Graphics

Used extensively in games and, to a lesser degree, in most commercial programs, screen graphics are shapes and drawings that enhance screen design. The programming instructions for screen graphics, like the instructions for any type of program, must be detailed and step by step.

The following topics are discussed in this chapter:

- Introduction to graphics
- Low-resolution graphics
- High-resolution graphics
- Introduction to motion

Introduction to Graphics

Graphic screen design consists of combinations of color, vertical points, and horizontal points. A set of one vertical point and one horizontal point is used to indicate one block on the screen. In addition, you can use low-resolution or high-resolution graphics.

Because of the differences between ADAM and Apple, some modifications are necessary if you want to use Applesoft BASIC graphics. For example, the PEEK and POKE statements used in some Applesoft programs cannot be translated into PEEK and POKE statements in SmartBASIC. If you find a graphics program with PEEK and POKE, you will want to rework the program to operate without those statements.

Low resolution in SmartBASIC means the screen has 1600 blocks in 40 columns by 40 rows, plus 4 lines for text at the bottom of the screen.

High resolution in SmartBASIC means page 1 of the high-resolution screen has 44,800 points in 280 columns by 160 rows, with 4 lines for text at the bottom of the screen. Page 2 of the SmartBASIC high-resolution screen has 53,760 blocks in 280 columns by 192 rows. All of page 2 is used for graphics.

Low-resolution graphics and high-resolution graphics can be displayed in 16 colors.

Whether you are creating low- or high-resolution graphics, each program line can contain instructions for color, horizontal points, and vertical points. Further, you can create subroutines for such graphics activities as moving an object. In the tables that follow, several graphics functions are listed with their corresponding SmartBASIC statements. Table 8.1 shows the statements used in low-resolution graphics, and Table 8.2, the statements used in high-resolution graphics.

Low-Resolution Graphics

Rules for Design

Rules to remember when designing low-resolution graphics are

- The screen is 40 columns wide, and the columns are labeled 0-39.
- The screen is 40 rows high, and the rows are labeled 0-39.
- Blocks are different from lines. The PLOT statement tells ADAM where to place a block. The HLIN

Table 8.1
Low-Resolution Graphics Statements

Activity	*Statement*
Change color	COLOR=
Place current color in a specified block	PLOT
Draw a horizontal line in the current color	HLIN
Draw a vertical line in the current color	VLIN
Find position specified on low-resolution graphics screen	SCRN

Table 8.2
High-Resolution Graphics Statements

Activity	*Statement*
Change to page 1 of high-resolution graphics	HGR
Change to page 2 of high-resolution graphics	HGR2
Indicate high-resolution screen color	HCOLOR=
Indicate location of block on high-resolution screen	HPLOT

statement tells ADAM where to place a horizontal line. The VLIN statement tells ADAM where to place a vertical line.

- Each instruction for a block or line must also specify the column and row where the block or line should appear. The number that is used to indicate the column or row is called a coordinate.

- A line drawn between a coordinate in one row and a coordinate in another row is a vertical line or screen line (see the explanation of the VLIN statement). A line drawn between a coordinate in one column and a coordinate in another column is a horizontal line (see the explanation of the HLIN statement).
- 16 colors are available through use of the COLOR= statement (0-15).
- The last four lines of the screen are reserved for text. That is, you can instruct the computer to display text on the last four lines, but you cannot instruct it to draw lines or plot blocks in that area.

Figure 8.1 shows an example of a grid used to design low-resolution graphics. Notice that the columns are numbered 0-39 and the rows are numbered 0-39. The coordinates for the first block are (0,0), and the coordinates for the last block are (39,39).

Low-Resolution Statements Defined

The low-resolution graphics screen is different from the text and high-resolution graphics screen. You must use the GR statement to change to the low-resolution screen.

GR

The GR statement tells ADAM to change from a text screen to a graphics screen. The screen clears to black, the cursor moves to the beginning of the last screen line, and any text window that may have been set is cleared.

COLOR=

The screen is black until you tell the computer to change it. The COLOR= statement sets or changes the color. You can change the color by using this statement anywhere in a low-resolution graphics program. The 16 colors available in low-resolution graphics are indicated by using 0-15 as shown in Table 8.3.

The = (equals) symbol is part of the statement. Do not leave a space between COLOR and = : COLOR=. Although you cannot assign a value

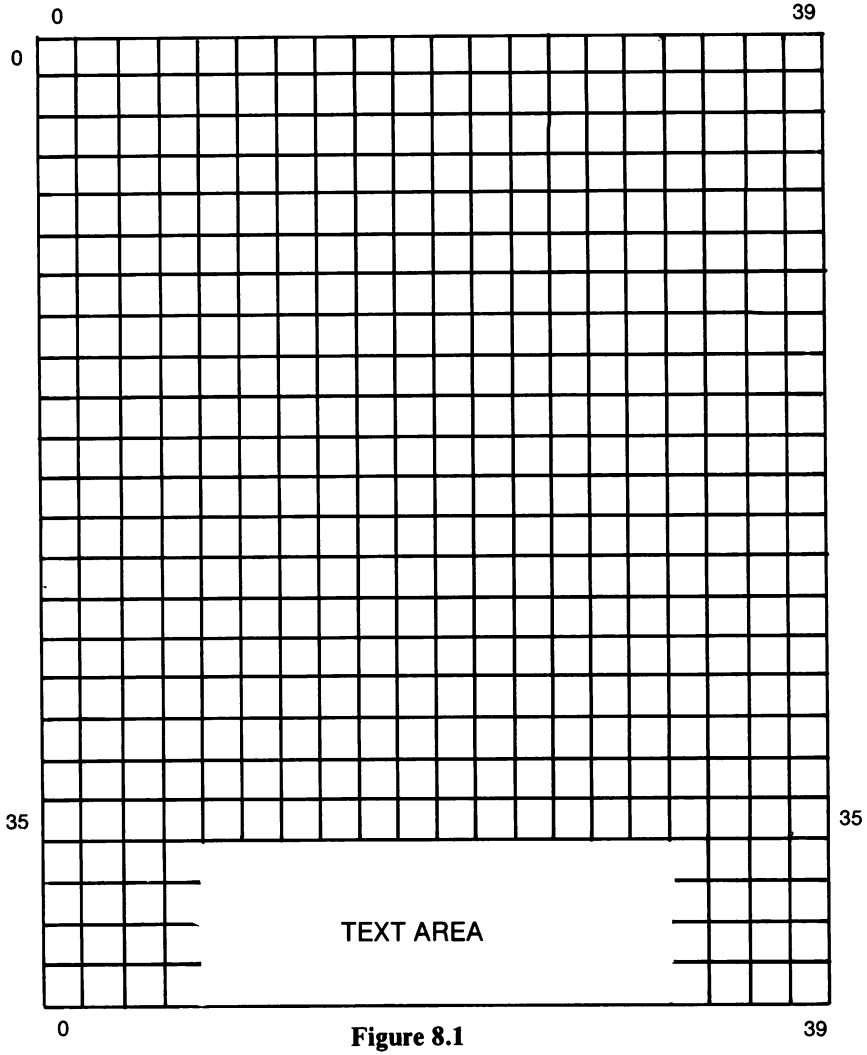


Figure 8.1
Low-Resolution Graphics Grid

to this statement, you can create a FOR . . . NEXT loop in which the instructions are repeated for two or more colors, as shown in Figure 8.2.

Line 100 tells ADAM to change to a low-resolution graphics screen. Line 120 initiates C as colors. Line 110 tells ADAM that the following instructions are to be executed for each color. Lines 130, 140, and 150

Table 8.3
Low-Resolution Colors

Color	*Number*	*Color*	*Number*
black	0	light yellow	8
magenta	1	medium red	9
dark blue	2	dark gray	10
dark red	3	pink	11
dark green	4	light green	12
light gray	5	light yellow	13
medium green	6	light blue	14
light blue	7	white	15

```
100 GR
110 FOR C = 0 TO 15
120 COLOR= C
130 PLOT 17, 11 : PLOT 17, 12 : PLOT 17, 13
140 PLOT 18, 11 : PLOT 18, 12 : PLOT 18, 13
150 PLOT 19, 11 : PLOT 19, 12 : PLOT 19, 13
160 NEXT C
```

Figure 8.2
Example of Low-Resolution Colors in FOR . . . NEXT Loop

instruct the computer to color the blocks indicated by the PLOT statement. Line 160 tells ADAM to find the next color, if there is one, and follow the instructions.

In the explanations below, the phrase "current color" refers to the most recently set color in a program. All blocks and lines in a program appear in the current color. To change the current color, simply use the COLOR= statement before assigning coordinates for blocks and lines.

PLOT

PLOT column, row

The PLOT statement includes two numbers, which assign the low-resolution current color (see the COLOR= statement) to a specified block. The first number indicates the column, and the second number the row. Examples of PLOT statements are shown in Table 8.4.

HLIN

HLIN column, column at row

The HLIN statement is used to draw horizontal lines in the current color. Horizontal lines are drawn between columns. You must specify

Table 8.4
Examples of PLOT Statements

These Statements:	*Give These Instructions:*
PLOT 20, 2	Color block at column 20, row 2.
PLOT 0, 0	Color first block on screen. (upper-left corner)
PLOT 0, 39	Color last block in first column.
PLOT 39, 0	Color last block in first row.
PLOT 39, 39	Color last block on screen.
PLOT 0, 40	Second coordinate is outside of screen boundary. You will get an error message.
PLOT 40, 12	First coordinate is outside of screen boundary. You will get an error message.

coordinates to indicate the column where the line begins, the column where the line ends, and the row that the line is in.

The longest horizontal line on a low-resolution screen starts in column 0 and ends in column 39. Examples of statements that instruct the computer where to place horizontal lines are shown in Table 8.5.

Table 8.5
Examples of HLIN Statements

These Statements:	*Give These Instructions:*
HLIN 5, 15 AT 10	Draw a line from column 5 to column 15 in row 10 (screen line 10).
HLIN 0, 39 AT 0	Draw a line across the top of the screen.
HLIN 0, 39 AT 39	Draw a line across the bottom of the graphic screen. There are still four text lines at the bottom of the screen.
HLIN 0, 39 AT 19	Draw a line through the approximate middle of the screen.

VLIN

VLIN row, row at column

The VLIN statement is used to draw vertical lines in the current color. Vertical lines are drawn between rows. You must specify coordinates that indicate the row where the line begins, the row where the line ends, and the column that the line is in.

The longest vertical line on a low-resolution screen starts in row 0 and ends in row 39. Examples of statements that instruct the computer where to place vertical lines are shown in Table 8. 6.

SCRN

SCRN column, row

The SCRN function provides the color code (0-15) for the position specified on the low-resolution screen. Table 8.7 gives several examples of the SCRN function.

Table 8.6
Examples of VLIN Statements

These Statements:	*Give These Instructions:*
VLIN 5, 15 AT 10	Draw a line from row 5 to row 15 in column 10.
VLIN 0, 39 AT 0	Draw a line down the left side of the screen.
VLIN 0, 39 AT 39	Draw a line down the right side of the screen.
VLIN 0, 39 AT 19	Draw a line down the approximate middle of the screen.

Table 8.7
Examples of the SCRN Function

These Statements:	*Read As:*
SCRN (0,5)	Find the color at the beginning of row 5.
SCRN (3,15)	Find the color at the third position of the fifteenth row.
SCRN (5,10)	Find the color at the fifth position of the tenth row.

High-Resolution Graphics

Rules for Design

Rules to remember when designing high-resolution graphics are

- Statements described for low-resolution graphics do not work on a high-resolution graphics screen.

- The HGR screen is 280 columns wide (four times as many points as in a low-resolution graphics screen), and the columns are labeled 0-279.
- The HGR screen consists of 160 rows (four times the resolution of the low-resolution graphics screen), and the rows are labeled 0-159.
- Four lines are available at the bottom of the HGR screen for text, allowing you to create games or educational drills with instructions at the bottom of the screen.
- The HGR2 screen, page 2, is also 280 columns wide (four times as many points as in a low-resolution graphics screen), and the columns are labeled 0-279.
- The HGR2 screen does not have four lines available at the bottom for text; it consists of 192 rows, and the rows are labeled 0-191.

Figure 8.3 shows a grid used to design high-resolution graphics. Notice that the columns are numbered 0-279, and the rows are numbered 0-159. The coordinates for the first block are (0,0), and the coordinates for the last block are (279,159).

