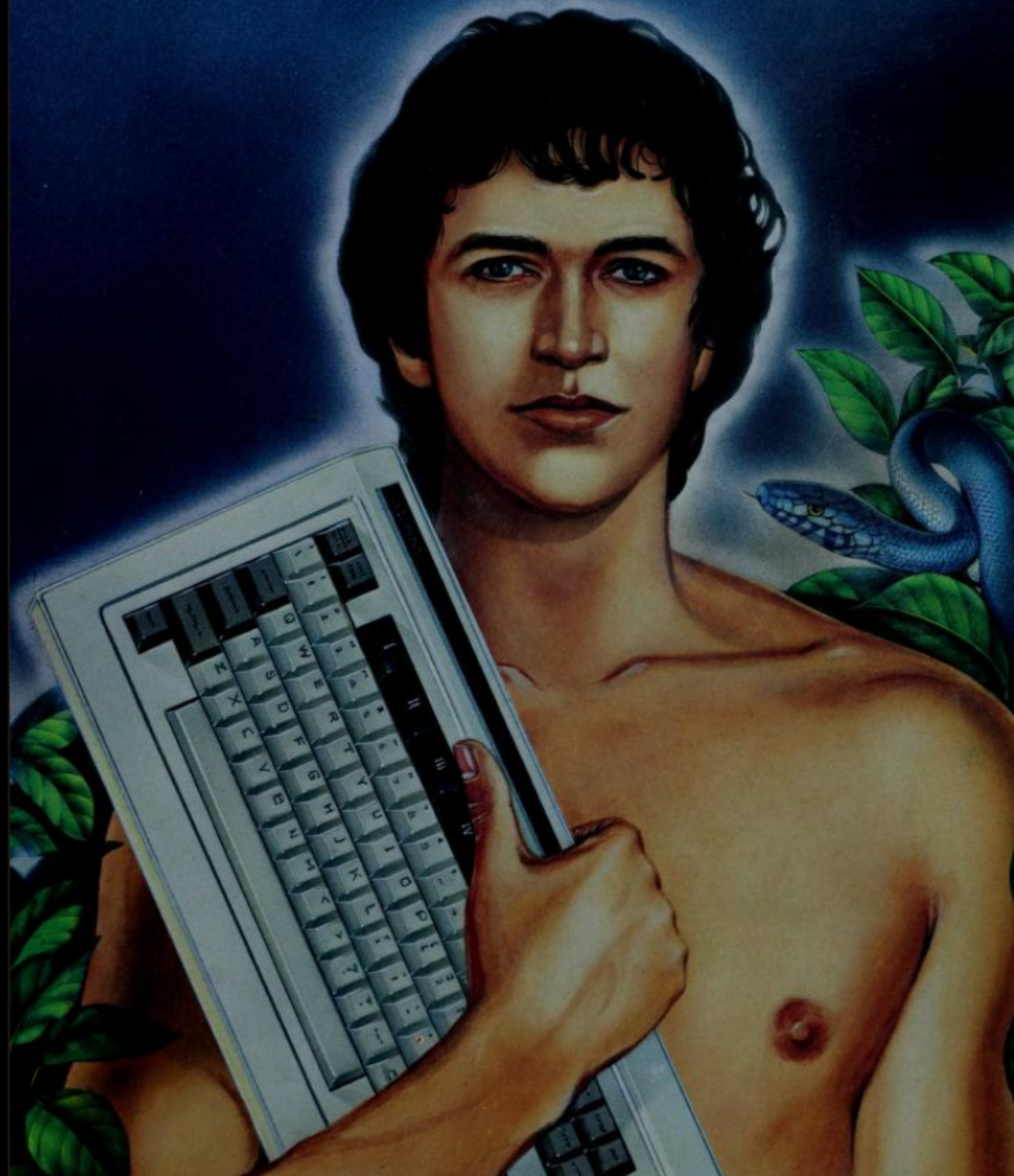
SYBEX COMPUTER BOOKS

THE EASY GUIDE TO YOUR COLECO ADAM™

THOMAS BLACKADAR



The Easy Guide to Your Coleco Adam

The Easy Guide to Your Coleco Adam™

THOMAS BLACKADAR

SYBEX®

Berkeley • Paris • Düsseldorf

Cover art by Sato Yamamoto
Book design by Mary Rose Ogren

Library of Congress Card Number: 84-50364
ISBN 0-89588-181-0
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Acknowledgments

A book such as this is always a joint project, and many people share the credit. I'd particularly like to thank Rudolph Langer, Paul Panish, and David Kolodney for their support and advice in the development of this project. Joel Kreisman contributed valuable technical assistance at all stages, and Amy Einsohn gave it a careful and timely editing. Thanks also to these wonderful people who did so much to turn a manuscript into a book: Valerie Robbins and Sarah Seaver, word processing; Gloria Maya, typesetting; Janis Lund, proofreading; and Mary Rose Ogren, design.

At Coleco, Kathleen McGowan and her staff were extremely helpful. I'd like particularly to thank Andi Freed for promptly sending advance copies of Adam documentation. Without this support, this project would have been more difficult.

Table of Contents

SECTION 2

The Word Processor

SECTION 3

Basic Programming

Appendices

Introduction

Coleco advertises its Adam™ computer as an "all-in-one" system. The box you buy includes the computer, the keyboard, the printer, and a built-in word processing program. With just these parts and a television, you should be able to use the Adam as a full computer or word processing system.

You can, but you will need a guide. You might think you can learn the system by reading the manuals that Coleco provides. If you are like most people, you will find that you won't be able to. Coleco's manuals are difficult, disorganized, and at times misleading.

This book can be your guide, helping you learn quickly what you need to know about your computer. In the course of this book, you will set up your computer and begin to use it. You will learn about the built-in word processor and find out where you can buy other programs that let you start using your computer right away. Then you will learn how to write your own programs, and how to store them on cassettes.

What do you need to know beforehand? Nothing. This is an *easy guide* that assumes you have never used a computer before. You can start from the very beginning, and move slowly enough to master the concepts without needing to cope with computer jargon.

If you know a little already, you can still use this book. You may want to skim through sections that you already understand, then read more carefully once you get to the sections you still need to cover.

I have tried to make this book both serious and enjoyable. I have assumed that you are truly interested in learning about your Adam, and that you want a clear, no-nonsense introduction. At the same time, I have concentrated on the most attractive, entertaining features of your Adam.

The chapters are grouped into three sections, to help you organize your explorations. The first section, Getting Acquainted, will help you learn the fundamentals of your machine. You will find out what the Adam is, how to set it up, and how to buy programs that will let you use your computer without knowing anything about how it works.

Section Two treats the word processor that is built into the Adam. This section is kept intentionally short, since greater detail would tend merely to confuse the point. While this brief introduction may skim over some of the more specialized features of the Adam word processor, it will explain everything you'll need to know to create simple letters and reports on your computer. If you want more information, you can read one of the many books devoted to the Adam word processor.

Chapter Four, at the end of the Section Two, concerns the Adam's high-speed cassettes, or *digital data packs*. While this chapter is part of the section on word processing, you should read it no matter what you plan to do with your Adam, since you will want to use the cassettes to load and store programs. Other chapters may occasionally refer you back to these descriptions.

Section Three moves into programming. The Adam system you bought includes a version of the BASIC programming language. By using this language you will be able to make your computer do exactly what you want. You can start by typing simple commands that do one thing at a time. Later, you can combine your commands into larger programs in order to perform more complex tasks. BASIC is one of the computer languages most commonly used today.

To use this book most effectively, you will need to have access to an Adam computer. For most of the book, you should have your computer sitting in front of you, so that you can try out the examples as you read.

It doesn't matter whether you have the full Adam system or the Adam *Expansion Module #3* with a ColecoVision game machine. The set-up procedures in Chapter 1 explain the differences between the two systems. Once you have them set up, the systems work in exactly the same way.

I hope you will enjoy your reading. Computers may seem mysterious while you are an outsider, but after your first steps, you will be able to continue on your own. This book merely aims to make those first steps a little easier.

SECTION 1

GETTING ACQUAINTED

Chapter 1

Meet the Adam

You have recently bought an Adam computer and are anxious to start using it. You are curious to find out what your machine can do and how you can control it.

This chapter is an overview. It gives some tips that will help you become comfortable with your Adam, and detailed instructions on how to set it up. By the end of the chapter, you will be ready to use your computer.

FEAR

Before you begin your journey, you'll need to overcome your fears. If this is your first experience with a computer, you may be a little intimidated by it. You don't have to be: a computer is a machine, nothing more. Like a television set or a dishwasher, the computer can help you if you learn how to use it, and it won't bite you if you make a mistake.

You don't need to worry that you might wreck your computer by pressing the wrong key. Your Adam is an electronic device that deserves the same care you give to a radio or a pocket calculator, but you don't have to be timid with it. There is little you can do in normal use that will harm it. At the very worst, you might lose the words and numbers you have just typed, but you won't hurt the machine.

Your Adam gives you a kind of panic button. This is the RESET COMPUTER switch, located along the top of the computer, just to the left of the cartridge slot. If you ever get lost or don't know what's

happening, you can always pull this switch. Your computer will go back to the way it was when you turned it on, ready to start again. If that doesn't do the trick, you can always turn the power off and back on.

If you have some lingering fears, play a game. There is no better way to become comfortable with a computer than to have fun with it. In Chapter 2 of this book, I'll tell you about some of the best Adam games, but for now you might as well start with the one that came with the machine, Buck Rogers: Planet of Zoom. Adam's home version of this game is a sure-fire cure for computer fear.

WHAT IS A COMPUTER?

How does a computer work? You don't need to know. You can think of it as a closed box that takes in information, churns it around, and spits out the results. Along the way, it must do many complex operations, but that's not something you need to concern yourself with. Just think about what goes in and what comes out.

You talk to the computer through its *keyboard.* The Adam has a full set of typewriter keys that let you type letters, numbers, and special symbols. You will use these keys for all your communications with the computer.

To talk back, the computer has an electronic circuit that can display words and pictures on your television screen. So, as you type letters on the keyboard, your computer shows them on the screen. Then, if it needs to give an answer, it displays that as well. With *graphics,* you can also have the computer draw pictures on the screen.

The computer has a *memory,* so that you can store information for future reference. You can use this memory to save the lines of a program, the numbers you will use in a future calculation, or the words of a text you are writing.

How much you can store depends on the size of your computer's memory, which is measured in *bytes.* A byte is roughly the space needed to store a single letter or a three-digit number. The size of a computer's memory is usually stated as a number followed by the letter K, which represents roughly a thousand bytes (1024 to be exact). One K would hold about one third of this page.

For the purposes of this book, you won't need to worry about the size of your computer's memory. The Adam has 80K bytes built in—more than enough for anything in this book. While you can buy a memory expansion that adds another 64K, you will rarely need this unless you plan to use very complex programs.

When you turn the computer's power off, everything you have stored in its memory is erased. To store information permanently, you must use the Adam's *digital data drive,* which is built into the front of the main computer unit. This is a high-speed cassette player that lets you store information on a cassette tape, so that you can reuse it at a later date. It also lets you load other people's programs into your computer, so that you can use your computer in different ways.

The Adam also comes with a *printer,* which lets it type words on a sheet of paper. The printer is a necessary companion to the word processor, since your ultimate aim is to produce a text that could be mistaken for a typewriter's. In other programs, you may also want to use the printer to obtain a permanent record of your work.

WHAT CAN IT DO?

Your Adam can perform complex calculations quickly and easily, display bar graphs, and paint colorful pictures on your television screen. With its built-in word processing program, it lets you revise a letter without retyping it. And of course, it can play games.

The computer's great strength is its flexibility. It can do calculations if you wish, or limit itself to letters and words. It can react to whatever you type on the keyboard and arrange the screen display as you direct. With its memory, it can store and arrange information in many useful ways.

You control your computer with a *program,* a detailed list of instructions that it follows like a recipe, step by step, to perform its task. The program can be as simple or as complicated as you like, and can ask the computer to make decisions as it works. Programs are often called *software.*

The Adam has one program built right into the machine: the word processor. As you turn your computer on, it automatically begins running this program so that you can use your Adam as an intelligent typewriter. You don't need to know anything about the computer to use this program. You just type and let the computer handle the details. This word processing program is described in Section Two of this book.

When you bought your Adam, another program was included in the box—the Buck Rogers: Planet of Zoom game. You can play this interesting video game without any special equipment—just follow the directions in Chapter 2.

You can buy many other programs from Coleco and other companies. You can use most of this *commercial software* without knowing how it works. You just give your computer general instructions and answer a few questions, then let the program take care of the details. Chapter 2 of this book is an introduction to some of the software that you can buy. Games are very popular, and widely available for the Adam. You can also buy programs for more serious purposes, such as education or home management.

At some point, you may want to write your own programs, so that you can do exactly what you want with your computer and use it to its full potential. In the later chapters of this book, you will learn about the Adam's SmartBASIC programming language, which will let you write your own programs and explore the deeper aspects of your machine.

SETTING UP YOUR COMPUTER

The Adam comes in a very large box. When you open it up, you find three large pieces of equipment and a variety of smaller items. The three main pieces are the printer, the keyboard, and the computer itself. Once you have connected these, you will be able to use your computer immediately.

The only thing you'll need that does not come with your computer is a television set. You can use any set, but you will get the best results from newer models. Televisions more than ten years old may give a jittery screen image.

A color television is strongly recommended. Your Adam has an excellent color graphics system that can display brilliant colors. Many of the programs in this book take advantage of these displays. While you can run them on a black-and-white television, you will be missing a lot.

If you are selecting a television to use with your computer, you should probably choose a small or medium-sized screen. Screens measuring 10 to 15 inches are quite large enough to be readable. Larger screens are unnecessary and often harder to read.

The Adam also allows you to hook up a *color monitor* instead of a television. A monitor looks like a television, but lacks a tuner and thus cannot be used to receive broadcasts. A monitor will usually give a clearer picture than a regular television.

Coleco sells two versions of the Adam. One is the "full" Adam computer system, which you can use all by itself. The other is the *Adam Expansion Module #3,* designed to work with the ColecoVision video-game system. If you already own a ColecoVision, you can save over a hundred dollars by buying the expansion system; otherwise, just buy the full Adam. Once you have everything connected, the two versions work identically.

You can do the set-up yourself with nothing more than a screwdriver. The *Adam Set-Up Manual* that comes with your computer contains detailed instructions on setting up the machine, complete with pictures. Unfortunately, you will get so bogged down in the details of that manual that you may never get through it. I suggest you try my instructions instead: I have arranged the steps in an order that should be easier to follow. Don't be concerned if I tell you to do things in a slightly different way than the manual. Both methods will work.

When you open the box, lay each of the pieces out on the floor. Unpack the box carefully, and make sure you don't lose anything. Once you've sorted through the styrofoam, you should find three major items: a printer, a keyboard, and the computer itself.

The first step is to hook up the printer. All the electricity for the entire Adam system must go through the printer, so you must have it connected even if you don't plan to use it. Fortunately, the hookup is easy.