High-Resolution Statements Defined

The high-resolution graphics screen is different from the text and low-resolution graphics screens. The high-resolution screen consists of two pages. You must use the HGR statement to change to page 1 of the high-resolution screen, and the HGR2 statement to change to page 2 of the high-resolution screen.

HGR

The HGR statement is used to change to page 1 of the high-resolution screen from the text screen, low-resolution screen, or page 2 of the high-resolution screen. When the HGR statement is executed, the graphics area is cleared. The cursor does not appear unless it is on one of the four bottom lines of the screen.

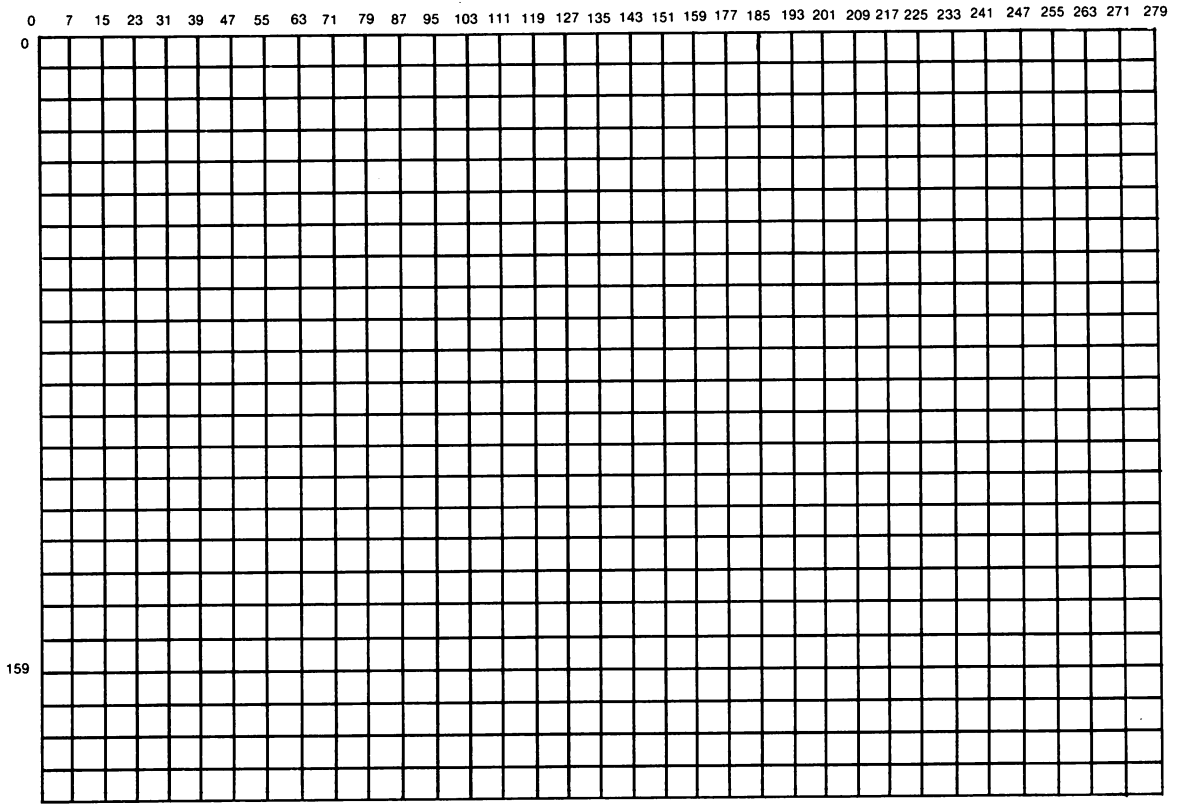


Figure 8.3
High-Resolution Graphics Grid

HGR2

The HGR2 statement is used to change to page 2 of the high-resolution screen from the text screen, low-resolution screen, or page 1 of the high-resolution screen. When the HGR2 statement is executed, the graphics area is cleared.

HCOLOR=

The screen is black until you tell the computer to change it. The HCOLOR= statement sets or changes the color. You can change the color by using this statement anywhere in a high-resolution graphics program. The 16 colors available in high-resolution graphics are indicated by using the numbers 0 through 15, as shown in Table 8.8.

Table 8.8
High-Resolution Colors

Color	*Number*	*Color*	*Number*
black-1	0	brown	8
green	1	dark blue	9
violet	2	gray	10
white-1	3	pink	11
black-2	4	dark green	12
orange	5	yellow	13
blue	6	aqua	14
white-2	7	magenta	15

The = (equals) symbol is part of the statement. Do not leave a space between HCOLOR and = : HCOLOR=. You cannot assign a value to this statement. However, you can create a FOR . . . NEXT loop in which the instructions are repeated for two or more colors, as shown in Figure 8.4.

Line 100 tells ADAM to change to page 1 of a high-resolution graphics screen. Line 120 initiates C as colors. Line 110 tells ADAM that the

```
100 HGR
110 FOR C = 0 TO 15
120 HCOLOR= C
130 HPLOT 17, 11 : HPLOT 17, 12 : HPLOT 17, 13
140 HPLOT 18, 11 : HPLOT 18, 12 : HPLOT 18, 13
150 HPLOT 19, 11 : HPLOT 19, 12 : HPLOT 19, 13
160 NEXT C
```

Figure 8.4
Example of High-Resolution Colors in FOR . . . NEXT Loop

following instructions are to be executed for each color. Lines 130, 140, and 150 instruct the computer to color the blocks indicated by the HPLOT statement. Line 160 tells ADAM to find the next color, if there is one, and follow the instructions.

HPLOT

HPLOT column, row
HPLOT column, row TO column, row
HPLOT column, row TO column, row TO column, row TO column, row

The HPLOT statement is used to plot blocks, horizontal lines, vertical lines, squares, triangles, and rectangles, among other shapes. Each block consists of a column coordinate and a row coordinate. Each line consists of a beginning block and an ending block.

In a vertical line, the first coordinate (column) of the first and last blocks are the same. The second coordinate (row) of the first block is less than the second coordinate (row) of the last block.

The longest vertical line on a high-resolution screen starts in row 0 and ends in row 159. Examples of statements that instruct the computer where to place vertical lines are shown in Table 8. 9.

In a horizontal line, the second coordinates (row) of the first and last blocks are the same. The first coordinate (column) of the first block is less than the first coordinate (column) of the last block.

Table 8.9
Examples of HPLOT Statements for Vertical Lines on Page 1 of the High-Resolution Screen

These Statements:	*Read As:*
HPLOT 10,5 TO 10,15	Draw a line from row 5 to row 15 in column 10.
HPLOT 0,0 TO 0,159*	Draw a line down the left side of the screen (*191, if page 2 of high-resolution graphics screen).
HPLOT 279,0 TO 279,159*	Draw a line down the right side of the screen (*191, if page 2 of high-resolution graphics screen).
HPLOT 127,0 TO 127,159*	Draw a line down the approximate middle of the screen (*191, if page 2 of high-resolution graphics screen).

The longest horizontal line on a high-resolution screen starts in column 0 and ends in column 279. Examples of statements that instruct the computer where to place horizontal lines are shown in Table 8.10.

A square represented by the statements shown in Figure 8.5 appears on the screen as shown in Figure 8.6.

A triangle represented by the statements shown in Figure 8.7 appears on the screen as shown in Figure 8.8.

A rectangle represented by the statements shown in Figure 8.9 appears on the screen as shown in Figure 8.10.

Motion

Using graphics to create moving cartoons is fun. When attempting to animate, keep in mind what is actually happening: the colors of lines and blocks are changed to match the background or stand out from the background. One block or point is erased and replaced by the next. When you run the program, one or more blocks and lines appear to move around the screen.

Table 8.10
Examples of HPLOT Statements for Horizontal Lines

These Statements:	*Read As:*
HPLOT 5,10 TO 15,10	Draw a line from column 5 to column 15 in row 10 (screen line 10).
HPLOT 0,0 TO 279,0	Draw a line across the top of the screen.
HPLOT 0,159* TO 279,159*	Draw a line across the bottom of the screen (*191 if page 2 of the high-resolution graphics screen).
HPLOT 0,79* TO 279,79*	Draw a line through the approximate middle of the screen (*95 if page 2 of the high-resolution graphics screen).

```
10 HGR
20 HCOLOR= 5
30 HPLOT 0, 0 TO 150, 0 TO 150, 100 TO 0, 100 TO 0, 0
```

Figure 8.5
Statements That Plot a Square

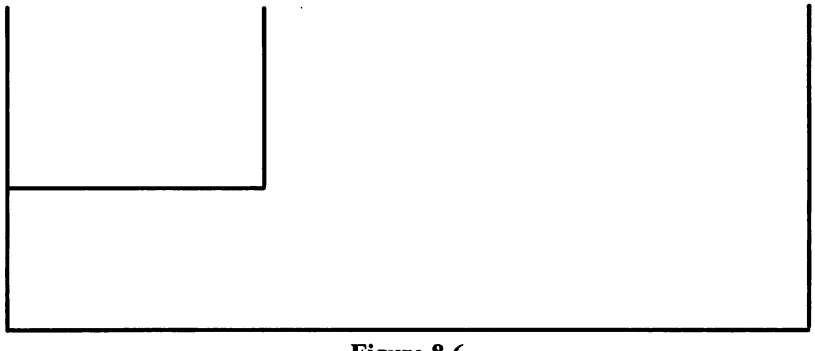


Figure 8.6
Appearance of Square on the Screen

```
10 HGR
20 HCOLOR= 7
30 HPLOT 0, 7 TO 80, 7 TO 23, 100 TO 0, 7
```

Figure 8.7
Statements That Plot a Triangle

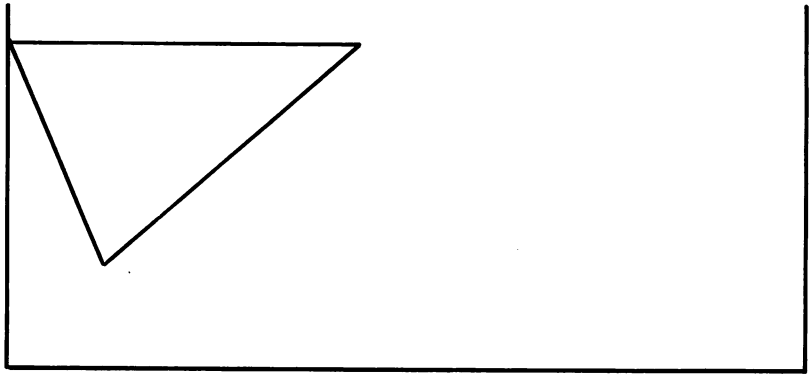


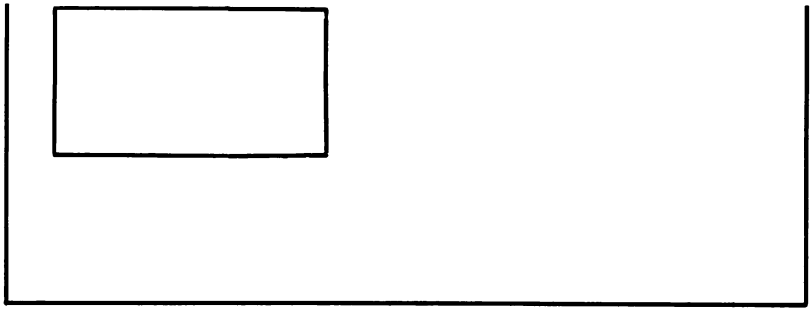
Figure 8.8
Appearance of Triangle on the Screen

```
10 HGR
20 HCOLOR= 3
30 HPLOT 7, 6 TO 100, 6 TO 100, 80 TO 7, 80 TO 7, 6
```

Figure 8.9
Statements that Plot a Rectangle

The program shown in Figure 8.11 instructs the computer to draw blocks on a low-resolution graphics screen so that one block appears to be moving around the screen.

Line 20 directs the computer to go to the subroutine that starts on line 1000 and provides the instructions for using the program. Line 40 changes from text screen to the low-resolution graphics screen. Line 70



instructs the computer to go to the subroutine that starts on line 2000. Line 80 sends the computer again to the subroutine that starts on line 1000.

The first part of line 1010 changes the screen to the text screen. Chances are, the first time this routine is executed, the screen is a text screen. However, once this subroutine and the subroutine that begins on line 2000 are run, the screen has been changed to the low-resolution graphics screen. The second part of line 1010 clears the screen (but not memory) and moves the cursor to the first position on the screen.