Set the printer on a flat surface. Remove the styrofoam packing material that is holding the ribbon and print wheel in place, then turn the unit around, as in Figure 1.1, so that you can see where you will be making your connections. Never set anything on top of the printer: it's a mechanical beast that needs space to operate. Above all, never put a cassette tape on top of the printer, since its motors can erase the magnetic recordings.

Find the power switch on the back panel and make sure it is in the OFF position. While the Adam's cables are not dangerous to touch, it is a good practice always to turn the power off before making connections.

The Adam printer has two cords coming out of the back, as you can see from Figure 1.1. One is a long black cord that you can

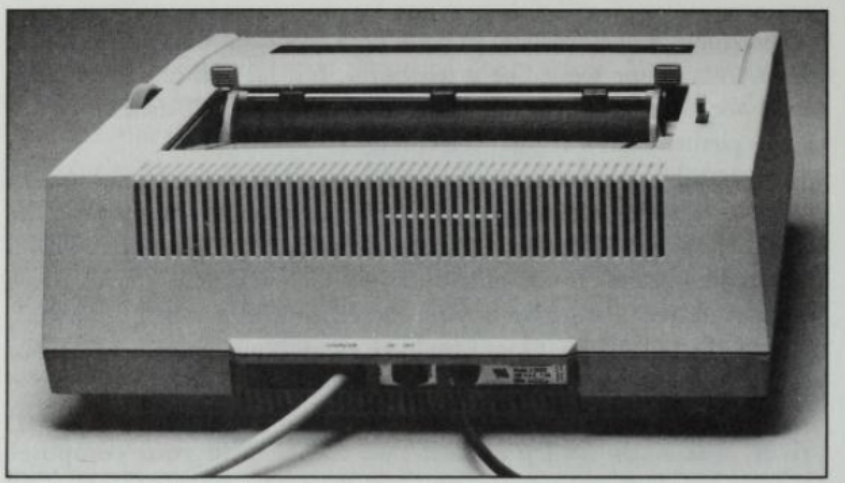


Figure 1.1: *The back of the Adam printer, where you will make your first connections.*

connect to a power outlet in your house. Don't connect this yet, but remember that it is the source of power for the entire system.

The other cable coming out the back of the printer is a gray cord with a long connector on the end. This is the umbilical cord that links the printer to the rest of the system. Through it, the computer will obtain its electricity; in return, it will send electronic signals back to the printer whenever it needs to type something on paper.

Set the main computer unit on your table and connect the printer's gray cable to the slot on the left side. It will fit only one way, with the cable going out the back. All the remaining connections take place on the back of the computer.

If instead of the full Adam you are using the Adam Expansion Module #3 for the ColecoVision video-game machine, there is an extra step at this point. Put your ColecoVision on the plastic tray that came in the box with your Adam. Lift the black plastic door on the front of the game machine, then slide the Adam Expansion into it. By doing this, you are connecting the Adam Expansion to your ColecoVision, so that the two will act as one unit. Except for the placement of the connections, the combined system will work exactly like the full Adam. All the power for the system will come through the Adam printer: you will no longer use the game machine's power source.

Whichever version you have, you now need to make some connections to the main computer box. All the following steps refer to the

connections on the full Adam system. If you have a ColecoVision game machine and the Adam expansion, the connections will be the same, but some of the terminals will be on the game machine, rather than on the Adam.

The next step is to connect your computer to your television. For this, you will need to use the small *switch box* that came with your computer. These come in several forms, but yours probably looks like the one in Figure 1.2.

The switch box allows you to connect your television to your Adam, yet still use it for regular TV reception when you aren't running the computer. The box has a small black pin that you can slide between two positions. When it is in the COMPUTER position, your television will display the computer's images. Slide the pin back into the TV position to watch television programs.

Start by connecting the switch box to your television. Disconnect the antenna or rabbit ears. On most sets, the antenna is connected by a double cable to two screws labeled VHF on the back of the set. Loosen these screws and remove the wires. You don't need to disconnect the UHF antenna, if you have one. (If you are connected to a cable TV system, see below.)

The switch box has a pair of short wires coming out one end. Connect these wires to the two screws you have just loosened, then tighten the screws.

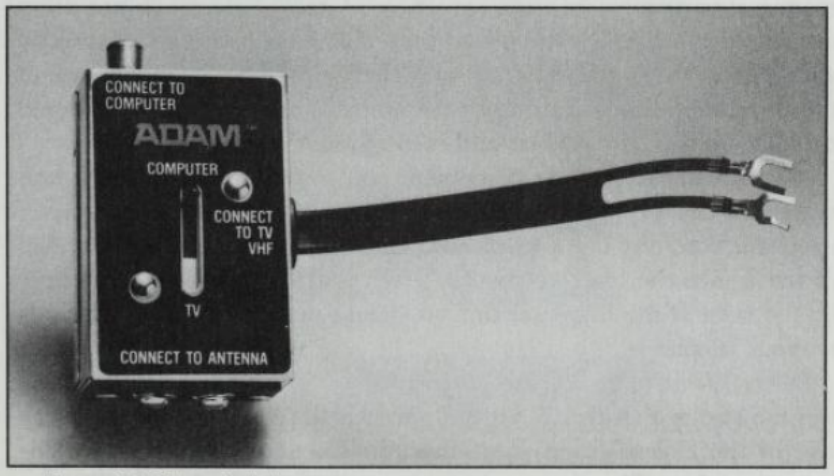


Figure 1.2: *The switch box.*

On the side of the switch box with the label CONNECT TO ANTENNA, you will find another pair of screws. If you plan to use your set to watch television programs as well as your computer display, loosen these screws and attach the wires from your television's antenna. When the switch is set in the TV position, your antenna will be connected just as if you had never removed it from the set.

If you subscribe to a cable network, your television is probably connected with a single round plug instead of a pair of screws. If so, you have to buy an adapter to connect to the screws. You can find one in a television or electronics store. Try Radio Shack.

Find the black cable that comes with the computer. It should be about six feet long, and have a round metal plug on each end. Connect one end to the hole in the COMPUTER side of the switch box. Then connect the other end to the hole labeled TV on the back panel of the computer. If you have the ColecoVision game machine and the Adam Expansion Module, use the terminal on the back of the game machine.

You can have your television receive the computer's signal on either channel 3 or 4. You will generally get less interference if you choose the one not used by a television station in your area. Set the channel-selector switch on the back of your computer, then tune your television to the same channel.

For simplicity, I have neglected to cover a few possibilities here. You may have a television that has only a round connector, or an antenna attached through a round "coaxial" cable. If so, you should read the instructions in the *Adam Set-Up Manual*. Also, if you are using a monitor instead of a television, you will need to connect it directly with a cable to the MONITOR or VIDEO OUTPUT on the back of the computer, rather than using the switch box. Your monitor should contain the necessary cables and instructions.

The last major piece of equipment you must connect to the Adam system is the keyboard. Find the white cable that looks like your telephone cord and has a small "modular" plug on each end. Plug one of these into the back of the keyboard and the other into the hole on the front of the computer unit. It should snap in, just like the cords on your telephone.

If you have followed these directions, your system should look like the one in Figure 1.3. At the center is the Adam computer module (or the ColecoVision game machine linked to the Adam expansion module). The three other pieces are attached to it like the spokes of a wheel: the printer through its gray umbilical cord, the television through the switch box, and the keyboard through its telephone cable.

That's it for the essential connections. You can now use your Adam as a full computer system, complete with keyboard, screen, and printer. Nothing else is necessary, but you might want to connect the other items while you're at it.

Figure 1.3: *The full Adam system consists of the computer console, the printer, the television, and the keyboard.*

The most important pieces that remain in the box are the two *game controllers,* or *joysticks,* as they are called. These are a pair of white boxes with small round handles and a numeric keypad. If you play a game, you can use these handles to move your characters around the screen. The joysticks also have two *fire buttons,* located along the left and right sides. The buttons serve various purposes, usually to fire a shot of some kind.

Along the right side of your Adam (or on your game machine if you are using the ColecoVision Expansion), you will find two rectangular holes. These are where you connect the joysticks. Generally, you will be using only the first of the two joysticks, saving the other for two-person games.

If you want, you can attach one of the joysticks to the side of the keyboard, using the plastic holder that comes with your computer. By doing this, you can move around a word processor text by pressing on the joystick handle. You can also use the joystick's keypad as a convenient alternative for entering numbers. The second joystick must remain loose.

If you check the machine's packing box, you will find a few other items, including three cassette tapes. Put these aside in a safe place. You will need them later.

Now it's time to turn on the power. Attach the power cable from the back of the printer to an electric outlet in your house. As you can see, the Adam has a three-pronged plug, which must be connected to a three-pronged outlet. If you have only two-pronged outlets in your house, you can use the adapter that Coleco provides (except in Canada, where such adapters are illegal). Make sure you connect the adapter to the central screw on the wall plug. This will help to protect the computer and make it safer to operate.

Now turn on the power— first on your television, then on the back of the printer. The printer should whirr a bit, then stop with a clack. Your television, once it has warmed up, will show the message ADAM'S ELECTRONIC TYPEWRITER, as in Figure 1.4. If the image is fuzzy, adjust the fine tuning on your television. If the screen's background isn't light blue, or if the box displaying the message isn't yellow, adjust the color, tint, and brightness controls on your set. You may need to turn off the automatic fine tuning (AFT) or automatic color controls: some televisions provide a better image when adjusted manually.

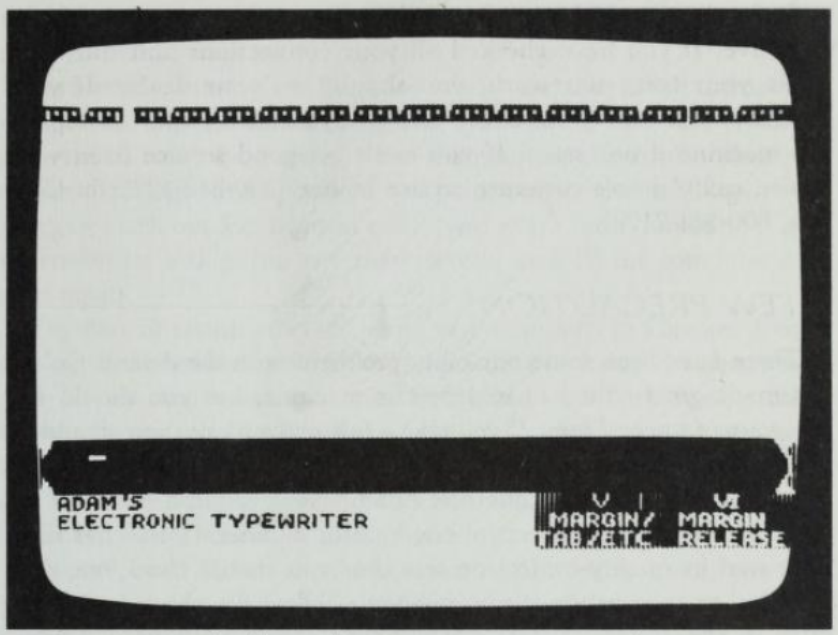


Figure 1.4: *You should see this picture on your screen when you turn on your Adam.*

If you can't get any picture or can't get a sharp image, you may have done something wrong. The problem could be any of the following:

- You have forgotten to turn on the computer or the television, or have forgotten to plug them in.
- You have not connected the cable from the printer to the computer. All the power goes through this cable, and nothing will work if it isn't attached properly.
- The cable from the computer to the switch box is not connected securely. Try wiggling the connections to see if this helps.
- You have not securely tightened the wires from the switch box to the VHF screws on the television.
- The switch box is not in the COMPUTER position.
- Your television is not tuned to the same channel (3 or 4) as your computer. Try changing to the other channel.

Sadly, there could be another explanation: your computer might be defective. If you have checked all your connections and still can't make your computer work, you should see your dealer. If your machine is defective, the dealer can give you advice and can replace the machine if necessary. If you can't get good service from your dealer, call Coleco's customer service hotline (800-842-1225; in Canada, 800-361-2122).

A FEW PRECAUTIONS

There have been some reliability problems with the Adam. Coleco has made great efforts to improve its machine, but you should still be aware of the problem. If you take a few precautions, you should be able to avoid most troubles.

In the first few months after the Adam was introduced, many of the units sold had minor defects of one kind or another. Coleco has since improved its quality-control process, but you should check out your machine as soon as you bring it home. Follow the check-out procedure in the *Hints and Tips* booklet that comes with your Adam. If there is a serious problem, it will probably show up in these tests. If you find anything wrong, call your dealer immediately, while you're covered by the warranty.

Leave your machine on overnight during the first week. Most defects show up in the first 24 hours of use. By leaving the machine on, you can see immediately whether it is going to fail. This test doesn't use much electricity, and it won't hurt the machine. If the computer survives this initial *burn-in,* it will probably work well for a long time. If any problems show up during this burn-in, you're still covered by the warranty.

Observe two precautions during this burn-in procedure. First, make sure the printer has enough ventilation—it will get fairly hot if it's not out in the open. Second, turn off the television that you have connected to the computer. You can permanently damage the phosphors on the TV screen by displaying the same image for a long time.

Even if your Adam is working perfectly, you may occasionally run into a few minor design flaws. Both the built-in word processor and the SmartBASIC programming language have some features that just don't work the way they are supposed to. These *bugs,* as they are called, are part of the machine itself, so you have to learn to work around them.

Fortunately, most of the bugs are minor, and they don't disrupt the computer's important functions. Still, you should be aware of them, so that you won't be puzzled when the machine doesn't do what you expect. Throughout this book, I will point out some of the bugs you'll need to watch out for. In most cases, you won't harm anything if you do encounter a bug, but you may have to turn off the computer and start again.

Pay careful attention to the series of precautions in Chapter 4 concerning the cassette tape drive. The Adam's high-speed cassette system is very convenient but must be treated with considerable care. Cassette tapes must be handled and stored correctly to prevent damage. Read pages 54-56, near the beginning of Chapter 4, before you use a tape for the first time. If you follow these guidelines for handling cassettes, you should have little trouble.

Overall, your Adam should be quite reliable. If it is not, you will probably discover the problem before the end of the warranty period, and can exchange it for a unit that will give you good service.

We have now reached the end of the preliminaries. Your machine is set up, and you are waiting to go. With the next chapter, you will start using it. Good luck, and enjoy yourself.

Chapter 2

Commercial Software

In the first chapter, you learned something about how your Adam computer works. You set it up, turned it on, and saw the message ADAM'S ELECTRONIC TYPEWRITER on your screen. Now you are eager to do something.

There are two ways you can approach the Adam for the first time. Your television screen suggests one of these: with no additional equipment, you can use the Adam's word processor as a typewriter.

The other way to begin using your computer is to buy programs written by other people. In many cases, you can simply load a program into the computer and start it running. You don't need to know anything about how the computer works or how to write programs.

This chapter is devoted to this second approach. You will learn how to play the game that comes with your computer—Buck Rogers: Planet of Zoom. You will learn what other kinds of software you can buy and how you might use them.

If you are primarily interested in the word processor, you should skip straight to Chapter 3. Nothing in that chapter will assume knowledge of anything you might read here.

You can buy all sorts of software from Coleco and other manufacturers. Games are the most popular, but excellent programs are also available in other areas, such as education, home finance, and music.

This chapter begins with a description of the Buck Rogers: Planet of Zoom game. With some hints on starting the game and a few basic strategies, you will be able to play immediately.

The rest of the chapter discusses some of the other programs that you can buy for your Adam. While this survey is not a full buyer's

guide, it includes some tips on good programs. It also mentions the things you should know as you shop, so that you can protect yourself from some of the bad programs that are sold alongside the good.

This chapter ends with some hints on how to use the programs you have bought. Although most programs come with an instruction book that explains the details of operation, my advice can help get you started.

BUCK ROGERS: PLANET OF ZOOM

If you like games, you might want to start by playing the one that comes with your computer—Buck Rogers: Planet of Zoom. This game, based on the Buck Rogers novels, was originally a video-arcade game, made by Sega Enterprises. Coleco bought the rights to adapt the game and sell it as a part of its Adam system. While it is hardly the best game available for the Adam, you may as well start with it, since it's free.

In the game you fly a spaceship through a variety of terrains. The object is to shoot as many flying saucers as you can within the time limit. There are ten different screens, or *sectors,* in each round of the game, and each presents a variety of dangers, enemies, and obstacles. As your skill increases, you will work your way through each of the sectors, until you reach the enemy's mother ship. If you can disable this ship, you fly into a space-warp tunnel and reach the next round of the game.

The first screen, shown in Figure 2.1, is a long trench. As you fly forward, the sides and floor seem to move past you. By moving your joystick handle, you can go in any direction you want. Left or right moves the spaceship horizontally. Up or down changes the altitude, as you can see by watching the shadow on the floor. Your range of motion is limited so that you cannot crash into the walls or climb out of the trench.

In this first screen, you face three types of attackers. The most common are the flying saucers, which swirl in from behind and circle back to collide with you. Occasionally, you must also shoot or avoid a bomb that falls slowly from the top of the screen and explodes in front of you. Finally, at the end of the level, you must face a number of *bouncing tripeds,* which jump around the trench and try to land on top of you.

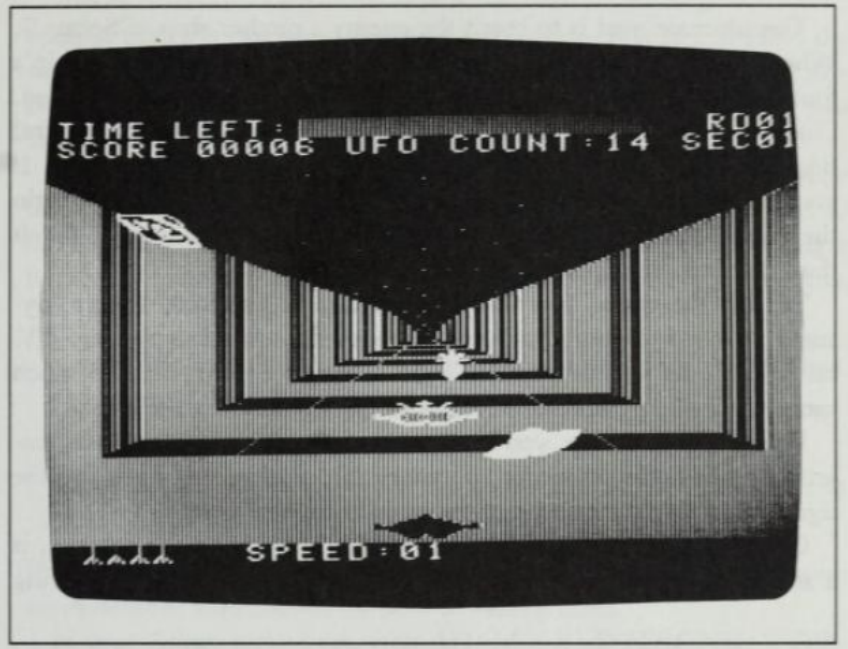


Figure 2.1: *The first sector of Buck Rogers: Planet of Zoom.*

On each side of your joystick, there is a small *fire button.* With the right-hand button, you can shoot forward to destroy attackers, so that you can gain points and move safely toward the end of the trench. The left-hand button applies power to the spaceship's engines, so that the game moves faster.

At the top of the screen, a small colored bar shows the time that you have left in the trench to shoot the attackers. Below that is a number that represents the UFO COUNT, or the number of attackers remaining. If you can shoot all the UFOs in the given time, you receive a bonus and proceed to the next sector. If your time runs out, you do not receive a bonus, but you can still continue with the next part of the game.

In some of the other sectors, you face other forms of attack. Some, for example, put you out in space, where you must shoot saucers and avoid asteroids. In others, you may find yourself above the surface of the Planet of Zoom, trying to fly through obstacles. Sector 3, shown in Figure 2.2, is one of the most interesting. Here you must fly carefully between pairs of futuristic electricity towers, while still shooting saucers.

The ultimate goal is to reach the enemy's mother ship in Sector 9. When you reach this sector, you must shoot out the mother ship's four main engines, while still avoiding the enemy's fighter planes. Once you have disabled the ship, you can shoot through its central door and reach the tenth and final sector, a trenchlike space warp. If you can negotiate this final sector, you return to the first and begin the sequence again. This second round is somewhat more difficult than the first, but presents the same obstacles.

To play the game, you will need to have at least one of your joysticks attached to your computer. You can use a black-and-white TV, but a color set is better: you will be able to see your enemies much more easily if they are set off from the background by their colors.

Before you can play, you have to load the program from the cassette tape into the computer's memory. To do this, you must use the *digital data drive* on the front of the main computer unit.

On the top of the computer, just above the door for the cassette, is a small plastic latch marked EJECT. Pull it, and the door will spring

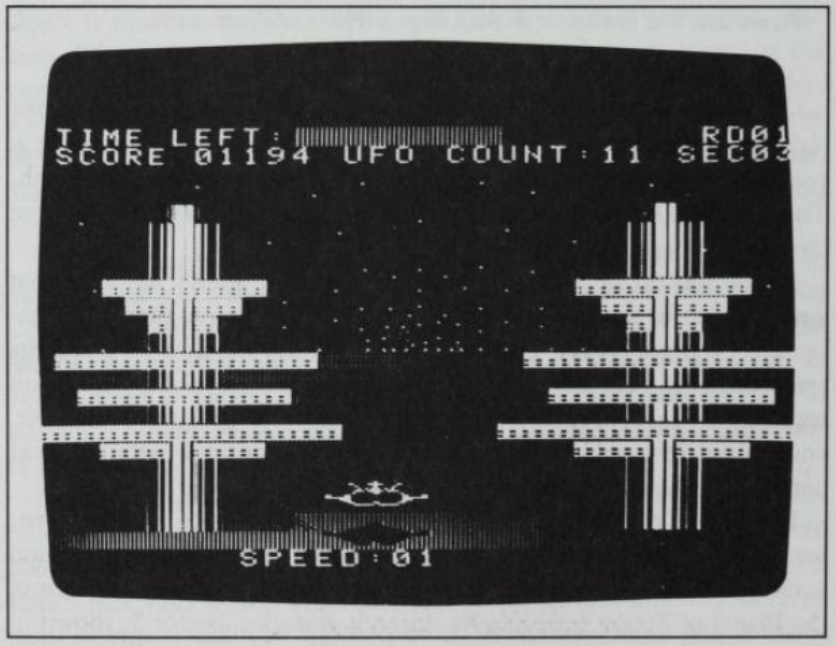


Figure 2.2: *In Sector 3, you must fly between sets of electricity pylons on the surface of the planet.*

open so that you can insert the tape, as shown in Figure 2.3. Hold the tape as shown, with the label facing outward and the recording surface facing down into the drive. If you have inserted it correctly, you can close the door, just as you would on a stereo cassette recorder.

A few precautions are in order at this point. The Adam cassette tapes are rather fragile and easily damaged. If you erase any part of the recording surface or the encoded information, you may no longer be able to load the program. So, be careful to observe these three don'ts:

- Don't touch the magnetic recording surface with your fingers.
- Don't leave the tape in the computer when you turn it on or off.
- Don't put the tapes on top of the printer, the television, or any other device capable of producing a magnetic field.

If you're careful, your tapes should last a long time. Chapter 4 gives more detailed instructions on how to handle tapes.

Is your Adam turned on, with ADAM'S ELECTRONIC TYPEWRITER on the screen? If not, turn it on and wait for the typewriter screen. You must always go through the word processor's initial display on your way to running any other program. While you can have the Adam automatically load a cassette program by turning the computer on with the tape in the drive, you risk damaging your program if you do it that way.

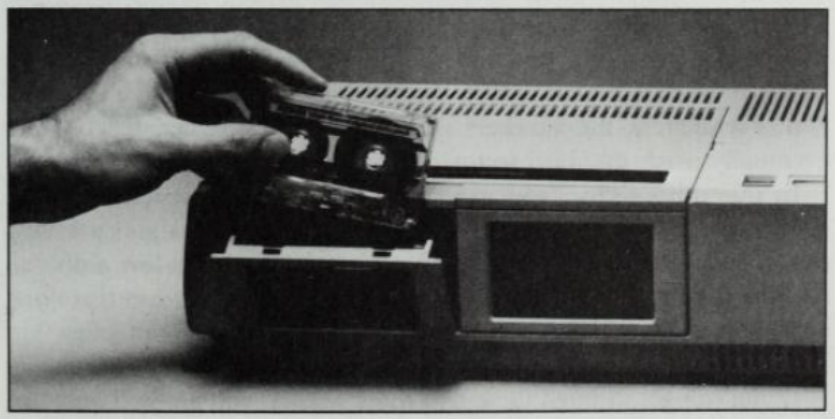


Figure 2.3: *How to insert a cassette tape.*

Find the RESET COMPUTER switch next to the cartridge slot on the top of the computer, then pull it toward you. The screen will go blank and the cassette drive will start to spin as it loads the program. You can use the RESET COMPUTER switch any time you want to start or restart a program.

The computer will start by asking you how many people are playing, and what skill level you want to use. Choose skill level 1 (BEGINNER) unless you have played the game many times.

After a moment, you will find yourself in the first sector, the trench shown in Figure 2.1. Start by practicing your joystick flying controls. They respond more or less the way you would expect, but you may still need a few minutes to become accustomed to them.

Try firing some shots at the flying saucers that appear. Your bullets travel forward down the trench and hit the enemy if correctly aimed. If they miss when it seems they shouldn't, they were probably traveling at the wrong altitude. You cannot really judge this vertical dimension, but you will become better with practice.

Don't be upset if you can't master the game right away. The first few times you play may be a bit discouraging, until you get used to flying and shooting at saucers. Stick with it and keep trying things until something works.

Even if you don't shoot any saucers, you will eventually reach the second game screen just by waiting for the time to run out. This sector takes place in outer space, with a series of colored bars moving above and below to give perspective.

You may find this wave slightly frustrating, since it is hard to hit the UFOs. Unlike the first sector, your shots here travel straight up and down the screen, not into the distance. Thus you may find it rather hard to visualize where they are going and how you should aim. Fortunately, the attackers will rarely hit you either, so you will eventually reach the third sector just by waiting.

In this third sector, you must fly between the electricity pylons, shown in Figure 2.2. The trick here is to fly as low as you possibly can. The distance between the pylons is greater at this low altitude, and the UFOs will rarely run into you down there. You can therefore concentrate on maneuvering between the pylons and can often hit UFOs simply by shooting straight ahead.

If you can get through these first three sectors, you are well on your way. Many of the others are similar to these three and can be handled

in the same way. By reading the instructions on the card that comes with the computer, you can get a feel for what you will be facing in the rest of the game.

After each game, the computer lets you enter your name and save the game's high score. To spell out your name, use the joystick to move the pointer to the appropriate letter on the screen, then press the left fire button. After you've entered all the letters, hold the right fire button down and push the joystick handle to the right until the underline reaches the word DONE at the top of the screen. Then press the left fire button and the game will start over. If you don't want to enter your name, you can skip this whole procedure and restart the game by pressing the * button on the joystick's keypad.

When you've finished playing for the day, you must take the tape out of the tape drive before you turn the computer off. Wait until the tape stops, if it is turning, then pull the EJECT switch on the top of the drive. The end of a game is a good time to remove the tape: the drive always stops turning while it waits for you to enter your name. Put the tape away in its plastic case. If you want to use the word processor or another cassette game, pull the RESET COMPUTER switch. Otherwise, just turn the computer off with the power switch on the back of the printer.

Buck Rogers: Planet of Zoom is a fairly interesting game in some ways. The simulated three-dimensional motion is quite convincing at times, and the action can be enjoyable once you have gotten used to it. Don't judge it too harshly, though, if you think it's not the world's best video game. Coleco included it merely as an example of some of the programs you can buy for your Adam. Better games and other programs are available.

HOW TO BUY SOFTWARE

Coleco sells many programs for its Adam, and many more are available from other sources. From this large selection, you should have no trouble finding some that suit your needs.

Programs are sold for the Adam on cartridges, cassettes, or diskettes. Games tend to be sold as cartridges, although the more complex will soon appear on cassette. Serious programs are usually sold on cassette or diskette.

A cartridge stores its program in an electronic circuit, which the computer can read directly. Because it plugs directly into the computer and cannot be erased, a cartridge is generally more convenient and more durable than other forms of software.

The cartridge slot on the top of the Adam will accept any cartridge made for the ColecoVision video-game system. Since that machine has been popular for some time, many games are readily available for it. If you buy one of these, usually marked COLECO in the store, you can plug it in and use it immediately.

In general, you can use only cartridges made expressly for the Coleco. Game cartridges made for other computers and game machines will not fit in the Adam's cartridge slot, and would not work even if they did. This lack of compatibility is a sad fact of life in the home computer industry.

By buying Coleco's Expansion Module #1, you can gain access to cartridges made for one other type of machine: the Atari 2600 video-game machine (sometimes called the VCS). Since the Atari 2600 is the most widely sold game machine ever, more games are available for it than for any other machine. Atari, Parker Brothers, Activision, and even Coleco make game cartridges for the 2600.

There's a catch, though. The Atari 2600 is a relatively primitive game machine, which is far outclassed by the newer home computers. The graphics, in particular, are notoriously limited, so that arcade games are only a pale reflection of the originals. If you play a 2600 game on your Adam, you will have these same limitations, and you will probably be quite disappointed with it, compared to a game made specifically for the Coleco.

One other note, if you're thinking of buying the Atari expansion module: the Atari 2600 is not part of Atari's line of home computers and will not run the same cartridges. Even if you have the expansion module, you will not be able to use programs written for the Atari 400, 800, or XL series computers.

Another form in which you can buy Adam software is the *digital data pack*, a glorified name for a cassette tape. Coleco's game cassettes have the program encoded as magnetic markings on its surface. With the cassette drive built into your system, the computer can decode, store, and use these messages.

Coleco's cassettes are an improved version of the tapes you might use with a stereo cassette recorder. The computer can read them much faster than an audio cassette, and can automatically find the location of a stored piece of information.