Line 1020 instructs the computer to print the question on screen line 10. Line 1030 instructs the computer to print the prompt screen line 14 in position 5. It also instructs the computer to assign the user's answer as the value of variable N. If the user slects zero times, then the program ends. Line 1050 RETURNs the computer to the end of the GOSUB 1000 statement.

Line 2010 initializes the FOR . . . NEXT loop and establishes the limits of the loop, which is the value of N$. Line 2020 contains the following instructions:

- Establish limits of the HT loop as 9 to 28
- Change the color to the background color (black)
- Place a block on the screen at line 9, row HT
- Change the color to light blue
- Place a block on the screen at line 9, row + 1
- Repeat for the next value of H

```
10   REM MOVE A BLOCK AROUND THE SCREEN
20   GOSUB 1000:REM INSTRUCTIONS TO USER
30   REM CHANGE TO LOW-RESOLUTION GRAPHICS SCREEN
40   GR
50   REM CHANGE COLOR
60   COLOR= 7
70   GOSUB 2000:REM DRAW BLOCKS
80   GOTO 20:REM START AGAIN
1000 REM MOVE A BLOCK AROUND THE SCREEN
1010 TEXT:HOME
1020 VTAB 10:PRINT "How many times do you want the block to go
     around the screen?"
1030 VTAB 14:HTAB 5: INPUT "Select from 0 TO 20 and press the RETURN
     key_"; N
1040 IF N<1 THEN END
1050 RETURN
2000 REM DRAW BLOCKS
2010 FOR I = 1 TO N
2020 FOR HT=9 TO 28: COLOR =0: PLOT HT 9: COLOR= 7: PLOT HT+1, 9:
     NEXT
2030 FOR VR=9 TO 28: COLOR =0: PLOT 29, VR: COLOR =7 PLOT 29, VR+1:
     NEXT
2040 FOR HB=29 TO 10 STEP -1: COLOR =0: PLOT HB 29: COLOR =7 : PLOT
     HB-1, 29: NEXT
2050 FOR VL=29 TO 10 STEP -1: COLOR =0: PLOT 9, VL: COLOR =7: PLOT 9,
     VL-1: NEXT
2060 NEXT
2070 RETURN
```

Figure 8.11
Line Game

The intent of line 2020 is to give the impression that the block across the top of the screen is moving. This illusion of motion is created by drawing a block in a color other than the background color, redrawing it in the background color, and so on with each segment until the line is complete.

Lines 2030, 2040, and 2050 have the same purpose. Line 2030 tells the computer to "move" the block down the right side of the screen; line 2040 tells the computer to move the block across the bottom of the screen; and line 2050 tells the computer to move the block up the left side of the screen.

Line 2060 starts the loop again until the line has moved around the screen N$ times (the number of times requested by the user). Line 2070 returns the computer to the end of the GOSUB 2000 statement in line 70. Line 80 is then executed, which starts the program again.

If you want to try this program, use the following steps:

1. Save the program currently in memory to a blank or partially filled digital data pack by typing SAVE and the FILE NAME, and pressing the RETURN key.
2. Type NEW and press the RETURN key to clear memory.
3. Type the program shown in Figure 8.11. Remember to press the RETURN key at the end of each program line.
4. Type RUN and press the RETURN key.
5. Answer the prompts and press the RETURN key.

For a program that uses the graphics screen, see Hangman in Chapter 9.

9
More Programs

This chapter contains two programs that you can run on ADAM:

- Hangman—a word game that displays text and graphics
- Meal Planner—a program designed to generate weekly menus

Although detailed explanations are given for each statement in the Hangman program, less specific explanations are provided for the statements in the Meal Planner program. After working through the Hangman program, you will probably appreciate the opportunity to apply what you have learned and determine the purpose of each statement yourself.

LOADing SmartBASIC and SAVEing the Programs

Before typing a program you must LOAD SmartBASIC, if you haven't already, by following these instructions:

This chapter was written with Richard L. Roth, executive vice president of InfoSoft in Norwalk, CT, and leading consultant for Coleco's SmartWRITER word processor.

1. Insert the SmartBASIC digital data pack into drive A. If the tape is inserted correctly, the drive door will close easily.
2. Find the RESET lever on top of the console or module and pull the lever forward. Remember, loading takes about 30 seconds. When SmartBASIC is LOADed, the screen is dark, and the cursor appears to the right of a right bracket (]). If SmartBASIC does not LOAD after 30 seconds, pull the RESET lever forward again.

   NOTE: Do not RESET once SmartBASIC is successfully LOADed. Resetting after SmartBASIC is LOADed will reload SmartWRITER, from which point you will have to repeat the above instructions.
3. Remove the SmartBASIC digital data pack, put it in its box, and insert a blank digital data pack into the drive.

To SAVE the sample program after you have entered it, follow these steps:

1. Make certain that a digital data pack is in the drive.
2. Type SAVE HANGMAN or SAVE MEAL (or assign other file names) and press the RETURN key. When the digital data pack stops running, the file is SAVEd.

WARNING

Any attempt to remove the digital data pack while the tape is running will probably result in destroying the tape.

Hangman

In this game the computer chooses a word, and you must guess what it is, one letter at a time. For each incorrect guess, a piece is added to the victim in the following order:

- Head
- Eyes
- Mouth
- Body
- Left arm

- Right arm
- Left leg
- Right leg

If you don't get the word on the eighth attempt, the victim is hanged and you lose. You can tell the computer to select another word or end the game. The program consists of four subroutines:

- The first subroutine, starting at line 1000, instructs the computer to display the title and the instructions for playing the game. It also sets up DATA statements (lines 1210 through 1280) and allocates space for the array called GUESS$ (line 12).
- The second subroutine, starting at line 2000, instructs the computer to change to the low-resolution graphics screen and plots the border, scaffold, brace, and limbs.
- The third subroutine, starting at line 3000, instructs the computer to evaluate each guess and take action based on the guess and how many guesses have been made.
- The fourth subroutine, starting at line 4000, instructs the computer to end the game and print one of two messages, depending on whether you won or lost the game.

The listing for this program is shown in Figure 9.1.

```
10   REM **HANGMAN A GAME**
20   GOSUB 1000: REM TITLE, GAME DESCRIPTION, AND WORDS
30   GOSUB 2000: REM GRAPHICS
40   GOSUB 3000: REM EVALUATING GUESSES
50   GOSUB 4000: REM RESPONDING TO PLAYER
1000 REM SCREEN #1
1010 REM **SCREEN TITLE**
1020 TEXT: NORMAL
1030 HOME
1040 TITLE$ = " HANGMAN A GAME"
1050 HTAB - 15 LEN(TITLE$)/2
1060 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
1070 REM GREETINGS
1080 PRINT: PRINT TAB(7); "WELCOME TO HANGMAN!"
```

```
1090 REM INSTRUCTIONS
1100 PRINT: PRINT "HOW TO PLAY HANGMAN:"
1110 PRINT: PRINT " THE OBJECT OF THE GAME IS TO SAVE THE VICTIM BY
     GUESSING THE WORD IN 8 GUESSES OR LESS. "
1120 LINE$ = "HINT: START WITH VOWELS"
1130 HTAB 15 - LEN(LINE$) / 2
1140 VTAB 18: PRINT LINE$
1150 VTAB 20: PRINT "PRESS ANY KEY TO CONTINUE";
1160 GET A$
1170 REM DATA STATEMENTS WITH WORDS
1180 RESTORE
1190 FOR W = 1 TO RND(1)*34+1: READ WRD$
1200 NEXT
1210 DATA "BOOK", "TIGER", "TEDDYBEAR", "THOROUGH"
1220 DATA "DILEMMA", "ACTUALLY", "ELEPHANT", "ELEVATOR"
1230 DATA "HOUSEHOLD", "WISCONSIN", "AVIATOR", "ALLIGATOR"
1240 DATA "ABILITY", "SOCKS", "COMPUTER", "BARBARIAN"
1250 DATA "LIBRARY", "PERSONNEL", "POPCORN", "RADIATOR"
1260 DATA "ELEMENTARY", "INTERMEDIATE", "ADVANCED", "NOVICE"
1270 DATA "TELEVISION", "THEATRE", "MOVIE", "BALLET", "OPERA"
1280 DATA "ANALYST", "PROGRAMMER", "SYSTEM", "PRINTER", "MODEM"
1300 WL = LEN(WRD$)
1310 FOR I = 1 TO WL
1320 GUESS$(I) = "-": NEXT
1330 GUESSED$ = " ": RETURN
2000 REM SCREEN #2 ** GRAPHICS **
2010 GR: COLOR=13: REM CHANGE COLOR
2020 REM BORDER AROUND SCREEN
2030 HLIN 1, 39 AT 0: HLIN 1, 39 AT 38: VLIN 0, 39 AT 39: VLIN 0, 39 AT 0
2040 REM SCAFFOLD
2050 HLIN 7, 35 AT 29: VLIN 8, 36 AT 6: HLIN 7, 22 AT 8
2060 REM BRACE
2070 PLOT 7, 12: PLOT 8, 11: PLOT 9, 10: PLOT 10, 9: PLOT 21, 9: PLOT
     21, 10
2080 RETURN: REM BODY PARTS
2090 COLOR=3: HLIN 20, 23 AT 11: HLIN 20, 23 AT 17: VLIN 13, 16 AT 18:
     VLIN 13, 16 AT 24:
2092 PLOT 19, 12: PLOT 23, 12: PLOT 19, 16: PLOT 23, 16: RETURN
2100 COLOR=2: PLOT 20, 14: PLOT 20, 13: PLOT 22, 13: PLOT 22, 14:
     RETURN
```

```
2110 COLOR=12: PLOT 20, 16: PLOT 21, 15: PLOT 22, 16: RETURN
2120 COLOR=7: VLIN 18, 24 AT 21: RETURN
2130 COLOR=6: HLIN 18, 21 AT 20: VLIN 20, 22 AT 18: RETURN
2140 COLOR=14: HLIN 22, 25 AT 20: VLIN 20, 23 AT 24: RETURN
2150 COLOR=6: PLOT 20, 24: PLOT 19, 25: HLIN 17, 19 AT 26: RETURN
2160 COLOR=14: PLOT 22, 24: PLOT 23, 25: HLIN 24, 26 AT 26: RETURN
3000 REM EVALUATING GUESSES
3010 HOME: VTAB 21: PRINT "ANSWER: ";
3020 WON%=1: FOR I=1 TO WL
3030 PRINT GUESS$(I); : IF GUESS$(I)="-" THEN WON%=0
3040 NEXT: PRINT
3050 IF WON%=1 GOTO 3190
3060 PRINT "ATTEMPTS: "; GUESS$
3070 PRINT: PRINT "TAKE A GUESS: ";
3080 GET ANSWER$
3090 REM ANSWER OUT OF RANGE
3100 IF ANSWER$ < "A" OR ANSWER$ > "Z" THEN 3000
3110 REM GUESSED ALREADY
3120 RC%=0: FOR I = 1 TO WL
3130 IF MID$(GU$, I, 1)=AN$ THEN HTAB 1: PRINT "YOU GUESSED THAT
     ALREADY . . . ": I=WL: RC%=1: FOR PAUSE=0 TO 2000: NEXT
3150 IF MID$(WRD$, I, 1) = ANSWER$ THEN GUESS$(I) = ANSWER$: RC% = 1
3160 NEXT: IF RC%=1 THEN 3010
3170 GUESS$ = GUESS$ + ANSWER$: ON LEN(GUESS$) GOSUB 2090, 2100,
     2110, 2120, 2130, 2140, 2150, 2160
3180 IF LEN(GU$) < 8 THEN 3010
3190 RETURN
4000 REM **END THE GAME**
4010 FOR I = 0 TO 39: COLOR= RND(1)*15+1: HLIN 0, 39 AT I: NEXT
4020 FOR PAUSE=0 TO 1500: NEXT
4030 TEXT: HOME: VTAB 3
4050 IF WON% THEN PRINT "YOU WON"
4060 IF NOT WON% THEN PRINT "SORRY, YOU LOST . . . TRY AGAIN. "
4070 PRINT: PRINT "Do you want to play again? "; : GET A$: PRINT A$
4080 IF A$="Y" OR A$="Y" GOTO 20
4090 PRINT: PRINT "THE GAME IS OVER"
```

Figure 9.1
Listing of the Hangman Program

Generally, REM statements describe the purpose of program lines. When the computer executes a REM statement, it then ignores the remainder of the program line.