Even so, cassette tapes have several important drawbacks. Before you can use them, you have to load the program into the computer's memory. This procedure can take over a minute for a complex program. Cassettes are also less durable than cartridges, and can be made unreadable by magnetic fields or mechanical wear.

Coleco has recently announced a *disk drive* to provide a more convenient form of data storage. This device reads magnetically coded information from 5¼-inch plates called *diskettes*. You still have to load diskette programs into the computer before you use them, but the procedure is faster and more reliable than with cassettes. Coleco and other software manufacturers plan to sell their cassette programs in diskette form as well, once the disk drives become available. Read Chapter 4 for more information.

Coleco has announced an expansion for the Adam that will let you use programs written for the CP/M operating system. An *operating system* is the computer's underlying housekeeper that keeps track of what the computer is doing. CP/M is one of the most common operating systems for more expensive computers, and many highly respected programs will run with it. While the Adam might not handle all of these perfectly, the CP/M expansion will certainly broaden your choices of software. To use a CP/M program, you will need to have a disk drive.

You can buy programs for your Adam at most places that sell computers or video-game machines. Department and discount stores usually carry some programs, especially the more popular games. Many record and television stores now also carry software for home computers. In large cities, you can usually find stores that specialize in computer software. These are your best bet for finding the less popular titles.

You may need to be a little patient as you shop for programs. The Adam is quite a new machine, and software companies need time to convert their programs. Most game programs appear in the fall, to take advantage of the peak Christmas season.

Coleco has announced a variety of programs for the Adam, in addition to the ColecoVision cartridges it already makes. These were not yet available at the time this book was being written, and one can only hope they will come out soon. Many, however, are popular

programs already available for other machines, so a few generalizations are possible.

A number of independent software companies will also be selling programs that run on the Adam. Coleco has wisely encouraged this practice in the belief that a wide selection of software will help to sell its machines. These independent companies are called *third-party manufacturers.*

If you choose your programs off the shelf, you will learn an unpleasant fact: mixed in with the truly excellent programs are many poor ones. In the fast-moving computer business, some companies can sell inferior products merely by wrapping them in attractive packages. Even Coleco has rushed some programs to market before they were ready. Since you usually have no way to look at a program before you buy it, you can easily make mistakes.

Computer magazines are your best resource for the moment. *Personal Software* is an excellent new magazine, devoted to news and reviews of home computer products. *Compute!* is a general monthly magazine that often reviews new programs offered for home computers. Finally, there is the *ADAM Family Computing* magazine, to which you were invited to subscribe when you bought your computer.

GAMES

Buck Rogers: Planet of Zoom, described earlier in this chapter, is only one of the games you can play on your Adam. Coleco owes some of its popularity to its large selection of video games, many of them adaptations of famous titles from the video arcades. Game programs usually cost between $25 and $40.

Of the arcade games that Coleco sells for the Adam, Donkey Kong is the most famous. It is now available as a ColecoVision cartridge that you can plug directly into the computer. Another version has been announced in the form of a cassette tape that you will be able to read into the Adam's memory.

The object of the Donkey Kong game is to maneuver Mario, the man at the bottom of the screen, so that he can climb to the top and rescue his girlfriend. Donkey Kong, the gorilla at the top, tries to knock Mario out by rolling barrels down the girders on the screen. To stay alive, you must make Mario jump at exactly the right times.

Coleco has licenced a number of other games to sell for the Adam, including Zaxxon, Congo Bongo, and Star Trek: Strategic Operations Simulator. While it is too early to say how well Coleco's versions of these games will turn out, they are all good games in the arcade.

Stay away from Coleco's adaptation of Dragon's Lair, however. That arcade game owed its appeal to the animation effects made possible by its innovative videodisc system. Coleco's version is just a standard video game, and not a very good one. Coleco has announced an expansion system that will allow true videodisc games. If this system is marketable, Coleco plans to sell an improved version of the Dragon's Lair game.

If you're looking for adaptations of famous arcade games, you might want to try an unlikely source: Atari. Because of the vast popularity of its video game machines and computers, Atari has been able to buy the exclusive rights to sell home computer versions of many of the most famous video-arcade games. Until recently, Atari has jealously kept these titles for its own line of computers. Late in 1983, however, Atari launched a new division called AtariSoft to sell games for its competitors' machines from its vast library.

At the moment, Atari has issued only three titles for the Coleco: Centipede, Defender, and Galaxian. All are sold as cartridges that will fit in either the Adam or the ColecoVision game machine.

You cannot, however, buy the most famous game owned by Atari: Pac-Man. Even though AtariSoft adaptations of this best-selling game are available for other computers, Atari has so far refused to sell it for the Coleco. That is unfortunate, but there are many other games you can play.

Of the independent companies that have announced games for the Adam, Brøderbund Software is the most impressive. Of the Adam games that Brøderbund is preparing, Lode Runner is one of my favorites. In this game, you maneuver a man around a vertical maze, trying to pick up treasures without being caught by the three guards who are chasing you. Brøderbund has also announced an Adam version of its popular game called Choplifter!, in which you fly a helicopter behind enemy lines trying to rescue 64 hostages.

You should remember that you can buy more than just video games. Chess is one of the most interesting games you can play against the computer. Some computer chess programs can play a very respectable game.

Many people enjoy *adventure games.* Unlike a video game, an adventure usually does not involve fast shooting or quick action. Instead, it

places you in an imaginary situation, and asks you to find a treasure or solve a problem. By far the best of these is the Zork trilogy, by Infocom, in which you must explore an amusing underground empire.

You play Zork by typing short commands, such as "climb the stairs" or "look under the table." If the computer can understand you, it will do as you direct and then describe the results. You must often be very resourceful to find your way around an obstacle in the imaginary universe.

The joysticks that come with your computer will be good enough for most games. If you are a serious game player, you might want to invest in a fancier joystick, for greater comfort and better control. Coleco sells a set of *Super Action Controllers,* which are shaped so that you can hold them easily in one hand. For certain games, you might also consider an arcade-style *trackball,* which lets you control your players by rolling a ball in the direction you want to move. Coleco's trackball is sold under the name *Roller Controller.*

SERIOUS PROGRAMS

Coleco and other companies sell many useful programs that can help you in serious work as well. Education, home finance, and record keeping are only a few of the uses to which you can put your computer.

Many people buy educational programs for their children. A computer program can give a child individual attention and colorful, animated illustrations of concepts. It can check his work each step of the way, correcting mistakes as he makes them. Some people feel that a child who becomes familar with a computer will be more comfortable with a machine that can be expected to play a large part in his future.

Of the educational programs being developed for the Adam, my favorites are those made by Spinnaker Software. That company's Facemaker, designed for preschoolers, is one of the most entertaining programs available, yet it is effective in teaching basic skills of memory, recognition, and creativity. With this program, your child can create a face on the screen by choosing from a selection of hairstyles, eyebrows, noses, and mouths. He can also play a memory game that challenges him to recall a lengthening series of facial expressions. Spinnaker sells several other fine educational programs.

If you want to introduce your children to computer programming, you might consider Coleco's SmartLogo. This is a simplified programming language that lets a child explore the computer by creating graphic designs on the screen. It is sold on cassette tape or diskette.

Complex financial calculations haunt many homeowners. A number of computer programs are available that can simplify such tasks as calculating loan payments, making a budget, or keeping household records. You may be amazed at how easily a computer program can help you solve your financial problems.

One of the most useful financial programs is the *electronic spreadsheet.* This program gives you a large free-form table, with many rows and columns. You type numbers and labels into the cells of this table in any way you wish. You can then use the resources of your computer to add figures or perform complex calculations. This program is useful for anyone who needs to keep track of charts of numbers. Coleco has announced two different spreadsheet programs for the Adam: its own SmartSheet, and a licensed version of the well-known SuperCalc 2 program. To use SuperCalc, you must have the CP/M operating system expansion.

Many other programs are being developed that will let you do other things with your Adam. Music programs, such as Electronic Arts' Music Construction Set, will let you play melodies through your computer's sound system. Painting programs will let you draw colorful pictures on the screen. Keep your eyes open: you will find many programs that will interest you.

HOW TO USE SOFTWARE

There's no real trick to using most preprogrammed software. You just start the computer, give some simple commands, and let the program run. When the program needs some information from you, it will usually ask you for it with a simple message on the screen. The best programs are organized so that they actually help you organize your thinking, and give you help when you need it.

With a cartridge, you just plug the program into the computer and turn it on. The cartridge plugs in to the top of the computer, in the large hole that is covered by a gray plastic door. Make sure you turn the computer off before you insert or remove the cartridge: you can damage the program if you leave the power on. To start the game,

simply pull the RESET CARTRIDGE switch to the right of the cartridge slot.

With cassettes or diskettes, you must load the program into the computer's memory before you can use it. With most cassette programs, you simply turn on the computer, place the cassette in the tape drive, and pull the RESET COMPUTER switch, at the left of the cartridge slot. The computer will then automatically load the program. The procedure is similar for a diskette. Most software packages include detailed instructions on how to load and use the program. For further information about cassettes and diskettes, read Chapter 4.

Once you have the program loaded, the going is easy. The better programs give you full directions right on the screen. You can usually read the instruction manual for further explanation.

A LOOK AHEAD

This chapter has shown you how to use programs written and sold by other people. In it, you have seen some of the things you can do with your Adam with no experience whatsoever. By using preprogrammed software, you can learn about your machine and start it working for you immediately.

This discussion has ignored one important program: the word processor that comes built into your Adam. This is probably as useful as any program you can buy for your Adam, and very easy to learn. Since it forms such a central part of the Adam system, the word processor has been given its own section in this book.

If you want to explore your computer more completely, you may want to learn to write your own programs. Section Three of this book concerns the Adam's BASIC programming language. Programming will put you in direct control of your machine, so that you don't need to restrict yourself to someone else's routines.

Whether you choose to explore the word processor or the BASIC programming language, you will be taking many more steps with your computer. If you work slowly through all the examples in this book, you will acquire a good, firm knowledge of your Adam.

SECTION 2

THE WORD PROCESSOR

Chapter 3
Easy Word Processing

In Section One, you learned how to set up your Adam and how to buy programs that will start it working for you. If you have played Buck Rogers: Planet of Zoom or any of the other Adam games, you have already gotten a start.

This chapter is devoted to the program that is built into your Adam: the SmartWriter word processor. With this program, you can use your computer as an intelligent typewriter to type error-free letters and reports.

Because of space limitations, this chapter can offer only a brief introduction to word processing. If you want more detailed information, you might want to read *Word Processing with Your Coleco Adam,* by Carole Alden (SYBEX, 1984). Her excellent introduction to Adam word processing is a companion volume to this guide.

WHAT IS A WORD PROCESSOR?

Word processing is one of the most useful tasks you can perform with a home computer, and one of the most interesting. If you often write letters or type documents, a word processor could save you lots of time. You can revise a text without retyping it, or send personalized form letters that all contain the same text but have different names and addresses at the opening.

What is a word processor? It is like an electric typewriter, except that you first store your text electronically in the computer's memory, instead of typing it directly on paper. You can then review your text on the television screen. As you look it over, you can insert, delete, or

rearrange words—even change whole paragraphs. Once you are satisfied with your text, you type a command that asks the printer to type the entire text on paper. If you later want to change a text that you have stored, you can call it back, revise it on the screen, and print it again. You should never have to retype anything.

A true word processor, such as the one in your Adam, has a feature called *word wrap*. When you reach the end of a line, the computer automatically pulls your next word down to the next line and keeps on going, without waiting for you to press the RETURN key. You can just keep typing as if you were writing one long line. The only time you need to use the RETURN key is when you want a new paragraph. Since you don't need to stop at the end of each line, you can type more quickly.

The real advantage of word wrap is that it lets you *edit* your text. Since the line breaks are not fixed, you can insert or delete words in the middle of a paragraph. When you do, the computer will simply rearrange the other words to fit between your left and right margins. If you realize you have forgotten a sentence, you can go back and insert it.

You can use a word processor for anything you would normally do on a typewriter. Letters, school reports, or even notes to yourself—all are easy with a word processor. The Adam's printer produces a text that looks like a typewritten page, so there's no reason anyone needs to know you typed it on your computer.

A word processor is especially useful for repetitive tasks like form letters. Once you've printed your text, it remains in the computer's memory. If you want to change a few words or add a different name, you just make your correction and print the revised text. You could print a whole stack of form letters in this way.

Even if you don't expect to use your Adam for word processing very much, you will learn a lot by playing with the program. Word processing is one of the easiest ways to learn about a computer, and since it comes with your Adam, you might as well take advantage of it.

ADAM'S ELECTRONIC TYPEWRITER

When you first turn on your Adam, your screen will show the words ADAM'S ELECTRONIC TYPEWRITER, as in Figure 1.4. This screen, is designed to resemble a sheet of paper in a typewriter.

For example, your letters will appear in the black bar across the bottom, which represents the typewriter's rubber roller, or *platen*.

Coleco includes this electronic typewriter feature to help you see the resemblance between a word processor and a regular typewriter. While you will usually want to work on the full word processor, you can start with a few experiments on this simulated typewriter.

Put some paper in the printer, then type some letters on the keyboard. The letters appear immediately on the screen, on the black roller at the bottom. Then, about a half second after you press each key, the printer will type the letter on the paper, just as if it were a typewriter. The half-second delay comes from the time it takes your computer to process the letter and send the appropriate signal to the printer.

Type as much as you want, just as you would on an electric typewriter. Your Adam can accept letters as fast as you can type. If you type quickly, the printer may lag a few letters behind, but the letters will not be lost. The computer stores them and sends them to the printer as quickly as it can print them. When you stop typing, the printer will catch up and print the rest of the letters as you typed them.

Figure 3.1 shows the Adam's keyboard. It looks very much like a standard typewriter keyboard, but with some special keys added. The letters are all in the same places, and only a few symbols have been moved. Around the edges, there are a number of special keys that help with certain word processing functions.

The most important of these special keys is the BACKSPACE, which lets you go back and change one of the letters on the screen.

Figure 3.1: *The Adam keyboard.*

Since the letter has already been printed on the page, you cannot correct it on paper; instead, the new letter will be typed on top of the old. (When you use the full word processor, you will be able to correct errors on the screen before they reach the printer.)

To type a capital letter, hold down one of the SHIFT keys while you press the letter. If you want all your letters to be capitalized, you can press the LOCK key at the bottom-left corner of the keyboard. This key works just like the shift lock on a typewriter, capitalizing a series of letters. To shift back into lowercase, press the LOCK key again. The SHIFT key does not release the capital LOCK as it would on a regular typewriter.

Auto-repeat is another useful feature. If you hold down any key for more than a second, it will repeat. You could, if you wanted, type a string of Zs across the screen by merely holding down the Z key. The repeating will stop when you release the key. The auto-repeat feature is particularly useful with the space bar, for moving to the middle of a line.

At the right of the line you are typing, you will see a black underline symbol, called the *cursor.* Whenever the computer is ready to accept letters from the keyboard, it uses the cursor to mark the spot where your next letter will be displayed. As you type, the cursor moves to the right, in front of the letters that are appearing.