Lines 20 through 50, shown in Figure 9.2, instruct the computer to execute the subroutines that begin on the indicated lines. The remainder of this chapter describes the subroutines in the hangman program.

```
20 GOSUB 1000: REM TITLE, GAME DESCRIPTION, AND WORDS
30 GOSUB 2000: REM GRAPHICS
40 GOSUB 3000: REM EVALUATING GUESSES
50 GOSUB 4000: REM RESPONDING TO PLAYER
```

Figure 9.2
Lines 20 through 50

The first subroutine begins on line 1000 and ends on line 1330. Lines 1000 through 1060, shown in Figure 9.3, display the title.

```
1000 REM SCREEN #1
1010 REM **SCREEN TITLE**
1020 TEXT: NORMAL
1030 HOME
1040 TITLE$ = "HANGMAN A GAME"
1050 HTAB 15 - LEN(TITLE$) / 2
1060 VTAB 2: INVERSE: PRINT TITLE$: NORMAL
```

Figure 9.3
Lines 1000 through 1060

Line 1020 instructs the computer to change the screen to a text screen (TEXT statement) and change any special text screen effects to normal (NORMAL statement). Line 1030 brings the cursor to the beginning of the screen and clears all text from the screen—but does not clear memory as do the CLEAR and NEW statements.

Lines 1040 through 1060 use the centering formula to center the screen title on line 2 and print it in inverse. Line 1040 assigns variable name

TITLE$ to the string HANGMAN A GAME. Line 1050 contains the formula that finds the position at which the screen title must begin on the screen in order to be centered. The pieces of the formula are as follows:

- HTAB 15 represents the midpoint of the 31-character screen line
- TITLE$ is the variable set in line 1040 to represent the value of HANGMAN A GAME
- LEN(TITLE$) is the length of the value of TITLE$ and equals 14
- LEN(TITLE$) / 2 is one-half the length of the title and equals 7

Therefore, HTAB 15 - LEN(TITLE$) / 2 is 8. When the computer executes lines 1050 and 1060, it applies the value of TITLE$ to the formula and determines that the title must begin at position 8 of a screen line in order to be centered.

Lines 1070 through 1160, as shown in Figure 9.4, instruct the computer to display the remainder of the instructions, the hint, and the PRESS ANY KEY TO CONTINUE prompt.

```
1070 REM GREETINGS
1080 PRINT: PRINT TAB(7); "WELCOME TO HANGMAN!"
1090 REM INSTRUCTIONS
1100 PRINT: PRINT "HOW TO PLAY HANGMAN:"
1110 PRINT: PRINT " THE OBJECT OF THE GAME IS TO SAVE THE VICTIM BY
     GUESSING THE WORD IN 8 GUESSES OR LESS. "
1120 LINE$ = "HINT: START WITH VOWELS"
1130 HTAB 15 - LEN(LINE$) / 2
1140 VTAB 18: PRINT LINE$
1150 VTAB 20: PRINT "PRESS ANY KEY TO CONTINUE";
1160 GET A$
```

Figure 9.4
Lines 1070 through 1160

Lines 1120 through 1140 again use the centering formula to center the HINT: START WITH VOWELS line. Lines 1150 and 1160 display the PRESS

ANY KEY TO CONTINUE prompt and use the GET statement to get one character. Line 1160 allows the user to have plenty of time to read the screen before continuing with the game.

Lines 1170 through 1280, as shown in Figure 9.5, instruct the computer to select a word randomly from the data provided in the DATA statements.

```
1170 REM DATA STATEMENTS WITH WORDS
1180 RESTORE
1190 FOR W = 1 TO RND(1)*34+1 : READ WRD$
1200 NEXT
1210 DATA "BOOK", "TIGER", TEDDYBEAR", THOROUGH"
1220 DATA "DILEMMA", "ACTUALLY", "ELEPHANT", "ELEVATOR"
1230 DATA "HOUSEHOLD", "WISCONSIN", "AVIATOR", "ALLIGATOR"
1240 DATA "ABILITY", "SOCKS", "COMPUTER", "BARBARIAN"
1250 DATA "LIBRARY", "PERSONNEL", "POPCORN", "RADIATOR"
1260 DATA "ELEMENTARY", "INTERMEDIATE", "ADVANCED", "NOVICE"
1270 DATA "TELEVISION", "THEATRE", "MOVIE", "BALLET", "OPERA"
1280 DATA "ANALYST", "PROGRAMMER", "SYSTEM", "PRINTER", "MODEM"
```

Figure 9.5
Lines 1170 through 1280

Line 1180 uses the RESTORE statement to reset the DATA statements so that the READ statement reads from the beginning of the DATA statements. Line 1190 is the random-word generator, which uses the RND function and the number of words in the DATA statements to select a word randomly. In this example there are 34 words, so the value is 34.

NOTE: If you want to add words to the DATA statements, make sure you change 34 to the number of words that you have.

The program line instructs the computer to find a random number from 1 to 34 and then add 1. The READ statement in line 1190 READs a word, and the NEXT statement in line 1200 executes the loop again.

Lines 1300 through 1330 appear in Figure 9.6.

```
1300 WL = LEN(WRD$)
1310 FOR I = 1 TO WL
1320 GUESS$(I) = "-": NEXT
1330 GUESSED$ = " ": RETURN
```

Figure 9.6
Lines 1300 through 1330

Line 12 allocates space for words of up to 20 characters. Line 1300 assigns the length of the word to the variable WL. Line 1310 starts the guess loop for the length of the word. Line 1320 indicates that a hyphen (-) is a letter that has not been correctly guessed yet. If a hyphen (-) is found, the NEXT statement instructs the computer to continue the loop. Line 1330 tells the computer what to do when a null string rather than a hyphen is found: the loop ends; the subroutine is complete; and control is RETURNed to line 20. In line 20 the computer executes the REM statement and moves to the subroutine in line 30.

Lines 2000 through 2080, as shown in Figure 9.7, provide instructions for the border, scaffold, and brace.

The GR statement in line 2010 changes the text screen to the low-resolution screen and changes the color to yellow (COLOR=13). Line 2030 draws the border in yellow. Line 2050 draws the scaffold in yellow. Line 2070 draws the brace in yellow.

```
2000 REM SCREEN #2 ** GRAPHICS **
2010 GR: COLOR=13: REM CHANGE COLOR
2020 REM BORDER AROUND SCREEN
2030 HLIN 1, 39 AT 0: HLIN 1, 39 AT 38: VLIN 0, 39 AT 39: VLIN 0, 39 AT 0
2040 REM SCAFFOLD
2050 HLIN 7, 35 AT 29: VLIN 8, 36 AT 6: HLIN 7, 22 AT 8
2060 REM BRACE
2070 PLOT 7, 12: PLOT 8, 11: PLOT 9, 10: PLOT 10, 9: PLOT 21, 9: PLOT
     21, 10
2080 RETURN: REM BODY PARTS
```

Figure 9.7
Lines 2000 through 2080: Border, Scaffold, and Brace

Line 2080 returns control to line 30, where the computer executes the REM statement and then executes the subroutine in line 40. The remaining lines in the 2000 subroutine are not executed until the guesses are evaluated in the 3000 subroutine and the ON . . . GOSUB routine in line 3170 is executed.

Lines 2090 through 2160, as shown in Figure 9.8, provide instructions for the body and limbs.

```
2090 COLOR=3: HLIN 20, 23 AT 11: HLIN 20, 23 AT 17: VLIN 13, 16 AT 18:
     VLIN 13, 16 AT 24:
2092 PLOT 19, 12: PLOT 23, 12: PLOT 19, 16: PLOT 23, 16: RETURN
2100 COLOR=2: PLOT 20, 14: PLOT 20, 13: PLOT 22, 13: PLOT 22, 14:
     RETURN
2110 COLOR=12: PLOT 20, 16: PLOT 21, 15: PLOT 22, 16: RETURN
2120 COLOR=7: VLIN 18, 24 AT 21: RETURN
2130 COLOR=6: HLIN 18, 21 AT 20: VLIN 20, 22 AT 18: RETURN
2140 COLOR=14: HLIN 22, 25 AT 20: VLIN 20, 23 AT 24: RETURN
2150 COLOR=6: PLOT 20, 24: PLOT 19, 25: HLIN 17, 19 AT 26: RETURN
2160 COLOR=14: PLOT 22, 24: PLOT 23, 25: HLIN 24, 26 AT 26: RETURN
```

Figure 9.8
Lines 2090 through 2160: Body and Limbs

Lines 2090 and 2092 draw the head in dark red. Line 2100 draws the eyes in dark blue. Line 2110 draws the mouth in green. Line 2120 draws the body in light blue. Line 2130 draws the left arm in medium green, and line 2140 draws the right arm in light blue. Line 2150 draws the left leg in medium green, and line 2160 draws the right leg in light blue.

Lines 3000 through 3190 evaluate the guesses. Lines 3000 through 3050, as shown in Figure 9.9, instruct the computer to print, in the text part of the low-resolution screen, the word ANSWER: and either the letters that were correctly guessed or a hyphen.

Lines 3060 through 3190, as shown in Figure 9.10, tell the computer how to evaluate the answers and what to display on the screen. If no hyphens remain, then the player has guessed the word and won, in which case the computer is instructed to go to line 3190.

```
3000 REM EVALUATING GUESSES
3010 HOME: VTAB 21: PRINT "ANSWER: ";
3020 WON%=1: FOR I=1 TO WL
3030 PRINT GUESS$(I); :IF GUESS$(I)="-" THEN WON%=0
3040 NEXT: PRINT
3050 IF WON%=1 GOTO 3190
```

Figure 9.9
Lines 3000 through 3050

```
3060 PRINT "ATTEMPTS: "; GUESS$
3070 PRINT: PRINT "TAKE A GUESS: ";
3080 GET ANSWER$
3090 REM ANSWER OUT OF RANGE
3100 IF ANSWER$ < "A" OR ANSWER$ > "Z" THEN 3000
3110 REM GUESSED ALREADY
3120 RC%=0: FOR I = 1 TO WL
3130 IF MID$(GU$, I, 1)=AN$ THEN HTAB 1: PRINT "YOU GUESSED THAT
     ALREADY . . . ": I=WL: RC%=1: FOR PAUSE=0 TO 2000: NEXT
3150 IF MID$(WRD$, I, 1) = ANSWER$ THEN GUESS$(I) = ANSWER$: RC% = :
3160 NEXT: IF RC%=1 THEN 3010
3170 GUESS$ = GUESS$ + ANSWER$: ON LEN(GUESS$) GOSUB 2090, 2100,
     2110, 2120, 2130, 2140, 2150, 2160
3180 IF LEN(GU$) < 8 THEN 3010
3190 RETURN
```

Figure 9.10
Lines 3060 through 3190

Line 3060 prints incorrect guesses. Line 3070 asks the player to type a letter, and 3080 gets the response as soon as the player types it (no need to press the RETURN key). Lines 3090 through 3100 send the computer back to line 3000 to ask the player to take a guess if the player did not type a letter. Lines 3110 through 3140 take care of the situation in which the player already guessed a letter. If a letter has been guessed, the computer prints: YOU GUESSED THAT ALREADY.

The GOTO 3010 instruction in line 3160 then gives the player the opportunity to guess again—even if the last guess was the eighth. Line 3150 evaluates the next letter.

Lines 3130 and 3150 assign RC (replace count) with a value of 1. The value of RC can be either 0 or 1. Line 3150 compares the I position of the word (WRD$) with ANSWER$. If it is the same as the ANSWER$, then the GUESS$ is the ANSWER$. The replace count is assigned the value of 1. If the replace count is 1, as instructed by the second part of line 3160, the computer is sent to line 3000, where the player is given the opportunity to take another guess.

In line 3170, the value of GUESS$ is reassigned as the value of GUESS$ plus the value of ANSWER$. If the guess is incorrect, the ON . . . GOSUB routine in the second part of line 3170 sends the computer to lines 2090 through 2160, which draw the parts of the victim.

Line 3180 directs the computer to return to the beginning of the 3000 subroutine, which instructs the computer to print the correct or incorrect guess, if the player has made fewer than eight guesses, and continue through the loop with the next guess. If this is the eighth incorrect guess, then the computer ignores line 3180 and executes line 3190.

Line 3190 returns control to line 40. Here the computer executes the REM statement—a description of subroutine 3000 and an instruction to continue to the next statement—and then executes the subroutine in line 50.

The subroutine in lines 4000 through 4090, as shown in Figure 9.11, instructs the computer on what to do when eight guesses have been made.

Line 4010 randomly selects and displays bars of color across the screen to end the game. Line 4020 pauses the program for a few seconds so that you can appreciate the graphics. Instructions in line 4030 replace the low-resolution graphics screen with the text screen, clear the screen and move the cursor to the first position, and finally move the cursor to the beginning of the third line.

Lines 4050 and 4060 evaluate the value of WON%. If the value of WON% is 1 (as assigned in line 3020), then the condition in 4050 is true, and the computer displays YOU WON. Otherwise, the value of WON% is 0 or NOT WON%, as assigned in line 3030, and the computer displays YOU LOST.

```
4000 REM **END THE GAME**
4010 FOR I = 0 TO 39: COLOR= RND(1)*15+1: HLIN 0, 39 AT I: NEXT
4020 FOR PAUSE=0 TO 1500: NEXT
4030 TEXT: HOME: VTAB 3
4050 IF WON% THEN PRINT "YOU WON"
4060 IF NOT WON% THEN PRINT "SORRY, YOU LOST . . . TRY AGAIN. "
4070 PRINT: PRINT "Do you want to play again? "; : GET A$: PRINT A$
4080 IF A$="Y" OR A$="Y" GOTO 20
4090 PRINT: PRINT "THE GAME IS OVER"
```

Figure 9.11
Lines 4000 through 4090

Meal Planner

Do you ever wish someone else would decide what to have for dinner tonight and still include the dishes you like to eat? The meal planner may be the answer. It randomly selects one from each of the following categories:

- Appetizer
- Entree
- Vegetable
- Side dish (for example, rice, potato, stuffing)
- Beverage
- Dessert

You can change the meal selections to suit your tastes as you type the program. In addition, a surprise meal—go out to dinner—is written into the program and appears as a meal five percent of the time. You can change the frequency of the surprise meal by changing the constant in the random number generator.

You tell the computer how many meals you want it to select for the week. You also tell it whether or not you want it to print out a list of the meals that you can use as a shopping list. The meal planner screen appears, as shown in Figure 9.12.

The program listing appears in Figure 9.13.

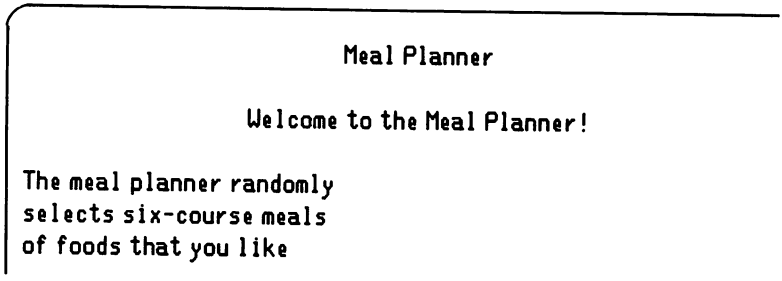


Figure 9.12

```
10   REM The Meal Planner copyright 1983
20   REM By Richard L. Roth and Pamela J. Roth
30   REM SCREEN TITLE
40   HOME
50   TITLE$ = " MEAL PLANNER"
60   HTAB 15 - LEN(TITLE$)/2
70   VTAB 2: INVERSE: PRINT TITLE$: NORMAL
80   WELCOME$ = "Welcome to the Meal Planner!"
90   HTAB 15 - LEN(WELCOME$)/2
100  VTAB 4: PRINT WELCOME$
110  VTAB 6: PRINT "The meal planner randomly"
120  PRINT "selects six-course meals of ": PRINT "foods that you
     like. "
130  P9 = 1 : REM PRINTER SLOT NUMBER
140  DIM M$(6): REM SAVE SPACE FOR 6 COURSES IN A MEAL
150  REM GENERATE MEALS AS REQUESTED
160  PRINT: INPUT "HOW MANY MEALS TO PLAN? "; I2
170  PRINT: PRINT "DO YOU WANT PRINTER OUTPUT? "; GET A$: PRINT A$
180  IF NOT (A$="N" OR A$="N" OR A$="Y" OR A$="Y") THEN 170
190  P = 0: IF (A$="Y" OR A$="Y"THEN P = P9
200  REM SELECT FOOD
210  FOR I = 1 TO I2
220  FOR J = 1 TO 6
230  S$ = MID$("AEVSDB", J, 1): REM SELECT COURSE
240  GOSUB 1000: REM FIND A COURSE
250  IF (R$ = "FREE") THEN M$(1) = "FREE": GOTO 300
260  M$(J) = R$
```

```
270  NEXT J
300  REM PRINT MEALS
310  PR #P
320  PRINT: PRINT "MEAL NUMBER "; I
330  IF (M$(1) <> "FREE") THEN GOTO 400
340  PRINT "PLEASANT SURPRISE: " : PRINT TAB(14); "GO OUT TO
     DINNER"
350  GOTO 460
400  PRINT "APPETIZER:"; TAB(12); M$(1)
410  PRINT "ENTREE:"; TAB(12); M$(2)
420  PRINT "VEGETABLE:"; TAB(12); M$(3)
430  PRINT "SIDE DISH:"; TAB(12); M$(4)
440  PRINT "DESSERT:"; TAB(12); M$(5)
450  PRINT "BEVERAGE:"; TAB(12); M$(6)
460  PR #0
470  NEXT I
500  END
1000 REM RANDOMLY PICK A MEAL COURSE
1005 REM S$ = ONE CHARACTER COURSE SELECTOR ON ENTRY
1007 REM R$ = NAME OF SELECTED COURSE ON EXIT
1010 REM FIRST: COUNT NUMBER OF SELECTIONS
1020 N = 0
1030 RESTORE
1040 READ T$ : REM READ ALL ENTRIES UNTIL END
1050 IF T$ = "DONE" THEN 1080
1060 IF LEFT$(T$, 1) = S$ THEN N = N + 1: REM COUNT MATCH
1070 GOTO 1040
1080 REM RANDOMLY PICK ONE
1090 IF (N = 0) THEN R$ = "NONE": GOTO 1180
1100 N1 = RND(1) * N * 1.05: REM PICK WITH 5% FREE
1110 IF (N1 > N) THEN R$ = "FREE": GOTO 1180
1120 RESTORE
1130 FOR N = 1 TO N1
1140 READ T$ : REM LOOP PAST IMPROPER ENTRIES
1150 IF LEFT$(T$, 1) <> S$ THEN 1140
1160 NEXT N
1170 R$ = MID$(T$, 2, 255): REM PICK OFF SELECTOR
1180 RETURN
2000 DATA "APEA SOUP", "ACLAMS ON THE HALF SHELL", "ACHOPPED
     LIVER", "AMINNESTRONE", "ASHRIMP COCKTAIL"
```

```
2010 DATA "ECHICKEN", "ESTEAK", "EHAM HAWAIIAN", "EHADDOCK
     FILET", "ELOBSTER"
2020 DATA "VLIMA BEANS", "VBROCCOLI", "VCARROTS", "VPEAS",
     "VSQUASH", "VCORN"
2030 DATA "SFRENCH FRIES", "SRICE PILAF", "SPEDDLER FRIES",
     "SSTUFFING"
2040 DATA "DICE CREAM", "DAPPLE PIE", "DCUP CAKES", "DSTEWED
     PRUNES", "DPARFAIT"
2050 DATA "BTEA", "BCOFFEE", "BCHOCOLATE MILK", "BWINE"
2060 DATA "DONE"
```

Figure 9.13
Listing of the Meal Planner Program

Lines 10 and 20 introduce the program and its authors. Line 30 introduces the routine used to display the title in the middle of a screen line. Line 40 clears the screen and moves the cursor to the beginning of the screen. Lines 50, 60, and 70 should be familiar to you: they include the formula used throughout this book to center text.

Lines 80, 90, and 100 center the line "Welcome to the Meal Planner!" and print it on line 4. Lines 110 and 120 print this description of the program on screen lines 6, 7, and 8:

```
The meal planner randomly
selects six-course meals
of foods that you like
```

Line 130 assigns the variable used to change the current output device from the screen to the printer. The change is made when the user responds Y to the "DO YOU WANT PRINTER OUTPUT?" prompt in program line 170.

Line 140 sets up or "dimensions" array M$ with enough room for six courses in each meal (0-6). Line 150 announces what is to come next: the request to generate meals. Line 160 asks the user how many meals are to be generated and instructs the computer to look for a response, which is assigned to variable I2. Line 170 asks the user whether or not the meals are to be printed and instructs the computer to look for a response, assigned to variable A$.

Line 180 instructs the computer to evaluate the user's response to the printer output prompt. If the user's response is neither Y nor N, the computer is instructed to ask the question again (GOTO 170). If the answer is Y or N, the computer skips the second part of 180 and executes line 190. Line 190 tells the computer that if the user answered Y (IF A$ = Y), change the current output device to the printer. If the user answered N (does not want a printout), the computer leaves the screen as the current output device and moves on to line 200.

Lines 200 through 270, as shown in Figure 9.14, instruct the computer to select the courses for the desired number of meals.

```
200 REM SELECT FOOD
210 FOR I = 1 TO I2
220 FOR J = 1 TO 6
230 S$ = MID$("AEVSDB", J, 1): REM SELECT COURSE
240 GOSUB 1000: REM FIND A COURSE
250 IF (R$ = "FREE") THEN M$(1) = "FREE": GOTO 300
260 M$(J) = R$
270 NEXT J
```

Figure 9.14
Lines 200 through 270

Line 200 introduces the next section of the program: loops used to select the courses. Line 210 starts the first loop, which uses the response to the HOW MANY MEALS TO PLAN? prompt: I2. Line 220 starts the second loop, which will cycle six times, once for each course.

In other words, if the user wants two meals selected, the computer will go through the meal selection loop twice, selecting six courses each time through.

Line 230 assigns S$ as the course, and each time through the loop the course changes. The string "AEVSDB" consists of the first letter of each course: appetizer, entree, vegetable, side dish, dessert, and beverage. Each time through the loop, J increases by 1, causing the computer to select the next letter in the string. When the computer has selected a letter that represents a course, it executes line 240, which starts the subroutine beginning on line 1000.

The subroutine that begins in program line 1000 uses S$ to select a course placed in R$, using T$ as a temporary. The value of R$ is then saved in line 250 as the correct course.

In lines 1040 through 1070, shown in Figure 9.15, the computer is instructed to READ and evaluate DATA statements to find the options for the course that matches the current value of S$.

```
1040 READ T$ : REM READ ALL ENTRIES UNTIL END
1050 IF T$ = "DONE" THEN 1080
1060 IF LEFT$(T$, 1) = S$ THEN N = N + 1 : REM COUNT MATCH
1070 GOTO 1040
```

Figure 9.15
Lines 1040 through 1070

In line 1040 the computer is instructed to READ DATA into T$. (Notice that in lines 2000 through 2050 there is one DATA statement for each course. Also notice that the choices within each course begin with the first letter of the course.) Line 1050 instructs the computer to go to another set of instructions (starting on line 1080) if all the DATA statements have been read.

The value of T$ is evaluated in line 1060. If the first letter of T$ is equal to S$, the first letter of a course, then the course is counted. If the first letter is not equal to S$, the computer READs the next DATA statement and so on until it counts all matching courses.

Lines 1080 through 1110, as shown in Figure 9.16, provide the instructions for the computer to pick one of the choices for one course.

Line 1090 catches entries whose first letters do not match the first letter of the course currently being selected. If you add options to this program, make sure you add the first letter of the course to the option as follows:

- A for appetizer
- E for entree
- V for vegetable
- S for side dish
- D for dessert
- B for beverage

```
1080 REM RANDOMLY PICK ONE
1090 IF (N = 0) THEN R$ = "NONE": GOTO 1180
1100 N1 = RND(1) * N * 1.05: REM PICK WITH 5% FREE
1110 IF (N1 > N) THEN R$ = "FREE": GOTO 1180
```

Figure 9.16
Lines 1080 through 1110

Line 1100 gives instructions to generate a random value. The five-percent free area included in the random generator means that five percent of the time the computer will not choose one of the available selections. Line 1110 tells the computer how to evaluate the random value. If the value of N1 is greater than the value of N, which occurs five percent of the time, variable R$ equals FREE. Otherwise, a regular course is selected as the value of R$.

Lines 1030 and 1120 RESTORE the DATA statements so that the computer will READ through them from the top during the next loop.

Lines 1130 through 1180 are shown in Figure 9.17.

```
1130 FOR N = 1 TO N1
1140 READ T$ : REM LOOP PAST IMPROPER ENTRIES
1150 IF LEFT$(T$, 1) <> S$ THEN 1140
1160 NEXT N
1170 R$ = MID$(T$, 2, 255): REM PICK OFF SELECTOR
1180 RETURN
```

Figure 9.17
Lines 1130 through 1180

Lines 1130 through 1160 instruct the computer to READ through the DATA fo find the course requested (N1). Then R$ is evaluated in line 1170 as everything to the right of the first letter, or everything except the value of S$.