You will also notice that the Adam cannot fit as many letters on each line of the screen as the printer can on the paper. When you reach the end of the first line on the screen, the computer will continue onto the second line, but the printer will still be on the first line. Although the lines are not displayed in their full width on the screen, they are still stored that way, internally.

The black ruler that runs across the top of the screen shows you your postion relative to the margins on the printer. The red bars near the left and right ends of the black bar represent the margins, beyond which you cannot type on the paper. Somewhere in between, a white vertical line indicates which column the printer is actually typing in. If you have just begun the second line on the screen, the white bar will be near the middle of the ruler, which means that the printer is roughly halfway across the page.

If you have your television's sound turned up, you will hear a bell as your letters reach the right margin on the printer. Like a typewriter, the Adam warns you when it is about to reach the end of the line. Press RETURN to start a new line, just as you would hit the carriage return on a typewriter. (On the Adam's full word processor, the computer's word wrap will be taking care of the line ends automatically.)

Don't press RETURN just yet—type a few more letters until the Adam will not accept any more. You have now reached the right margin of the page, and the computer has stopped you to keep you from typing off the edge of the paper.

Suppose you want to fit a few more letters on this line. On a typewriter, you would press the MARGIN RELEASE key. The Adam does not have a special key for the margin release; however, it has a key that serves the same purpose.

Look at the two blue boxes at the bottom-right corner of the screen. These are labels that correspond to the row of black keys marked with the Roman numerals I through VI on the top row of the typewriter keyboard. These *SmartKeys* are general function keys that perform various special tasks. The SmartKeys' meanings change depending on what you happen to be doing at the time. Look at the blue boxes on the screen to see what each SmartKey means at the given moment.

Right now, the screen shows two boxes, numbered V and VI. This means that SmartKeys I through IV have no function in this situation, but that you can use the other two keys. Don't worry about SmartKey V just yet: it changes the margin settings of the paper. Press the VI key to release the margins.

The effect is the same as if you pressed the MARGIN RELEASE on a typewriter. You can now continue and type a few more letters on the line. Like a typewriter, however, the Adam's printer will soon reach the right edge of the paper, and the computer will not accept any more letters, even if you again press the VI key. You must then drop down to the next line to continue.

Press the RETURN key. The printer will advance the paper one line and return to the left margin to begin the new line. On the screen, a black triangle representing the RETURN will appear at the end of the line, and your text will shift upward so that the black roller is once again empty. You can now type another line into the computer, just as you typed the first. Press RETURN at the end of each line.

That's all there is to Adam's Electronic Typewriter. Play around with it for a while. Everything will work the way you'd expect an electric typewriter to work. Don't worry if you can't figure out the

SmartKey options just yet. They will become clear as you learn about the word processor.

THE WORD PROCESSOR

So far, you have been using the Adam just like a typewriter. This is a good way to become familar with it, but you'll quickly want to move on to the full features of the word processor. Once you get comfortable with the word processor, you will want to use it for everything.

Press the ESCAPE/WP key in the upper-left corner of your Adam's keyboard. The computer will switch from the simulated typewriter mode to the full word processor. Once you have switched, you cannot return to the typewriter without turning the computer off.

When you press the ESCAPE/WP key, the screen will change slightly, so that it will have the same form as Figure 3.2. Anything you may have typed in the typewriter mode will remain on the screen as the beginning of your word processor text. The blue boxes at the bottom of the screen will have changed to show the new SmartKey options available in the Adam word processor.

There is another change in the screen as well. Along the left side, a vertical ruler appears with the number 11 at the top. Just as the horizontal ruler line at the top of the screen lets you see your position between the left and right margins, the white marker on this vertical line shows you how far you have moved down the page. The number 11 indicates that the computer is expecting you to use standard letter-sized paper, 11 inches long.

Now type some characters. This time, the printer remains quiet and doesn't type anything. The letters appear *only* on the television screen and not on the paper. Only later will you see the letters on paper, when you have the computer print the text all at once.

This is the main difference between a typewriter and a word processor. On a typewriter, your letters are printed as you type them. If you make a mistake or want to change something you have typed, you must correct the page with correction fluid. With a word processor, the mistake does not go directly to the paper. Instead, it is held in the computer's memory, where you can easily make corrections. Since nothing is printed until you are satisfied with the entire document, the correction leaves no trace.

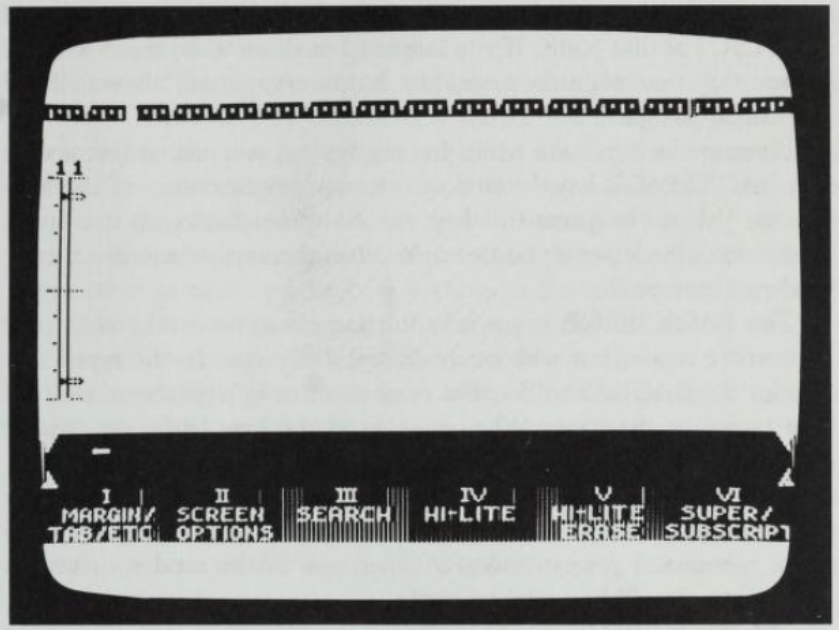


Figure 3.2: *The Adam word processor.*

Type out past the end of the line on the screen. The computer will break the line and start a new one when you reach the right edge of the screen. Keep typing until you reach the end of the second line. Notice that the word processor does not sound a bell when you reach the right margin. Instead, it automatically moves the line up off the black roller on the screen and continues the text on the next line down. The computer does not break a word in half, but starts the new line with a full word, so that you can read it clearly.

You can type as many words as you want. Try typing a full paragraph—use a passage from this book if you want. The computer will arrange the words neatly into clear lines on the screen. You never have to stop typing at the end of a line to hit the RETURN key.

On the word processor, the RETURN key is used only at the end of an entire *paragraph*. Within each paragraph, you let the word processor decide where to end each line: it will arrange the paragraph so that each line is roughly the same length. When you finish a paragraph, however, you will want the next one to start on a new line, so you hit the RETURN key to *force* the computer to end the line. A

small black triangle will appear to remind you that you pressed RETURN at that point. If you later add or delete text, the word processor may rearrange the lines, but the paragraph will always end at the black triangle.

If you make a mistake while you are typing, you can correct it with the BACKSPACE key, located at the upper-right corner of the keyboard. When you press this key, the computer backs up one space and deletes the letter typed there. You can then type the correct letter and continue on.

The BACKSPACE key works the same way as in the electronic typewriter mode, but with one essential difference. In the typewriter mode, the BACKSPACE could erase a letter on the screen but not one typed on the page. When you typed the new letter, the printer had to type it on top of the mistake. With the word processor, however, the BACKSPACE erases the error on the screen, and the new letter completely replaces the old one in the computer's memory. No trace remains of your mistake, so when you finally send your text to the printer, it will be typed correctly.

The BACKSPACE is the simplest of the many keys you can use to make changes in your text. In the next few pages, you will learn about other word processor functions that let you make more substantial changes in your text. The basic idea, however, remains the same: you correct your text on the screen before you have the printer type it, so that your text is perfect by the time the words appear on paper.

CONTROLLING THE CURSOR

You have seen that the letters you type are displayed on your television's screen. The letters appear from left to right, and the cursor, a flashing underline symbol, slides to the right in front of them, showing your position on the screen. However, by pressing certain special keys, you can move the cursor anywhere you choose in your text. Moving the cursor lets you correct letters that you have already typed.

The cursor always shows where the next letter you type will be displayed. So far, the cursor has always appeared at the end of the last word you typed. To move the cursor, you need to use the arrow keys, located just to the right of the keyboard. The left and right arrows move the cursor to the left or right along the line you are typing. The

up and down arrows let you move to other lines in the text, above and below the line you are typing.

Because of the way the Adam displays text on the screen, you must cope with one confusing feature. The cursor will always appear somewhere within the two lines displayed in the black roller region on the screen. You can use the up and down arrows to move the cursor to lines that fall outside this roller; however, instead of moving the cursor, the Adam moves the entire text up and down, as if it were moving a piece of paper up and down through the roller of a typewriter. So, when you want to move the cursor up to the line above, the cursor doesn't move, but the line rolls down to meet it.

In spite of this slight confusion, I will continue to speak of "moving the cursor." It is useful to think in these terms, since the cursor indicates where the next typed character will fall. The cursor will actually move in some cases, while in others it will wait for the text to move to it. In all cases, however, you are trying to position the cursor at some point in the text, and it's easier to think of moving the cursor, rather than the text.

Type some words. Now, press the left arrow key to move the cursor to the left. If you hold down the key, the cursor will move rapidly. You can move as far as you want along the line on the black roller. If you are on the second line on the roller, the left arrow will move the cursor back to the beginning of the second row and then jump to the end of the first line. It will finally stop when it reaches the first letter on the roller— the left margin of the paper as it will eventually be printed. You can use the right arrow in a similar way to move the cursor back to the right.

You may have noticed a similarity between the left arrow and the BACKSPACE key. Both keys move the cursor back to the left across letters you have already typed. But the BACKSPACE key deletes the letters it passes over, while the arrow keys do not. They simply move the cursor past the letters to position it. With the arrow keys, the text remains unchanged; only the cursor's position changes. This point will soon become important, when we want to change letters in the middle of a line.

If you have typed several lines, you are probably on the last line of the black roller, but you can see several previous lines on the light-blue portion of the screen. If you now want to make some changes in those lines, use the up arrows to move those lines.

Press the up arrow key. The text will be redisplayed on the screen so that the cursor is located one line up from its previous postion. The computer has moved all the lines down one. The line that used to be just above the black roller is now shown on the roller's first line, and the cursor is flashing somewhere beneath it. By using the up arrow, you have moved the cursor back to the line above the one you were typing. To return to the line you were typing, press the down arrow. The screen display will shift, and the cursor will be back on the line you were typing.

You may think it strange that the up arrow moves the text down and the down arrow moves it up on the screen. Remember that sometimes the Adam moves the text to the cursor, rather than moving the cursor to the text. The arrow keys indicate the direction of the cursor's movements. When you press the up arrow, you are asking the computer to move the cursor *up* to the line just above. Rather than move the cursor up, the Adam moves the text down to meet the cursor. The effect is the same, however: the cursor is now on the line above the line where it was before.

The HOME key, located between the four arrow keys, is also useful for moving the cursor. When you press HOME, the computer will place the cursor at the beginning of the first line at the top of the screen. If you haven't typed enough lines to fill the screen, the computer will move the cursor to the beginning of the first line of your text. Although you could reach this same line by pressing the up arrow key several times, it is quicker to use the HOME key, especially when you want to move back quickly through a long document.

The Adam word processing manual claims that you can use the HOME key in combination with the other arrow keys to move the cursor quickly. For example, the manual says that you can move down through your text a full screen at a time by pressing the HOME key and the down arrow at the same time. Unfortunately, this feature works erratically on most Adams. Try it. If this feature works reliably on your machine, you may find it useful. Don't get frustrated, however, if you occasionally get odd results.

CHANGING LETTERS

Now that you can move the cursor, you can return to any letter in your text and change it. When you move the cursor, the computer will place the next letter you type in at this new place, rather than at the end of the line you were on.

If you move the cursor and then continue typing, the new letters will replace the old ones on the screen. The cursor will again move from left to right along the lines, so that you will be typing new words over the old, starting from wherever you've located the cursor.

Let's correct a mistake. First, use the down arrow key to return to the last line you typed. Press the down arrow once more to start a new line, and type the following words on this clean line:

This line contains a mestake

The word 'mistake' is misspelled here. To correct it, you could press the BACKSPACE key six times to erase the end of the word and then retype it correctly. It is easier, however, to use the arrow keys, so that you have to retype only one letter. Press the left arrow key six times so that the cursor is under the *e*. Then type an *i*. The *i* replaces the *e* and corrects the misspelled word. You do not need to retype the rest of the word, since it is still stored in the computer's memory. To return to the end of the word 'mistake,' press and hold the right arrow key. Do not—as you would on a regular typewriter—use the space bar; it will replace the letters on the screen with blank spaces.

You have already used the BACKSPACE key to erase mistakes right after you typed them. You can also use this key to remove extra letters from the middle of a line without leaving a space. To see how this works, press RETURN to get to a new line, then type another misspelled line:

This line contains a misstake

Here you must remove the extra *s* from the word. Use the left arrow key to move the cursor under the second *s* in 'misstake'. Then press the BACKSPACE key. The extra *s* will disappear and the 'take' will move over to fill in the space. Use the right arrow key to pass over the corrected word.

WORD PROCESSING COMMANDS

With just the arrow keys and the BACKSPACE, you can move about your text, type over letters, and delete extra characters. The Adam's word processor lets you do much more, however. It lets you

insert text, delete whole blocks of text, and move and copy blocks to another point in the document. These useful functions are the real strength of a word processor.

The Adam has a group of six special keys for these functions, located just to the right of the BACKSPACE. Three of the six have very similar functions, so I will treat them now: INSERT, DELETE, and MOVE/COPY. The other three will come later.

The simplest is the INSERT command, which lets you add any number of words and letters anywhere in your text. The inserted letters are placed between letters you have already typed.

To see how the INSERT command works, let's try another example. Use the down arrow to move to a blank line at the bottom of your text. Then type

```
This line contains a mstake
```

Here, the *i* is missing from the word 'mistake'. If you move the cursor to the left and type the *i* at the proper place, it will appear on the screen but will replace a letter that you want. You would have to continue typing over the word until you reached the end of the line.

Instead, you can use the INSERT key. Move the cursor so that it is beneath the *s* in 'mstake'—always position the cursor to the right of the space where you want to add your letters. Then press the INSERT key. The screen will change and the letters to the right of the cursor will disappear, but they haven't been erased. You will get them back in a moment.

Now look at the message in the yellow box at the bottom of the screen:

```
INSERT
TYPE TEXT
```

This message indicates that you can now type the letter *i* without having it replace the *s* in the word.

When you pressed the INSERT key, the blue boxes at the bottom of the screen also changed to show new meanings for SmartKeys I through VI. In this situation, you won't need to worry about SmartKeys I through V: the first three have no function, and IV and V perform specialized tasks. SmartKey VI (DONE), however, is essential for telling the computer that you've completed your insertion (the letter *i*). The rest of your text will then be restored, and the regular SmartKey labels will return. Since you inserted the *i* before the *s*, rather than typing over it, the word will now be spelled correctly: 'mistake'.

You can insert text of any length. If your insertion runs past the end of a line, the word processor will start a new line just as it always does. When you press DONE to restore your full text, the computer will rearrange the words that follow so that they fit into a neat paragraph.

The ability to insert text is one of the great strengths of a word processor. Since you can always go back and insert a word or a sentence, you don't have to retype your text when you forget something. Some people intentionally leave whole sections out of their documents, then go back later to fill in the gaps.

This INSERT command is typical of the Adam's word processing functions. You first press the command key to show what you want to do. You then look at the SmartKey labels at the bottom of the screen for further directions and options. With INSERT, the computer also displays a yellow box that reminds you what to do next. In some cases, there will be no yellow box, but the SmartKey options will tell you what is needed.

At times you will find yourself in the middle of a function that you don't want to complete. You might, for example, have pressed the INSERT key or one of the SmartKeys by mistake. You can almost always cancel a command by pressing the ESCAPE/WP key. This will restore the text to the way it was before you hit the command key, and return you to the standard SmartKey options.

The opposite of INSERT is the DELETE command, which erases a whole section of your text. Just as the BACKSPACE key allows you to delete a few isolated letters, the DELETE key lets you erase a much larger chunk, such as a full sentence, a paragraph, or even a group of paragraphs.

To use DELETE, you also need to use the HI-LITE SmartKeys. If you press SmartKey IV (HI-LITE), a red underline will appear under the letters on the screen as you run the cursor past them. This underline will not be printed in the final text—it is merely the way you mark the block of letters that you want to delete. To stop painting the underline, press IV again (it now means HI-LITE OFF). If you go too far and highlight letters that you don't want to delete, press V (HI-LITE ERASE), then run the cursor back across those letters. The red underlines will disappear as you move.

You can highlight your text either before or after you press the DELETE key. I'd suggest you do it after. First press the DELETE key. The computer will display a yellow box with the instructions

HIGHLIGHT TO DELETE

Move the cursor to the beginning of the block and press SmartKey IV (HI-LITE). Then use the right arrow key to run the cursor past the characters you want to delete. If you want to delete more than one line, press the down arrow key to move to the second line, and use the arrow keys again to run the cursor all the way across the second line. There is no way to highlight an entire line in one step. You must run the cursor all the way across it.

When you reach the end of the block you want to delete, press IV (HI-LITE OFF). Check the highlighting to make sure it's correct, then press SmartKey VI—FINAL DELETE. The block of text will disappear and the text will be rearranged to fit the margins.

What if you decide you want the block back? The Adam's word processing manual claims that you can restore the text by immediately pressing the special UNDO key. You can—sometimes. In many cases, though, this key does not work correctly: the deleted text will return, but a word or two may be missing, or some letters may be garbled. This error results from a defect in the Adam's design, and you can't change that. So, don't rely on the UNDO key to restore deleted text; always make sure you want to delete the block before you press SmartKey VI for FINAL DELETE.

You can also MOVE or COPY a block of text. The MOVE option allows you to mark a section of your text and move it to another place. You could, for example, transpose two sentences or move a line further down the page. With COPY, the marked block will appear in both the new location and in its original position.

To MOVE or COPY text, press the special function key MOVE/COPY. Then press SmartKey V for MOVE or VI for COPY. From this point on, the procedures for MOVE and COPY are almost identical.

You must now highlight the text, but you need to mark only the first and last letter in the block. Move the cursor to the beginning of the block that you want to move or copy and press SmartKey IV (HI-LITE FIRST) to underline the character. Then move the cursor down to the end of the block and press SmartKey V (HI-LITE LAST). Note that you do not need to run the cursor past every letter in the block, as you did with DELETE. You merely need to move to the last letter and highlight it; use the down arrow key to move directly to the end of a long block.

After you have highlighted the first and last characters in the block, the computer will ask you to place the cursor at the spot where you want to move or copy the text. With MOVE, simply press SmartKey VI (DONE) once you have positioned the cursor. The computer will rearrange the text so that the block appears in its new place.

With COPY, the procedure is slightly different, because the Adam lets you copy a block several times. Position the cursor to show the location of the first copy, then press SmartKey V. The marked text will now appear twice in your document: in its original location and in the new one. If you want to make another copy, move the cursor to another point and press V again. When you're finished making copies, press SmartKey VI (DONE) to return to the standard word processing options.

The block commands DELETE, MOVE, and COPY are very useful, but they have a few limitations. You cannot use any of the three to change a block longer than a single screen. Sometimes also, the MOVE and COPY commands don't work correctly when you try to move a word or group of letters to another place in the same line: you must MOVE and COPY text from one line to another. Make do with these restrictions.

USING THE PRINTER

If you have been typing the examples in this chapter, you now have a fair amount of text on the screen, but none of it has appeared on paper: the word processor gains its power by delaying the actual printing until you have completed revising your text. Let's print out what you've got.

Put a blank piece of paper in the printer roller, or use the same sheet you used with the electronic-typewriter option.

Now press the PRINT key, located between INSERT and DELETE in the group of command keys. The SmartKey labels will give you a variety of options. You can use SmartKey IV if you want to print only what's on the screen, or use SmartKey III if you want

just the highlighted blocks. In most cases, however, you'll want to press SmartKey V to print the entire document: PRINT WKSPACE. Then press V again to start the printer.

The printer will type your entire document, line by line. Since you haven't asked for a particular format, the text is typed single spaced, with one-inch margins on either side. If your text is a page or less, the printer will type the entire text on the one sheet, then stop. You can then pull the paper out of the printer roller.

If your text is longer than one page, the printer will stop when it reaches the bottom. It does this so that you can pull out the paper and put in another sheet. Press SmartKey V (PRINT) once you have this second sheet in position. The printer will then continue from the point where the first page ends.

If you often print documents longer than one page, you might want to buy a *tractor feed* mechanism for your printer. Coleco sells one that snaps directly into the slots on the top of the Adam's printer. The tractor feed allows you to run a stack of continuous *fan-fold paper* through the printer. The feed mechanism guides the paper by pulling on sprocket holes located on strips along each side of the page. When the printing is finished, you can tear off the perforated strips and separate the sheets of paper. With a tractor feed, you won't have to position each sheet individually in the printer.

You can control many aspects of the document's final appearance: the left and right margins, the blank space at the top or bottom of the page, and the vertical line spacing. You can also change the size of the paper from 8½ × 11-inch *letter-size* to the longer *legal-size* paper.

To make such changes, press SmartKey I—MARGIN/TAB/ETC. The SmartKey labels in the blue boxes will change, giving you a variety of options, including SIZE OF PAPER (I), HORIZ. and VERT. MARGIN (II and III), LINE SPACING (V), and END OF PAGE (VI). Experiment with some of these. Most options are self-explanatory.

On the printer itself, you can change the style of the type by replacing the *daisy wheel,* the plastic disk that spins to strike the letters. Coleco sells several different daisy wheels for its printer.

Printing the text does not erase it from the computer's memory. It remains stored and displayed just as it was before you printed it. You can therefore go back and make changes if you don't like the printed product, and you can print the text as many times as you wish.

As you revise your document, you should always be thinking about the final printed text. The Adam word processor helps you visualize this with a feature called *on-screen formatting:* as you work, the program adjusts the display to reflect your changes. If you insert or delete a word, the display will show you what the text will look like when you print it out.

You have seen one major difference between the screen and the final printing. On the screen, the lines are limited to 36 characters. On the printer, the lines are longer—perhaps as many as 80 characters if you fill the full width of a page. The Adam therefore splits every line roughly in half on the screen, so that you can see the entire text.

The Adam also has another mode that lets you see the lines exactly as they are stored. This mode is called the *moving-window format,* because it lets you look at only part of each line through a *window* into the full-width text. Using the arrow keys, you can move the window right or left to see other sections of each line. The Adam limits you to part of each line, because it cannot show more than 36 characters on each screen line.

To get into the moving-window mode, press SmartKey II for SCREEN OPTIONS. Then press VI for MOVING WINDOW. Use the same procedure to switch back to the normal screen display.

The moving-window format is designed primarily to let you *preview* the entire text, so that you can catch gross formatting errors before you print. You can also use the moving-window display when you are designing a table with many columns. On the standard display, the columns will not line up properly; in the moving-window mode, all the columns are displayed exactly as they will appear on the page. The moving-window display also lets you DELETE, MOVE, and COPY larger screens of text.

ADVANCED WORD PROCESSING

You have now used most of the commands you'll need to compose and print your text. The Adam also has many other features that let you do more specialized tasks.

One important command is CLEAR, which lets you erase all or part of your document. You might want to use this command now to clear your screen of all the examples you have typed in this chapter. Press the CLEAR key, located just above the DELETE. The Adam

will give you two SmartKey choices: V for CLEAR SCREEN, and VI for CLEAR WK-SPACE. If you want to clear the entire workspace, press VI and the computer will ask:

CLEAR WORK SPACE: ARE YOU SURE?

Press VI (FINAL CLEAR) to finish the job.

The Coleco manual says that you can restore your document after a CLEAR command by pressing the UNDO key. Not necessarily. As noted earlier, the Adam's UNDO key gives unpredictable results. If your text is longer than a few paragraphs, UNDO will bring back only part of your text. The rest will be lost or replaced by gibberish. Don't use UNDO.

SEARCH is another useful feature. You can ask the Adam to scan through your entire document and look for every instance of a certain word or phrase. First use the HOME and up arrow keys to move the cursor to the beginning of the text you want to search. Then press SmartKey III for SEARCH, and tell the computer what string of characters to look for. The computer will then scan through the text and stop each time it finds the string of letters. You can also ask it to replace the string with another sequence of letters, in that one place or throughout your document.

You can also use the SEARCH feature to move to the end of a long document. The Adam has no command that takes you directly to the end of your text. If you are typing a short letter, you can simply hold the down arrow key and wait until you reach the end. With a long document, however, that procedure could take a long time. To get to the end quickly, tell the computer to SEARCH for a string of letters that you know is not in the text; you can use a nonsense string like 'qxz.' The computer will scan all the way through and stop at the end of the document.

Experiment with the various SmartKey commands. Most are clear enough that you can figure them out without instructions. You will discover that you can type *subscripts* and *superscripts,* (letters below and above the line), change the colors on the screen, or turn off the sound effects that imitate a typewriter's key clicks.

For more information, you will want to do some additional reading. I cannot recommend Coleco's word processing manual. It is unclear, poorly organized, and sometimes misleading. While it describes all the features of the word processor, the manual does not explain the purpose of the program. It also describes several features that do not work correctly. You may become frustrated if you rely on the Coleco manual.

If you want a fuller description of the Adam word processing system, look at *Word Processing With Your Coleco Adam,* by Carole Alden. This excellent book is very readable and follows a clear, well-organized outline.

There may come a time when you outgrow the Adam word processor. Although the SmartKey labels are quite easy to use, the unit's overall functions are limiting if you frequently write long documents. The most serious restrictions on the Adam's built-in word processor are speed and size. The commands work fast enough if your text runs only a page or two, but the program starts to slow down on longer documents. Also, since it is hard to move directly to the beginning or end of the document,it can be difficult to work with long documents.

Also, several features simply don't work the way they're supposed to. In this chapter, you have already learned to avoid certain pitfalls of the UNDO, DELETE, and MOVE/COPY commands. Other commands work inconsistently, too, and you may become frustrated if you are expecting reliable responses. Read the book by Alden to find out more about these pitfalls and to learn ways to work around them.

At some point, you may want to buy a more sophisticated word processing program than the Adam's built-in system. As this book was being written, no other word processors were available for the Adam, but one was in preparation: OmniWriter, by HesWare. I have seen that program on other computers and was quite impressed by its flexibility. If the Adam version turns out well, it might be an appropriate alternative for owners who outgrow the built-in system. Other word processing programs will probably become available too.

SUMMARY

The Adam word processor lets you correct typing mistakes before the ink hits the paper. Instead of typing directly on the page, you first store your words in the computer's memory, where you can easily make changes.

The four arrow keys are essential to the word processor. They let you move the cursor around the text, so that you can go back and

make changes. Without any special commands, you can type over parts of your text, or delete letters with the BACKSPACE key.

You can also edit your text in more complex ways. You can INSERT letters in the middle of a line you have typed. You can DELETE large blocks of text, or MOVE and COPY them from one place to another.

Once you are satisfied with your text, you tell the Adam to PRINT it. The printer will automatically type the entire document, stopping only to let you change paper at the end of each page.

In this chapter, I have skipped over one important command: STORE/GET. This key allows you to save a document on a cassette tape, so that you can turn the computer off without losing your work. To use this command, you must work with the Adam's cassette drive, which is the subject of the next chapter.

Chapter 4

The Cassette Drive

Your Adam has a cassette tape drive built into its front panel. In Chapter 2, you have seen how to use the drive to load commercial programs, such as the Buck Rogers: Planet of Zoom game. In this chapter, you will learn how to use it for another important purpose: permanent storage.

When you type a document into your computer's memory, it remains stored until you give a CLEAR command or turn off the power. That is fine if you just want to type a letter, print it, and go on to something else. Many people, however, like to store their writing for future use. With the cassette drive, you can save a document, turn the computer off, and load the text back in when you want to use it again. In this way, you can keep a text on hand and revise it later on.

The cassette drive can be used for storage with either the word processor or the BASIC programming language. Since this chapter is part of the word processing section, the examples involve the word processor. The same basic procedures, however, apply to storing BASIC programs, as described on pages 109-14, at the end of Chapter 7.

No matter how you plan to use cassettes, you should read the next few pages, which describe the tape drive and some precautions for handling the cassettes. These precautions are very important and they apply to all forms of cassette storage—not just to the word processor.

WHAT IS THE CASSETTE DRIVE?

From the outside, the Adam's cassette drive looks just like a stereo tape deck. It has a door you can open, two spindles for turning the tape, and a magnetic *head* for reading the recording surface. It accepts

tapes that are shaped exactly like those you might buy for a portable cassette recorder.

Inside, however, the cassette drive is very different. It can read the tapes at a much higher speed than a stereo recorder. It can automatically advance or rewind a tape, and can find its place in the middle of the tape. And while the drive looks like a regular cassette deck, it requires special cassettes available only from Coleco.

To distinguish its cassettes from the tapes used with stereo equipment and other computers, Coleco coined the name 'digital data pack.' I find this term cumbersome and confusing, so I won't use it in this book. A cassette is a cassette, and that's the name I'll use.

The Adam's cassette system is, in many ways, revolutionary. Other computers can use cassette tapes, too, but they read the tapes as a regular tape deck would: slowly and from start to finish. This severely limits their flexibility.

Designed specifically for computer data storage, the Adam's cassettes are superior to standard tapes. They spin much faster, so the computer can save or retrieve more information in a given time. The Adam's cassettes record more efficiently on their surface, so they can hold more information than other tapes.