Line 1180 instructs the computer to RETURN to line 250, where it evaluates the value of R$. If R$ is FREE, then the computer is instructed to

GOTO line 300, and it ultimately displays or prints this selection: PLEASANT SURPRISE: GO OUT TO DINNER. If R$ is a course, the course name is stored and the next course selected. The procedure continues until all six courses are chosen for the meal.

When the course selection loop has been completed six times, a meal has been planned. After the computer has selected the number of meals the user wants, the computer executes the print instructions that begin on line 400 (unless already executed for the FREE meal). Lines 400 through 470, shown in Figure 9.18, instruct the computer to print the selection for each course of each meal.

```
400 PRINT "APPETIZER:"; TAB(12); M$(1)
410 PRINT "ENTREE:"; TAB(12); M$(2)
420 PRINT "VEGETABLE:"; TAB(12); M$(3)
430 PRINT "SIDE DISH:"; TAB(12); M$(4)
440 PRINT "DESSERT:"; TAB(12); M$(5)
450 PRINT "BEVERAGE:"; TAB(12); M$(6)
460 PR #0
470 NEXT I
```

Figure 9.18
Lines 400 through 470

Line 500 signals the end of the program.

10

Packaged Software and Other Materials

Coleco is preparing and translating software for business, home, education, and entertainment uses. At present, these programs are being translated onto digital data packs. When flexible disk drives are available for ADAM, the programs will be available on flexible diskettes.

Software Currently Available

The following software is currently available for ADAM and can be used in either the memory console or the memory module. The prices shown are suggested retail and actual advertised prices for 1983. As more retailers stock ADAM, the prices will most likely drop.

Dr. Seuss Packages

Coleco has purchased the rights to use Dr. Seuss characters in educational software.

Dr. Seuss Reading

The Dr. Seuss Reading package, aimed at 3- to 5-year-olds, contains full-color reading exercises presented by the popular Dr. Seuss characters.

Retail price: from $29.70 to $34.95.

Dr. Seuss Numbers Fun

In the Dr. Seuss Numbers package, aimed at 3- to 5-year-olds, exercises for learning numbers are presented by Dr. Seuss characters.

Retail price: from $29.70 to $34.95.

Dr. Seuss Storymaker

The Dr. Seuss Storymaker package allows children to create stories with Dr. Seuss characters.

Retail price: from $29.70 to $34.95.

Smurf Packages

Coleco has purchased the rights to use Smurf characters in educational software.

Smurf Fun with Numbers

The Smurf Fun with Numbers package, aimed at 5- to 7-year-olds, contains arithmetic exercises presented by Smurf characters.

Retail price: from $29.70 to $34.95.

Smurf Reading Adventures

The Smurf Reading Adventures package, aimed at 5-to 7-year-olds, uses Smurf characters to present reading exercises.

Retail price: from $29.70 to 34.95.

Other Available Software

In addition to educational packages, the following software should have been available by the end of 1983:

Typing Tutor

The Typing Tutor is aimed at anyone who wants to learn or improve typing skills.

Retail price: $31.95.

SmartLOGO

Coleco has contracted with Seymour Papert, the developer of Logo and Professor of Mathematics at MIT, to develop Logo for the ADAM. Logo is a programming language designed to help children have fun while learning the fundamentals of programming.

With SmartLOGO, youngsters learn programming techniques through the process of completing such tasks as moving a "turtle" around the screen or drawing a leaf. Children learn to use and save a set of instructions, and, above all, to think logically.

Retail price: $79.95.

Software Scheduled for 1984 Delivery

In addition to the software currently available, Coleco intends to provide a data base management system, SmartFILER, that records, sorts, displays, saves, and prints information. Another educational package, Homework Helper, will allow students to quiz themselves on various topics. Coleco has also purchased the rights to the ColorForms artwork and will create a graphics package for children.

Hardware

The following hardware is available now for the ADAM:

Second Digital Data Drive

A second digital data drive can be used to store and retrieve programs and SmartWRITER documents. SmartWRITER is already set to store documents in drive A, B, or C. The first drive is A, and the second drive is B. Drive C is reserved for a diskette drive that will be available sometime in 1984.

Retail price: $149.95.

Digital Data Packs

Extra digital data packs for storing programs and documents are currently available from your Coleco retailer. The extra packs hold the same amount of data—the equivalent of 125 typewritten pages—as the blank pack that comes with the ADAM.

Retail price: from $10.00 to $12.95 each.

Ribbons and Daisy Wheels

The SmartWRITER printer uses standard Diablo ribbons and daisy wheels, which are available from computer stores, computer mail-order houses, office-supply stores, and office-supply mail-order houses. Daisy wheels are available in several different typefaces. The ADAM comes with a 10-point daisy wheel.

Hardware Scheduled for 1984 Delivery

Coleco plans to expand the ADAM and will be offering the following products:

- 5 1/4-inch diskette drive that will allow you to run CP/M-compatible software
- Smart Telephone Modem
- Electronic Sketchpad

Games Currently Available

Coleco now has two categories of games that you can play on the ADAM.

- ColecoVision Game Cartridges—can be played on both ColecoVision and ADAM
- ADAM Super Game Cartridges—can be played only on the stand-alone ADAM console or the ADAM module connected to a ColecoVision Game Machine

Super Game Cartridges, which take advantage of ADAM's extra graphics capability, will not run on ColecoVision without the ADAM module attached.

ColecoVision Game Cartridges

The following ColecoVision Game Cartridges are available at retail prices from $29.70 to $49.95.

- Mr. Do
- Time Pilot
- Sub Roc
- Front Line

- Looping
- Pepper II
- Venture
- Carnival
- Mouse Trap
- Space Fury
- Lady Bug
- Blackjack and Poker
- Cosmic Avenger
- Space Panic
- Gorf
- Smurf Rescue
- Donkey Kong, Jr.
- Q*Bert
- Popeye
- Zaxxon
- Miner 2049er
- Victory

Super Game Cartridges

The following Super Game Cartridges are available at retail prices from $29.70 to $39.95:

- Zaxxon
- Donkey Kong
- Donkey Kong, Jr.
- Smurf Rescue
- Sub Roc
- Turbo
- Time Pilot
- Tunnels and Trolls
- Sword and Sorcerer
- Ulysses and the Golden Fleece
- Cranston Manor
- Trolls Tale

Other Materials

Coleco has been working on several ventures to make certain that ADAM is a success, with the following results:

- In February, 1984, Scholastic, Inc. will begin distribution of a magazine devoted exclusively to the ADAM.
- In conjunction with AT&T, Coleco will be offering access to interactive games, that is, games that respond to the user's responses.
- An ADAM User's Club will be sponsored by Coleco. The details will be provided by Coleco in early 1984. Most likely, other user groups, such as the Boston Computer Society, will sponsor unaffiliated groups as well. Check your local computer society for details.

Appendix A
Error Messages

Error messages can appear while you are programming or while a program is running. The most likely occurrence of an error message is while you are programming. Each time you press the RETURN key, Adam's built-in editor reads the program line and evaluates it. If the syntax, that is, the structure of the line, does not match the SmartBASIC program language rules, an error message will appear on the screen.

In many cases, an arrow will point to the location of the incorrect structure. Sometimes the message will tell you exactly what the problem is, such as NUMERIC VALUE EXPECTED. At other times the message is vague. Either way, you must review the program line and determine what you have left out or incorrectly stated.

When evaluating error messages, look for clues in the message. If you get a message such as NUMERIC VALUE EXPECTED, but you did not intend to construct a statement that has a numeric value (you wanted to indicate a string variable instead), make certain that you put quotation marks (") around the string. Chances are, rather than a major reworking of the structure, the program line requires the addition of something you overlooked, such as

- One or more spaces
- Parentheses (especially around the value of a function)
- Numeric value
- Colon, semicolon, or comma
- Quotation marks

To correct a program line, you do not need to retype the entire line. You can move the cursor to the location in the line where you will begin to make changes, overstrike the line as necessary, move the cursor to the end of the program line (as opposed to the screen line), and press the RETURN key. As soon as you press the RETURN key, SmartBASIC evaluates the new version of the program line. If it accepts the new version, SmartBASIC will display a new line symbol (]) with the cursor (_) to the right of it, at which time you can continue programming or run the program.

If you get another error message, continue your review of the line. To isolate the problem in a line with several statements, place each statement on a separate program line and test each individually.

SmartBASIC does not find some errors until the program is running. Suppose, for example, you instruct the computer to READ DATA statements until it finds a string that matches a given condition. In the process, however, you fail to include a DATA statement with the matching data. In such a case, an error message will not appear until the program is running and the READ statement is actually searching for data. You will also get an error message if, for example, you include a GOTO statement with line number 1000, but then forget to write program line 1000.

To correct an error that appears while a program is running, use the following steps:

1. LIST the line indicated in the error message. Be careful not to type the line number by itself; otherwise, you will delete it.
2. Locate the error as you would if you got a message after pressing the RETURN key.
3. Overstrike the line with the corrections, move the cursor to the end of the program line, and press the RETURN key.
4. Type RUN and press the RETURN key. If you solve the problem, the computer will execute that line and continue to execute other lines in the program—unless the computer encounters another problem.
5. Repeat these steps until the program runs as you intended.

Most errors are easily corrected—once you determine the cause. At first, determining the cause and finding the cure is time-consuming, but after a while you begin to see patterns and can find and correct problems quickly.

Appendix B
Summary of Statements, Commands, and Functions

SmartBASIC Statements, Commands, and Functions at a Glance

ABS
ASC
Assignment Statement
ATN
CALL
CATALOG
CHR$
CLEAR
COLOR=
CONT
COS
DATA
DEF FN
DEL
DIM
END
EXP
FN
FOR
FRE
GET
GOSUB
GOTO
GR
HCOLOR
HGR
HGR2
HLIN
HOME
HPLOT
HTAB
IF...THEN
IN#
INPUT
INT
INVERSE
LEFT$
LEN
LET
LIST
LOAD
LOCK

LOG
MID$
NEW
NEXT
NORMAL
ON...GOSUB
ON...GOTO
PDL
PLOT
POP
POS
PRINT
READ
RECALL
REM
RESTORE
RESUME
RETURN
RIGHT$
RND
RUN
SAVE
SCRN
SGN
SIN
SPC
SPEED=
SQR
STOP
STORE
STR$
TAB
TAN
TEXT
UNLOCK
VAL
VLIN
VTAB

SmartBASIC Statements, Commands, and Functions

This section briefly describes each SmartBASIC statement, command, and function. Unless otherwise noted, you can find details in Chapter 5, "The SmartBASIC Language." An asterisk (*) indicates that this is the only description of the element in this book.

Statements can operate alone or on string expressions, arithmetic expressions, string variables, arithmetic variables, line numbers, integers, real numbers, and subscripts, as noted. Commands operate alone and on line numbers, as noted. Functions operate on string expressions or arithmetic expressions, as noted in each description.

Punctuation, including parentheses, is included when it is part of the syntax. Parentheses are part of the syntax of functions, not statements or commands. Brackets and braces are used to

[ ] indicate elements that you can omit
{ } indicate elements that you can repeat

The following abbreviations are used:

- ae means arithmetic expression
- se means string expression
- av means arithmetic variable
- sv means string variable

ABS (ae)

Function that provides the absolute value of the arithmetic expression without a sign

ASC (ae)

Function that provides the American Standard Code for Information Interchange (ASCII) for the first character in the arithmetic expression

Assignment Statement
NAME$ = "string"

Statement that assigns the value of the expression after the equals (=) sign to the variable before it. See also LET statement

ATN (ae)

Function that provides in radians the arc tangent of the arithmetic expression

CATALOG

Command that lists the contents of the digital data pack (see Chapter 4)

CHR$ (ae)

Provides the character for the ASCII code that is entered as the arithmetic expression

CLEAR

* Command that restores variables and internal control information to their original state

COLOR= ae

Statement sets the color (0-15) of the display for plotting low-resolution graphics. There is no space between COLOR and = (see Chapter 8).

CONT

* Command that resumes program execution after the program has been stopped by the STOP statement, the END statement, the

CONTROL-C keys, or the CONTROL-RESET keys. Use only for immediate execution.

COS (ae)
Function that provides the cosine of the arithmetic expression and must be expressed as a radian

DATA [string, literal, real, integer]
Indicates a list of items in the program to be read by the READ statement(s)

DEF FN name (name) = ae
Function that defines a new function for use in a program

DEL line number, line number
Command that removes one or more consecutive lines from a program (see Chapter 4)

DIM name [% or $] subscript [{, name [% or $] subscript }]
Statement that defines and allocates spaces for one or more arrays

END
Statement that indicates that the program has ended (see Chapter 4)

EXP (ae)
Function that raises a constant to the power of the arithmetic expression

FN name (ae)
Function that has been defined using the DEF FN function

FOR name = ae TO ae [STEP ae]
Statement that indicates the beginning of a FOR . . . NEXT loop. The TO statement is required. The STEP statement, which is optional, is used to indicate stepped values rather than consecutive values, for example, incrementing by 2.

GET variable
Statement that accepts a character from the keyboard as input without displaying it on the screen and without requiring the user to press the RETURN key

GOSUB line number

Statement that sends the computer to the beginning of a subroutine. The RETURN key statement at the end of a subroutine sends the computer back to the line after the GOSUB statement.

GR

Statement that changes the text screen of 31 columns by 24 rows to the low-resolution graphics screen of 40 columns by 40 rows with 4 rows for text at the bottom of the screen (see Chapter 8)

HCOLOR= ae

Statement that sets the color (0-15) for high-resolution graphics (see Chapter 8)

HGR

Statement that changes the text screen or low-resolution graphics screen to page 1 of the high-resolution graphics screen of 280 x 160 with 4 rows for text at the bottom of the screen (see Chapter 8)

HGR2

Statement that changes the text screen or low-resolution graphics screen to page 2 of the high-resolution graphics screen of 280 x 192 (see Chapter 8)

HLIN ae1, ae2 AT ae3

Statement that indicates the starting column (ae1), ending column (ae2), and row (ae3) where a horizontal line is drawn on a low-resolution graphics screen (see Chapter 8)

HOME

Statement that clears text from the screen and moves the cursor to the first position of the screen (0,0)

HPLOT ae, ae [{TO ae, ae}]

Statement that assigns the high-resolution current color (see the HCOLOR= statement) to a specified block or set of blocks. The first number indicates the column, and the second number indicates the row. Examples of HPLOT statements are shown in Table 8.9 (see Chapter 8).

HTAB ae

Statement that moves the cursor to a specific screen column; often used before a PRINT statement to indicate where a prompt should appear on the screen or where a response should be printed on a page

IF ae THEN statement, [{: statement}]
IF ae THEN [GOTO] line number
IF ae [THEN] GOTO line number

Statements that test for the given conditions. If the given condition is true or present, the THEN statement is carried out. If the condition is false or absent, the next program line is carried out.

IN# ae

Statement that indicates the source of subsequent input; usually used to change input from keyboard to memory console (digital data pack)

INPUT [se;] variable [{, variable}]

Statement that accepts a line of input from the current input device; also used to read a value into a variable place after the semicolon

INT (ae)

Function that provides the integer value of the arithmetic expression

INVERSE

Statement that reverses appearance of text on the screen (white-on-black to black-on-white) from the subsequent PRINT statement to the next NORMAL statement

LEFT$ (se, ae)

Function that provides the given number of characters from the beginning of the given string

LEN (se)

Function that provides the length of characters in the string

[LET] av = ae
[LET] sv = se

Same as assignment statement

LIST [line number] [-line number]
LIST [line number] [,line number]

Command that displays the given line numbers on the screen if the screen is the current output device; command that prints the given line numbers if the printer is the current output device; command that writes the given line numbers to a digital data pack if the memory console is the current output device

LOAD [name]
Command that reads a program from the specified file into internal memory from a digital data pack (see chapter 4)

LOCK
* Command that keeps files from being deleted. A LOCKed file cannot be deleted until it is UNLOCKed.

LOG (ae)
Function that provides the logarithm of the arithmetic expression

MID$ (se, ae [,ae])
Function that provides the given number of characters from the given position. If the number of characters is not given, the function provides all characters from the given position to the end of the string.

NEW
Command that clears the current program from internal memory and resets all variables and internal controls to their original states; not used within a program (see Chapter 4)

NEXT [ae[{, ae]}]
Statement that ends a FOR . . . NEXT loop and sends the computer to the next value of the variable in the FOR statement

NORMAL
Statement that returns previous INVERSE statement to original state of screen.

ON ae GOSUB line number [{, line number }]

Statement that indicates the line numbers to be executed when a given value occurs. If none of the given values occurs, the next program line is executed.

ON ae GOTO [{, line number}]
Pair of statements that indicates a subroutine to execute according to the value of the arithmetic expression. The RETURN statement at the end of a subroutine sends the computer to the line after the ON . . . GOSUB statement.

PLOT ae,ae
Statement that assigns the low-resolution current color (see the COLOR= statement) to a specified block. The first number indicates the

column, and the second number indicates the row. Examples of PLOT statements are shown in Table 8.4 (see Chapter 8).

POP

Statement placed after a subroutine and before a RETURN statement to instruct the computer to return to the statement that follows the starting point of the first subroutine statement, such as GOSUB and ON . . . GOSUB

POS (se or ae)

Function that provides the horizontal position on the text screen. The expression, though ignored, is required.

PR# ae

Statement that indicates or changes the current output device, such as the screen or printer

PRINT [{se or ae [,\;]}]

Statement that writes a line of output to the current output device, such as the screen or printer

READ var [{,var}]

Statement that reads value of DATA statements into indicated variables

REM {remark}

Statement that instructs the computer to ignore a descriptive passage after it

RESTORE

Statement that instructs the computer to read the first element of the first DATA statement into the variable indicated by the next READ statement

RESUME

* Statement placed at the end of an error-handling subroutine, RESUME instructs the computer to return to the beginning of the statement where the error occurred.

RETURN

Statement placed at the end of a subroutine. RETURN instructs the computer to return to the end of the GOSUB or ON . . . GOSUB statement.

RIGHT$ (se, ae)
Function that reads from the last character of the string on the left and provides the given number of characters in the string. The string expression is first (se), and the number of characters to be provided is second (ae).

RND (ae)
Function used to generate random numbers, strings, and graphics

RUN
RUN [line number]
RUN [name]
Command used to execute any SmartBASIC program in memory by typing RUN and pressing the RETURN key; also used to execute a program in memory starting from the given line number; also used to load and execute a program from a digital data pack (see Chapter 4)

SAVE name
Command that saves a SmartBASIC program to a digital data file (see Chapter 4)

SCRN (ae, ae)
Function that provides the code (0-15) for the color displayed at the indicated column and row on the low-resolution graphics screen

SGN (ae)
Function that provides -1, 0, 1, depending on the sign of the arithmetic expression

SIN (ae)
Function that provides the sine of the arithmetic expression

SPC (ae)
Function that moves the cursor or print head the indicated number of spaces in a line of text on the screen or on a printout; also used to leave the indicated number of spaces between words or characters. The SPC function is part of a PRINT statement.

SPEED= ae
Statement that allows you to change the speed at which text appears on the screen. The default and maximum speed is 255; the minimum is 0.

SQR (ae)

Function that calculates the positive square root of the arithmetic expression

STOP

* Statement used to stop a program after a particular program line. You can resume the program by typing the CONT command and pressing the RETURN key.

STR$ (ae)

Function that provides a string value for the arithmetic expression (numeric value) given

TAB (ae)

Function that moves the cursor or printer head the given number of spaces in from the beginning of the line. The TAB function is part of a PRINT statement.

TAN (ae)

Function that provides the tangent of the arithmetic expression and must be in radians rather than degrees

TEXT

Statement that changes a low-resolution or high-resolution graphics screen to a text screen

UNLOCK

* Command used to unlock a file that has been LOCKed to keep it from being deleted from a digital data pack

VAL (se)

Function that changes a string expression to a numeric value

VLIN ae1,ae2 AT ae3

Statement that draws a vertical line from row ae1 to row ae2 in column ae3. This statement can be used only on a low-resolution graphics screen (see Chapter 8).

VTAB (ae)

Statement that moves the cursor or printer head to the specified screen or page line

WRITE

* Command used to write print statements to a file

Appendix C
ASCII Code Equivalents

This appendix contains the American Standard Code for Information Interchange (ASCII) decimal and hexadecimal equivalents for characters that can be typed at the keyboard. The decimal equivalents are used in the CHR$(a) function, where "a" is the decimal equivalent of a character.

Char.	*Dec.*	*Hex.*
SPACE	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34

5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
[	91	5B
\	92	5C

]	93	5D
^	94	5E
_	95	5F
`	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
~	126	7E
DELETE	127	7F

Appendix D
SmartBASIC Reserved Words

Certain words or combinations of characters are reserved for use in SmartBASIC statements, functions, and commands and cannot be used as part of a variable name without causing an error message. For more information about variable names, see Chapter 5. The following list includes reserved words and character sequences that you should take care not to include in variable names:

ABS
AND
ASC
AT
ATN

CHR$
CLEAR
CLOSE
COLOR=
CONT
COS

DATA
DEF
DEL
DELETE
DIM

END
EXP

FN
FOR
FRE

GET
GOSUB
GOTO
GR

HCOLOR=
HGR
HGR2
HLIN
HOME
HPLOT
HTAB

IF
IN#
INPUT
INT
INVERSE

LEFT$
LEN
LET
LIST
LOAD
LOCK
LOG

MID$

NEW
NEXT
NORMAL
NOT

ON
OPEN
OR

PLOT
POP
POS
PR#
PRINT

READ
REM
RENAME
RESTORE
RETURN
RIGHT$
RND
RUN

SAVE
SCRN
SGN
SIN
SPC
SPEED=
SQR
STEP
STOP
STORE
STR$

TAB
TAN
TEXT
THEN
TO

UNLOCK

VAL
VLIN
VTAB

WAIT
WRITE

Glossary

The terms in this glossary are primarily computer terms. Many are new words that have evolved within the computer industry; others are common words or combinations of common words that have special meanings within the world of computers.

A

accumulator - Adds, divides, multiplies, and subtracts amounts that are entered into it

address (memory) - A particular location in memory; the act of finding a memory location

algorithm - A formula for solving mathematical problems

animation - The process by which graphics are used to write and erase points, blocks, horizontal, and vertical lines to give the appearance of motion

APL - An acronym for A Programming Language, which is the name of a programming language

Applesoft BASIC - A version of BASIC designed for Apple IIe computers. With slight modifications, programs written in Applesoft BASIC can be translated into SmartBASIC.

application - A program that has a particular application, such as programs written for the following accounting transactions: accounts receivable, accounts payable, inventory, and payroll

arithmetic expression - A numeric string

arithmetic operator - One of the five operators used to accumulate and manipulate numeric data: + (addition), - (subtraction), * (multiplication), / (division), and ^ (exponentiation)

array - A group of related data elements. The DIM name (x) statement is used to reserve space for x number of elements in array "name" (see DIM in Appendix B and Chapter 5).

argument - A value on which a statement, function, or command operates. For example, in the LOG (x) function, the computer will find the logarithm of the argument x.

B

backspace - The process by which the cursor is moved to the left or to the previous line

BACKSPACE key - The key used to move the cursor to the left, deleting any characters in its path

backup/backdown - The practice of making one or more copies (backup) of a document, and then using a copy (backdown) in case something happens to the original document or to the device on which it is stored. The term *backup* is used more than backdown.

BASIC - Beginner's All-purpose Symbolic Instruction Code, a programming language designed at Dartmouth College

binary - A counting system consisting of all the possible combinations of 0 and 1 available in the decimal and hexadecimal systems

bit - An abbreviation for binary digit

block - The point at which a row and column meet on a graphics screen (see Chapter 8)

branch - A particular path or route that a program can follow to process information; to create a path or route in a program; or to take a path or route in a program

bucket - Workspace where a number is kept before or after processing

buffer - An area of a computer that temporarily holds data until the computer is instructed to send the data elsewhere

bug - An error in the logic a programmer used and incorporated in a program; a problem in hardware; or a problem that occurs as a result of combining software and hardware

C

COBOL - COmmon Business Oriented Language, a programming language used primarily for business applications

communications - Software and hardware used to send data between computer components inside a computer and between different computers

compatible - A term used to describe the condition in which software written by and for one manufacturer can be used on a machine designed by another manufacturer

compiler - Converts readable source to equivalent machine operations. Linking or loading of object code is required to finalize the machine code according to the meanings set in a run-time library.

constant - Data whose meaning does not change throughout the program (see variable)

control (as in returning control) - Using the RETURN statement to return the computer to its location prior to executing a subroutine (see the RETURN and GOSUB statements in Chapter 5)

convention - A meeting of people interested in a particular subject or group of subjects; rules of a particular language.

current input device - The input device, such as a keyboard or digital data pack drive, from which the computer is instructed by the IN# statement to receive data

current output device - The output device, such as a screen or a printer, to which the computer is instructed by the PR# statement to send data

cursor - A marker that shows where the next character will appear on the screen. Common cursor markers are a reverse-video block, a blinking reverse-video block, an underline, or a blinking underline.