Flexibility is the greatest advantage of the Adam's cassettes. With a standard tape, the drive must start at the beginning and read straight through to the end. The Adam's cassette drive, by contrast, can go immediately to any spot you choose and find the information that you want to retrieve. You can save many different files on one cassette and quickly find the one you are looking for. This flexibility allows you to do many things that would be impractical on another cassette drive.

The special cassettes are available only from Coleco. The tapes are specially *formatted* with magnetic markings so that the drive can find its place on the tape. Other kinds of cassettes will not work with the Adam; in fact, they won't even fit in the recorder.

TAKING CARE OF CASSETTES

Coleco's revolutionary cassette system has its drawbacks, as well. Because of the high speed at which the tapes spin, the unit is far more temperamental than a stereo recorder. Even a minor flaw can make a tape unreadable, and you may wind up losing expensive programs or valuable data.

Take care of your tapes! You must load a document stored on a cassette every time you want to use it. If you damage a cassette, you may permanently destroy its contents. To avoid disaster, you must follow a few basic rules.

Because the information on a cassette is recorded magnetically, the tape can be erased by any strong magnet: the stored information would be lost, and the tape may become unreadable, since the computer will not be able to find its place on the tape.

There are many sources of magnetism in your house. One is probably sitting right next to your Adam: your television set. Even closer is your Adam's printer, which produces quite a strong magnetic field while it is running. Your telephone produces magnetism when it rings, and your stereo speakers have magnets in them.

Never set your tapes on top of any piece of equipment! It's very tempting to set a tape down on the nearest object. If that object happens to be your Adam's printer, your tape could be erased in a matter of seconds. Set a shoebox aside for all your tapes, and store them there any time you take them out of your machine.

Set up your system so that the cassette drive is at least two feet away from your television and printer. Magnetic interference can disturb the drive as it is reading or recording.

Remove your cassettes from the drive before you turn the computer on or off. Whenever you flip the on-off switch, there is a power surge, which can erase information from the tape. Although Coleco's program tapes were originally designed to load automatically when you turn the power on, you should never try that. Always turn the computer on first, then insert the tape and load the program with the RESET COMPUTER switch.

Always wait until the drive stops before trying to EJECT a cassette. The tape will be physically damaged if you try to remove it while it is spinning. No matter what it is doing with the cassette, the Adam always stops spinning it within a few minutes. Be patient, then remove the tape when it stops.

Keep dirt and dust away from your cassettes. Always store the tapes in their protective boxes, where they will not be contaminated. Store the tapes at normal room temperatures—away from extreme heat and cold.

Do not touch the surface of the tape when you remove it from the drive. Unlike regular cassettes, Coleco's cassettes do not have a plastic leader on the end of the tape. The bit of tape that is exposed through the cassettes's casing does contain information, and you can damage it permanently by touching it.

What can you do if you damage a tape? Usually nothing. If the tape is physically damaged, it will be unreliable even if you can succeed in reloading it. In some cases, though, you may be able to *reinitialize* a tape so that you can use it again even though you cannot recover the stored information. To do this, you will need to use a command in the BASIC programming language, described on pages 113-14 of Chapter 7. This procedure will, however, erase any files you have stored on the tape.

I hope you won't feel intimidated by all these "don'ts." If you treat your cassettes well, they will give you faithful service. You may sometimes get away with carelessness, but it's best to play it safe.

STORING A WORD PROCESSOR DOCUMENT

In Chapter 3, you composed and edited texts on the word processor. You typed your document, made changes, and printed the final version on the Adam's printer.

However, you were unable to store your document permanently. Once you turned off the computer, the document was cleared from the memory, and you could not reuse it without retyping it.

The word processor's STORE/GET function key lets you save a document on a cassette. Once you have saved the document, you can reload it any time you want to revise or reprint it.

Type some text into the word processor and press the STORE/GET key. A variety of new SmartKey options will appear on the screen. To save the text you have typed, use SmartKey V—STORE WK-SPACE. Other SmartKey options allow you to store just the screen you are using, or just the highlighted portion of your text. You will rarely need these options—just store the entire document.

Put a blank cassette in the drive, if you haven't already done so. One blank tape comes in the box with your Adam, and you can buy others from Coleco. If you forget to insert your tape, the Adam will ask you to do so.

When you press SmartKey V to store the workspace, the computer will ask:

```
SELECT DRIVE
```

If, like most people, you have only one tape drive, you will have only one choice: SmartKey III, for DRIVE A.

When you press this key, the computer will ask for something else:

```
FOR NEW FILE
TYPE FILE NAME
DRIVE A
```

A *file* is a block of information that you want to use all at once in the computer. A word processor document is normally a single file—you save it as a unit, then reload it all at once.

When you store a document on a cassette, you must always give it a *file name,* so that the computer can locate it when you want to retrieve it. The computer will keep track of each file's location on the tape, so that you can later ask for it by name.

Type a name for the file you want to store. The file name can contain numbers, but the first character must be a letter. Spaces and punctuation marks are not allowed. The file name can be up to ten characters long, but *no longer.* The computer may damage the tape if you use a name that is too long. Choose a name that describes the file's contents; otherwise, you are likely to forget where the information is stored.

Once you've finished typing, press SmartKey VI, which now reads STORE WK-SPACE. The cassette will spin and store your file. You cannot type other commands at the keyboard while the tape is turning. You cannot stop the tape either: you must simply wait until it stops spinning. The computer will then return to the normal SmartKey labels, and you can type again.

You may sometimes have trouble with this STORE procedure which is slightly more complicated than most other word processing functions. The computer will give you a message if you forget to insert a cassette, forget the file name, or haven't typed any text to store.

If, however, the tape starts spinning but the computer refuses to store the file, your problem lies with the cassette. Repeat the procedure. If the computer still cannot store the file, you may need to replace the cassette.

If the document is very important, you may want to make a *backup.* Record the file again on another cassette, and keep the two copies in different places. Then, if anything ever happens to your original, you can use the duplicate.

The Adam also keeps a backup of another sort. If you're revising a file that you have loaded from a tape, you can STORE it back on the tape without giving it a new name. The new version will take the place of the old, but the previous version will remain stored as part of a *backup directory,* which you can load if you can't retrieve the revised version.

RETRIEVING THE DOCUMENT

Time has passed. You have turned off your computer and cleared its memory. You now want to go back and reload the document you saved. To do this, you again start with the STORE/GET function key.

Press STORE/GET. As before, the computer will give you a choice of various SmartKey options for storing or retrieving a file. In this case, you want to GET a file, so you'll need SmartKey VI. When you press it, the computer will first ask you to select the drive. Press SmartKey III for DRIVE A.

At this point, the computer will display the message

ONE MOMENT –
GETTING DIRECTORY

Unless you've just been using the cassette, the computer will spin the tape for a few seconds. When it stops, it will display the *file directory,* an index of all the documents you have stored on the cassette. Figure 4.1 shows a typical directory: three documents have been stored under the names Test, Sample, and Demo.

Notice the small black triangle to the left of the first name in the file directory. Choose the file you want by moving this pointer with the four arrow keys. Then press SmartKey VI (GET FILE).

The cassette will now begin to spin again to load the file. When it stops, the computer will return to the usual word processor display, and the new file will be added to the end of any text now in the computer's memory. If, as is often the case, you want only the new document and nothing of what is on the screen, you should give a CLEAR command before pressing STORE/GET.

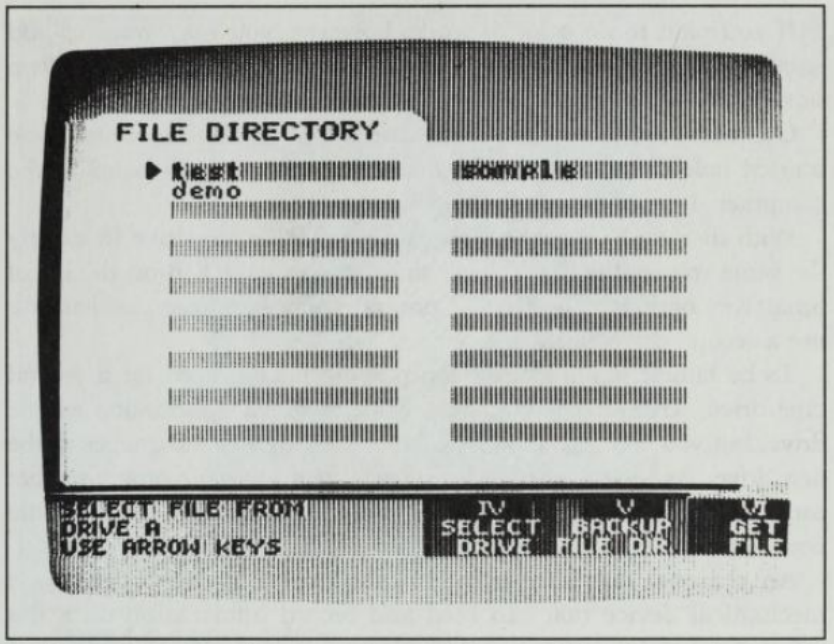


Figure 4.1: *The word processor shows a directory of all the files on the cassette.*

Sometimes you may want to see the directory of the cassette without loading a file. Follow the GET procedure to the point at which the directory is displayed, then press the ESCAPE/WP key. The word processor will return to the document you were editing, without loading a new one.

SmartKey V lets you use the cassette's backup directory, which contains the earlier versions of any files you have saved twice under the same name. Try this option if you ever have trouble loading a file. The backup will probably be an outdated version, but you may be able to load it and work with it. Files that you saved only once under a given name will not appear on the backup directory.

OTHER FORMS OF STORAGE

The Adam's built-in cassette recorder suffices for most people's needs. With a single drive, you can use the Adam for everything it was designed to do: word processing, BASIC programming, and prepackaged software.

If you plan to do a lot of work, however, you may want to add another storage device to your system. Coleco is preparing to offer two storage add-ons that will let you store more data with greater speed.

One option is to buy a second cassette drive, which mounts in the unused hole to the right of the first drive, on the front panel of the computer. It must be installed by a dealer.

With the word processor, you can use this second drive in exactly the same way as the first. It will show up as DRIVE B on the list of SmartKey options. The BASIC programming language also lets you use a second drive easily.

To be honest, I think few people would have a need for a second tape drive. True, it lets you store twice as much information as one drive, but you can get the same effect merely by switching tapes in the first drive. As time goes on, Coleco may begin to offer programs that use a second drive to simplify operations, but for now there is little need for it.

An optional *disk drive* is much more useful. A disk drive is a mechanical device that can read and record information on a flat magnetic surface. It acts somewhat like a record player, spinning a circular *diskette* at high speed and recording information in thin bands on its surface.

With a disk drive, your computer can read preprogrammed diskettes and can also store and reload diskettes that you program. A disk drive is faster, more flexible, and more reliable than a cassette drive. You can use preprogrammed software almost instantly, without having to wait for the cassette drive to read through the tape.

A diskette, shown in Figure 4.2, is a 5¼-inch vinyl plate covered with a magnetic recording surface. To protect it from damage, the magnetic disk is encased in a plastic envelope, with only a few small holes exposed. In operation, the drive spins the disk inside its envelope and records information on the surface as it passes the oval hole.

The great advantage of diskette storage is its reliability. While you must treat diskettes with care, they are more dependable than cassettes. If you frequently need to store important information on your computer, you should buy a disk drive rather than rely on the Adam's cassettes.

As this book was going to press, the disk drive was not yet available. Coleco's preliminary information indicates that the drive will mount on top of the main computer console. With the word processor, you will be

Figure 4.2: *A typical diskette.*

able to use the disk drive in much the same way as the built-in cassette drive. You cannot, however, use a disk drive with the standard SmartBASIC programming language. Coleco plans to sell a revised version, called SmartBASIC II, that will give access to diskettes.

SUMMARY

With the Adam's cassette drive, you can save word processor documents and load programs from cassette tapes. This means of permanent storage allows you to save your work so that you won't have to retype it every time you want to use it.

The Adam's high-speed cassette system is faster and more flexible than traditional forms of cassette storage. It does, however, require a few precautions. Above all, be careful in storing your cassette tapes so that you don't damage their magnetic recordings.

To save or retrieve a word processor document, use the STORE/GET function key. As you store the document, give it a file name, so that the computer can locate it on the directory when you want to reload it.

SECTION 3

BASIC PROGRAMMING

Chapter 5
Commands Of Your Own

In the preceding chapters, we have looked at programs designed by other people. With commercial software you can put your Adam to work for you, but only for what the program was written to do. With the word processor, you can easily do a specific task, but nothing else.

If you want to do something for which you can't buy a commercial program, you can write your own. It takes a little work, but once you have mastered a few simple commands, you can learn to make your computer do exactly what you ask.

With this chapter, you will begin to use your Adam's SmartBASIC programming language, which was included with your system on a cassette tape. BASIC stands for "Beginners' All-purpose Symbolic Instruction Code," if you can believe that. BASIC is easy to learn and simple to use; because of this, it has become the most popular language for small computers such as the Adam.

Of course, you don't have to write your own programs. Many people prefer to use preprogrammed software on their computers, rather than take the time to learn a computer language. However, programming is a good way to learn more about your computer. By giving your own commands, you can gain direct control over the machine and can make it do what you want. Even if you later decide to stick with preprogrammed software, you may want to know something about how the computer actually works.

In this chapter and the next, we will not be writing full programs, but merely single commands that tell the computer to do one thing at a time. In Chapter 7 we will begin combining commands to form entire programs.

At the beginning of this chapter, you will learn how to load the SmartBASIC tape and how to use the keyboard to type commands. For the most part, you will use the keyboard just as you did with the word processor, typing words onto the computer's display screen. The keyboard does, however, work a little differently in the BASIC mode, so you will need to learn some of its peculiarities.

At the same time, you will also learn how to control the words your computer displays on your television screen. In BASIC, the screen has two functions. First, it displays everything you type on the keyboard, so that you can see what you have written. Second, it displays the computer's responses to your commands.

This chapter introduces a simple command called PRINT, which lets you display a single line of text or numbers. As you learn more about this command, you will be able to position the text on the screen and clear unwanted clutter from the display.

Some of this may seem trivial. Why go through all this trouble just to have the computer say Hello? Why learn commands that let you clear the screen? Be patient. These are building blocks for the rest of the book. Before you can do anything more complicated, you need to know how to type commands into your computer and how to interpret its responses. Once you've passed this step, you can do many interesting things.

LOADING THE **BASIC** TAPE

Coleco, unlike most other home computer manufacturers, chose to provide the BASIC language on a cassette tape, rather than build it directly into the computer. This decision let Coleco build the word processor into the Adam, but added an inconvenience: you have to load the SmartBASIC tape every time you want to program the computer.

Take care of your BASIC tape! It is the only one you've got, and it's essential for any programs you may want to write or run. You cannot copy the tape, and you must buy another if you need to replace it. So be very, very careful.

Read the precautions in Chapter 4 concerning Adam cassette tapes. Several of these warnings are crucial. Don't put the BASIC cassette on top of the printer or leave the tape in the drive when you turn the machine on or off. Never touch the magnetic recording surface. And don't use the BASIC cassette to store any programs or word processing files; use blank cassettes for storage.