CURSOR DOWN key - Moves the cursor to the same position in the next line

cursor keypad - A group of keys on the keyboard, including the CURSOR LEFT, CURSOR RIGHT, CURSOR UP, CURSOR DOWN, and HOME keys

cursor keys - Keys used to move the cursor

CURSOR LEFT key - Moves the cursor one position to the left

cursor movement - Used to refer to the activity of the cursor

CURSOR RIGHT key - Moves the cursor one position to the right

CURSOR UP key - Moves the cursor to the same position in the previous line

D

data - Information

data entry - The process of entering data into a computer through the keyboard or other input device (similar to data input)

data input - The process of inputting data into a computer through the keyboard or other entry device (similar to data entry)

digital data pack - A storage device, similar to a cassette, that is designed, manufactured, and sold by Coleco especially for the ADAM computer; estimated to hold the equivalent of up to 125 typewritten pages

digital data pack drive - A device used to record data on digital data packs; designed, manufactured, and sold by Coleco especially for the ADAM computer

diskette - A common computer storage medium that comes in various sizes, commonly 5 1/4" or 8"; comes in single-density, double-density, single-sided, doubled-sided, soft-sector, and hard-sector varieties

diskette label - A self-adhering paper label for a diskette. You should NOT write on a label that is already on a diskette; the indentations from a pen or pencil can damage the diskette and the data on it. If you must write on the label, use a felt tip (not plastic tip) marker.

disk - A term used to refer to either a flexible diskette or a hard disk

disk drive - The drive that runs a flexible, removable diskette (diskette drive) or rigid, nonremovable disk (hard drive). The ADAM memory console is more like a cassette recorder than a disk drive.

document - Any text that includes a paragraph, chapter, memo, letter, note, report, novel, or short story

E

editor - A program used to enter and edit programs; a person who edits documents

entry - Data entered into the computer

error messages - Messages that appear on the screen when an error is made; may appear while you are programming or running the program

execute - The process by which the computer follows instructions given by the programmer

expansion slot - An area on the computer (hardware) where input and output devices are connected. The ADAM has three expansion slots inside the memory console or module.

expression - String or numeric value

F

file - An organization method used to find the beginning and end of a document or other group of related information. Files are created, saved, updated, and removed from digital data packs or diskettes.

firmware - An item that has traits of hardware and software, such as a chip with a program encoded on it. For example, the SmartWRITER word processor is encoded on a chip.

floppy - A nickname for a flexible diskette

flowchart - A chart that shows the progression of a program

flowcharting - The process by which a programmer creates a flow chart that represents a program. Flowcharting is useful in working out the bugs in program logic before spending time and effort in coding the program.

FORTRAN - A programming language used primarily for programming the solutions to scientific and engineering problems

function - The purpose of a task, activity, or operation; the task itself; or a numeric or string function in SmartBASIC or other programming language

function keys - Keys, such as the SMARTKEYS, that are used to begin word-processing or data processing activities. Function keys are also called command keys.

G

graphics - Statements and functions used to plot points to create graphs, horizontal lines, vertical lines, blocks, and geometric shapes for use in activities including business analysis, scientific analysis, mathematical analysis, architectural analysis, and games

H

hard disk - A device that stores data at the computer and, unlike a flexible diskette, usually cannot be moved from the computer area (see flexible diskette)

hard copy - A program or document printed on paper

hexadecimal - A base sixteen numbering system used for programming in (nearly) machine language

high-level language - A language that more closely resembles English than it does a machine language or a low-level language

I

increment - Increase; usually refers to passes through a loop. After each pass a counter can be incremented by 1. The opposite is decrement or decrease.

information - Data

input - Data, such as a list of names or a response to a screen prompt

input device - Any device, such as a keyboard, joystick, or digital data pack, used to input data

implement - To start a procedure, process, or project

integer - A whole number, that is, a negative or positive number without a fraction

ICB - Integrated circuit board

interface - Addressing an issue; communicating between hardware and software, hardware and hardware, or software and software; or hardware or software that interfaces

internal memory - Memory in the computer; also known as ROM (read-only memory) and RAM (random-access memory, also known as read and write memory)

invoke - Start or call up a procedure

K

keyboard - Input device with standard typewriter keys, cursor keys, and command keys

L

language - The characters and rules for combining those characters used in programming

LET - An optional assignment statement. For example, LET A$ = 0 is the same as A$ = 0.

line - Up to 31 characters when programming on the text screen in SmartBASIC or up to 36 characters when using SmartWRITER; up to 80 characters on a printout

line number - A number that identifies the line; generally incremented by 10 to leave space to insert other lines

loop - A set of instructions that are repeated in the same sequence one or more times with the same or a different set of data each time

logical operator - Used to compare the values of numeric or string variables

low-level language - A programming language that is closer to machine language than to English

M

machine language - The binary language into which the programming language is translated by the compiler

memory - The device used for temporary storage of data in the computer, also known as ROM (read-only memory) and RAM (random-access memory)

modem - Abbreviation for modulator/demodulator, a device used to communicate between computers over phone lines

N

network - Two or more computers or computer peripherals that can communicate

O

open (a file) - Procedure done before data can be read from or into a file

operating system - A set of programs that tell the computer how to respond to application programs

operator - A symbol that indicates what kind of operation is to be performed. In SmartBASIC, there are three kinds of operators: arithmetic, logical, and relational.

output - Data that has been processed and is ready to go to an output device

output device - Any device used to output data, files, and reports, such as a monitor, diskette, diskette drive, or printer

P

PL/1 - A program language most often used for insurance and other business applications

precedence - The order in which portions of formulas are executed

program - A set of instructions that can be interpreted by a computer

programming - The process by which a programmer designs, codes, tests, and debugs a program

programming technique - The most effective methods used to create a program

programmer - A person who writes programs

prompt - A question or other message displayed on a screen, eliciting a response from the user

R

RAM - Acronym for random-access memory, or read/write memory, a form of internal memory that the computer can read from and write to

report generator - A program whose easy-to-read statements make report generating easier

ROM - Acronym for read-only memory, a form of memory that the computer can read from, but cannot write to

routine - A group of program lines that act together to instruct the computer to complete an activity, such as center a line on the screen or print a report (see the GOSUB statement in Chapter 5)

S

screen - That part of a TV or monitor where data is displayed; a screen full of text; the appearance of the screen

screen design - The process of designing how the screen appears when a program is running (see Chapter 6)

sign - The minus, or negative, sign (-)

SmartBASIC - Written for the ADAM, SmartBASIC is a programming language that is, for the most part, Applesoft source compatible.

SmartWRITER - A word-processing system encoded on a chip in the ADAM memory console

statement - An instruction that the computer can interpret and carry out (see Chapter 5 and Appendix B for a list of SmartBASIC statements)

storage device - A device that stores data, documents, and programs. ADAM's storage device, called the memory console, is used to record data on digital data packs.

subroutine - A routine that can be repeated through use of the GOSUB or ON . . . GOSUB statements

syntax - As in any language, the structure of statements and functions that a computer can interpret and carry out

system - A group of related programs, such as a word-processing or accounts receivable system

T

text - Data composed of letters and numbers that are processed but not accumulated

V

vertical spacing - Spacing between lines on a page, typically in single, double, or one-and-one-half spacing; also referred to as line spacing, especially in SmartWRITER word processing

W

whole number - Also called an integer, a whole number without a fraction. When real numbers are changed into integers through the INT (x) function in SmartBASIC, the fraction is truncated, not rounded to the nearest high number. Truncation is something to consider when negative numbers are being used.

word processor - A program, such as ADAM's SmartWRITER, used primarily to type, edit, format, save, and print text rather than accumulate numeric data

Index

Assistant to the Managing Editor
Tim P. Russell

Production
Dennis R. Sheehan

Composed by Que Corporation
in Times Roman, Megaron, and Varityper Digital

Printed and bound by
Fairfield Graphics, Fairfield, Pennsylvania

Cover designed by Cargill Associates, Atlanta, Georgia

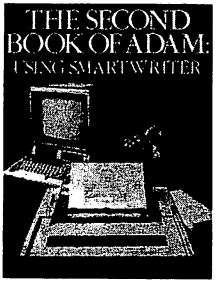
Artwork by Dennis R. Sheehan

More Computer Knowledge from Que

The First Book of (Coleco) Adam	$12.95
The Second Book of (Coleco) Adam: Smartwriter	9.95
TI-99/4A Favorite Programs Explained	12.95
Timex/Sinclair 1000 Dictionary & Reference Guide	4.95
IBM's Personal Computer, 2nd edition	15.95
IBM PC Expansion & Software Guide	19.95
PC DOS User's Guide	12.95
MS-DOS User's Guide	12.95
Spreadsheet Software: from VisiCalc to 1-2-3	15.95
Using 1-2-3	14.95
1-2-3 for Business	14.95
1-2-3 Tips, Tricks, and Traps	12.95
Using Microsoft Word	14.95
Using Multimate	14.95
Introducing IBM PC*jr*	12.95
Teaching Your Kids with the IBM PC*jr*	12.95
Real Managers Use Personal Computers	14.95
Multiplan Models for Business	14.95
SuperCalc SuperModels for Business	14.95
VisiCalc Models for Business	14.95
Using InfoStar	16.95
CP/M Software Finder	14.95
C Programming Guide	17.95
C Programmer's Library	19.95
CP/M Programmers's Encyclopedia	19.95
Understanding Unix	17.95
Commodore 64 Favorite Programs Explained	12.95
HP 150 Fingertip Computing	19.95
Networking IBM PCs	17.95
Improve Your Writing with Word Processing	12.95

ORDER FROM QUE TODAY
1-800-428-5331

LEARN TO MASTER SMARTWRITER™ WITH *THE SECOND BOOK OF ADAM*



Pam Roth, author of *The First Book of ADAM*, has written another book to help you get the most from your ADAM home computer. *The Second Book of ADAM: Using SmartWRITER* tells you all about the remarkable word-processing capability of the Coleco ADAM. In an interesting and easy-to-read style, the author explains the functions of writing, printing, saving, copying, moving text, searching and replacing, and much more. Practical examples allow you to begin using SmartWRITER almost immediately. With this book, you will soon be using your ADAM computer for all your home typing needs.

To order, use this form, or call Que at 1-800-428-5331.

Item	Title	Price	Quantity	Extension
111	The Second Book of Adam: Using SmartWRITER	$ 9.95		
			BOOK SUBTOTAL	
			Shipping & Handling ($1.50 per item)	
			Indiana Residents Add 5% Sales Tax	
			GRAND TOTAL	

Method of Payment:

☐ Check *Charge My:* ☐ VISA ☐ MasterCard ☐ American Express

Card Number _____ Exp. Date _____

Cardholder Name _____

SHIP TO: _____

Address _____

City _____ State _____ Zip _____

FOLD HERE

Place
Stamp
Here

Que Corporation
7999 Knue Road, Suite 202
Indianapolis, IN 46250

The new ADAM computer from Coleco has been described as one of the best values among home computers. *The First Book of ADAM* is designed to introduce you to the ADAM—both the Family Computer System and the Family Computer Module, which plugs into the ColecoVision video game system.

Inside you will find a description of each of the ADAM's components, instructions for programming the ADAM, and definitions of many computer terms. You do not need experience with computers or programming to use this book. Clear explanations and helpful exercises lead you through each step of using the computer and writing programs in SmartBASIC. With this book, you can quickly learn how to design and write your own programs for games, planning, graphics, and more.

The First Book of ADAM will also tell you about the many uses for this versatile computer that are available through the ADAM's graphics capability and packaged software. Included is a discussion of SmartLOGO, SmartFILER, games, and educational programs that are now available or will soon be available for the ADAM. A detailed explanation of ADAM's built-in word processing program, SmartWRITER, can be found in the next book of this series, *The Second Book of ADAM: Using SmartWRITER,* also by Pam Roth.

If you own an ADAM computer or if you are considering buying one, *The First Book of ADAM* can answer many of your questions. Read this book for an excellent introduction to this exciting new home computer!

que™

0-88022-063-5