I recommend the following procedure for loading the BASIC tape into the computer's memory:

1. Turn on the machine without a tape in the cassette drive, just as if you were using the word processor. ADAM'S ELECTRONIC TYPEWRITER will appear on the screen.
2. Insert the SmartBASIC tape into the cassette drive, with the labeled side facing outward. Close the door on the drive.
3. Pull the RESET COMPUTER switch, located just to the left of the cartridge slot on the top of the computer. The tape drive will start to spin and the computer will load the BASIC language into its memory. This will take about a minute, after which the screen will look like Figure 5.1.
4. Remove the BASIC cassette from the drive and put it back in its storage box. Put it away in a safe place until you need to use it again. Do not leave the cassette in the drive while you are using the computer.

If you had problems with the procedure at any step, go back to the beginning and start over. If you were able to get to the word processor screen, but the BASIC cassette wouldn't load, there may be something wrong with the cassette. Try loading it again, then check with your dealer if it still won't work.

You will rarely need to reload the BASIC tape while using the computer. If you type a command or run a program that causes the machine to "lock up" and not respond to anything you type, you will have to put the BASIC cassette back in the drive and pull the RESET COMPUTER switch to reload it. You can also reload the cassette any time you feel hopelessly lost and don't know how to clear the machine to get back where you were. Reloading BASIC erases everything you had just stored in the computer's memory and returns the machine to its initial state.

THE **BASIC** KEYBOARD

To type commands or programs, you need to use the *BASIC editor.* This is the part of the BASIC language program that lets you type

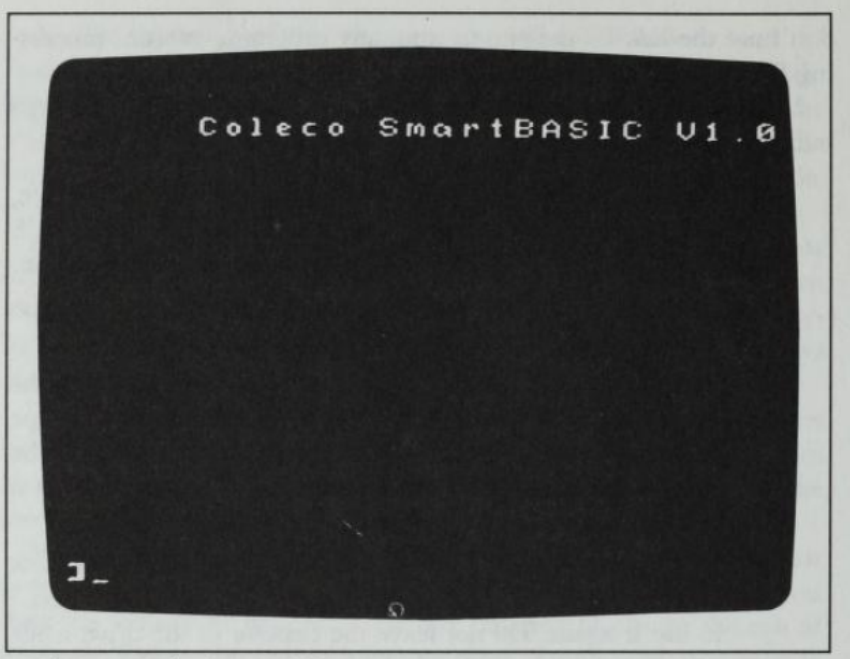


Figure 5.1: *The SmartBASIC start-up screen.*

lines on the screen and make changes in them. You will find you can use the keyboard and the screen in essentially the same way that you used the word processor. Some of the fancier word processing options are not available, but you can type most of your commands just as you would on a typewriter.

The BASIC screen editor is separate from the word processor because it is used for a different purpose. In BASIC you type words and commands that direct the computer's activity; with the word processor, you type English sentences that will be reproduced on the printer. Because most commands in BASIC are just one line long, you don't need all the powerful editing commands that you used with the word processor.

At the bottom of the screen, you will see a single bracket (]) and a flashing underline. The bracket is a *prompt,* a sign that the BASIC language is loaded in the computer's memory, ready to accept commands. As in the word processor, the cursor indicates where your next typed character will appear on the screen. The bracket always appears at the beginning of the line when the computer is expecting you to type a command.

Try typing some letters on the screen—just anything will do for now. You can type on the keyboard just as you did with the word processor. When you press a key, the letter will be displayed as lowercase, or as a capital if you hold down the SHIFT key. If you press the LOCK key, all the letters will be capitalized until you press the LOCK key a second time to disengage it.

As you type, you might notice a few of the peculiarities of the keyboard. First of all, there are keys for the numbers 0 and 1. If you are used to a standard typewriter, you may be accustomed to using the letters O and L for these digits. You can't do this when giving commands to the computer: you must give it numbers when it expects numbers.

If you have been using the word processor, you may have grown accustomed to using the SmartKeys I through VI and the special function keys such as DELETE, INSERT, and MOVE/COPY. Unfortunately, you cannot use these helpful keys with the BASIC editor. The computer will ignore these keys if you press them, or may even be confused by them.

You can still make some changes in the lines you are typing. You can use the BACKSPACE and the four arrow keys to move the cursor around the screen. In BASIC, the BACKSPACE key is identical to the left-arrow key.

For the moment, you will want to make corrections in a line you have just typed, so you will be relying on the left and right arrow keys. When you press the left arrow or the BACKSPACE, the cursor moves one space to the left. The letter remains on the screen, but it is erased from the computer's internal image of the typed line. If you then type a new character, it will replace the one under the cursor. If you want to restore any of the characters you backspaced over, you must run the cursor back across them by using the right-arrow key.

You can also use the up and down arrows to move the cursor to other lines on the screen. You might remember that the word processor moves lines down into the black "roller" at the bottom of the screen. In BASIC, you do not move the lines down the screen. Instead, you move the cursor up so that it is flashing on the line you want to change. You cannot return to a line that has disappeared off the top of the screen.

Although you can edit and reuse lines you have already typed by using the up and down arrows, this is a rather complicated procedure.

For the time being, just retype the lines you want to reuse. We'll come back to this subject in Chapter 7.

If you type past the right edge of the screen, the computer will continue to display your letters on the line below. This movement resembles the word processor's except that the BASIC editor does not worry if it splits a word in two. The computer, in fact, treats the two lines as one long line. You can type a single command of up to four full lines.

RETURN *AND THE ERROR MESSAGE*

If you're accustomed to an electric typewriter or the word processor, you might be tempted to press the RETURN key at the end of a line you have typed. In BASIC, the RETURN key has a special function: it tells the computer to take the line you have typed and read it as a command.

Try typing a line of text such as this:

```
This is not a command.
```

Then press the RETURN key. The computer will object with an error message, as shown in Figure 5.2.

What is happening here? When you press the RETURN key, the computer tries to read your line as a command. If it can understand the instructions, it will follow the command or store it in its memory. If it cannot understand your meaning, the computer assumes you have done something wrong, and prints an error message. Fortunately, errors are seldom disastrous. You can continue by just typing another command.

You might object that your line made perfect sense. It does to you and me, I agree, but not to the machine. Computers require that you do things in exactly the way they expect. Standard English sentences are much too complicated for the computer to understand: it is limited to more specific commands. Later in this chapter, you will learn some commands that are meaningful to the machine.

Don't be put off by error messages. They are merely the computer's way of saying it couldn't understand what you did. In this case, for instance, we typed something that made perfect sense to us, but not to the computer. The error was its fault, not ours.

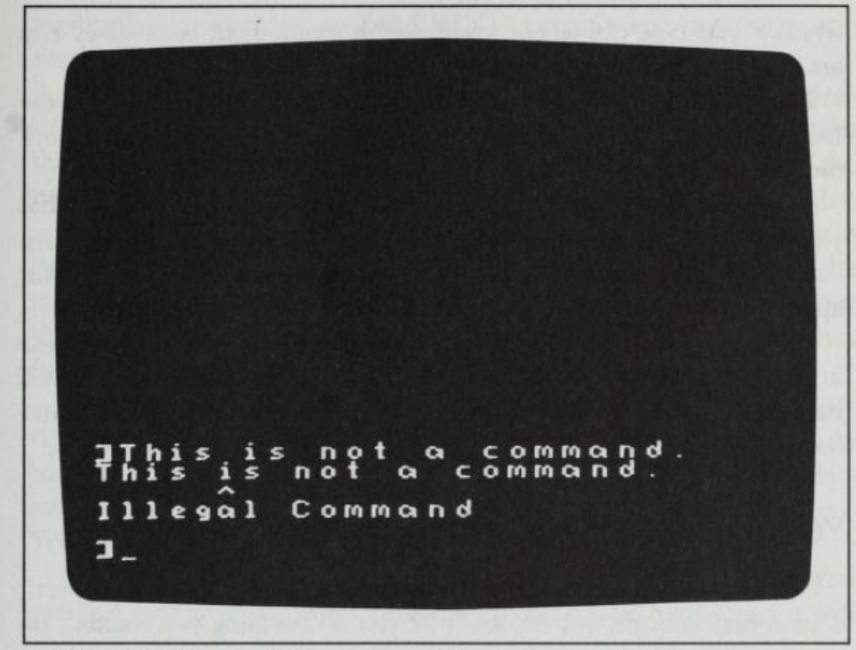


Figure 5.2: *You get an error message if you type a line that the computer can't understand as a command.*

Errors are part of the natural course of mastering your computer, and you shouldn't be ashamed if you get a few. Your computer can understand what you're saying only if you say it *exactly* in the form required. If you mistype even a comma, the computer will not understand you.

The computer will usually help you find your mistake by putting a small arrow under the first letter it couldn't understand. In Figure 5.2, for example, the arrow is beneath the *i* in 'is'. Your mistake will usually be to the left of this mark—in this case, the error started right at the beginning, with the word "This," since no part of the line was valid as a command.

As you type the commands from this book, make sure that you copy them exactly as they are printed, right down to the last quotation mark. If you get an error message for any line you type, check it letter by letter against what's printed here: you might have misspelled a word or lost a punctuation mark.

Most other computers insist that you capitalize all the letters of a BASIC command. The Adam's BASIC does not. It doesn't care

whether you type capitals or lowercase: it treats them both alike. You are therefore free to press the LOCK key or not, as you choose.

I suggest that you type everything in lowercase. You could use the LOCK key to type capitals, but then you will have to lock and unlock the keyboard in every line that has a number.

I use lowercase letters in most of the examples in this book, so that you can type the lines exactly as they appear. Some of the program listings in later chapters use capital letters to match messages that appear on the screen, but you can still type them in lowercase.

I do, however, use capitals for command words when they appear in the main text of this book; for example, we'll look at the PRINT statement in the next few pages. If you go on to read other computer books, you will find that most use all-capitals for command words.

YOUR FIRST COMMAND: **PRINT**

So far, you have merely typed some words and letters on the screen. This is easy enough, but it doesn't let you do anything you couldn't do more easily with the word processor. To control your computer, you have to learn commands that the computer can understand.

In this section, you will learn your first command: the PRINT statement. This lets you display a string of letters, numbers, and even the results of calculations on the screen. For right now, you will use this command only to have the computer retype words and phrases, but later, you will be able to apply it in many ways. By trying these simple experiments with PRINT, you will also learn what a command is and how it affects the computer.

To give a command, you type a line on the screen and press the RETURN key. The computer will then read the line you typed and try to follow its instructions. If it can't make sense of the line, the computer will give you an error message.

Type the following line, just as it is:

```
print "Hello"
```

Use the SHIFT key to type the capital H. When you've finished typing the line, press the RETURN key. The computer should display the word Hello, as shown in Figure 5.3. When you try this, you may have other words higher on the screen. If you get an error message when you press RETURN, you have probably made a mistake in typing. Make sure you spelled PRINT correctly and put the quotation marks before and after the word Hello.

You have typed your first command! The computer recognizes PRINT as a *keyword* with a special meaning. When the computer sees this word, it knows to search through the rest of the line and print the message enclosed in quotation marks. It finds the word Hello, so that is the message it displays. Note that the computer does not reproduce the quotation marks: they are merely a sign that the word Hello is the message to be printed. After displaying the message, the computer drops down one line and types the bracket symbol, to show it has finished and can accept another command.

The results may not be impressive, but you have made an important step with your command. You have asked the computer to do something, and you phrased your request in a way it could understand. It read your command and displayed the results on the screen. With this step, you have begun to control your computer.

You can try many variations on this simple PRINT statement. You can put anything you want between the quotation marks and it will

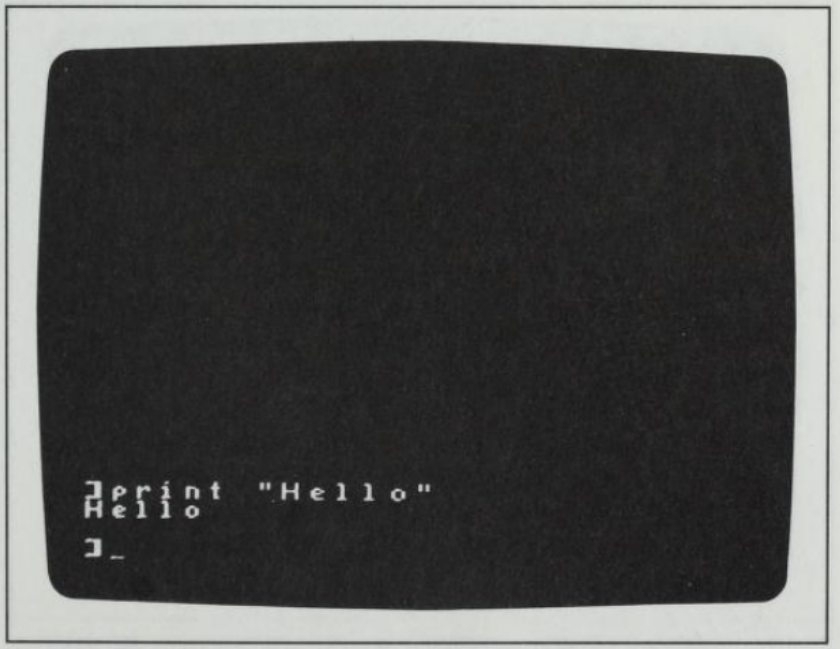


Figure 5.3: *How the computer should react to your first command.*

be printed exactly as you typed it. You can use lowercase and capital letters however you wish within the quotation marks: they will be reproduced as you type them. Also, any spaces you include between the words will be included in the message the computer prints. Figure 5.4 shows some of the things you can do.

The next-to-last PRINT statement in Figure 5.4 runs across more than one line. That is fine, as long as you type the entire command as a unit, without pressing RETURN. If you haven't finished your message when the cursor reaches the edge of the screen, just keep typing: the computer automatically goes on to the next line down. You can type commands up to four and one half lines long.

The last line of Figure 5.4 shows an important use of the PRINT command. If you do not type anything after the word 'print', the computer will print an extra blank line on the screen, in addition to the one that automatically appears above the bracket symbol.

You can also give more than one command at a time. Instead of pressing RETURN at the end of the first command, type a colon (:) and then your second command. You can add as many commands as

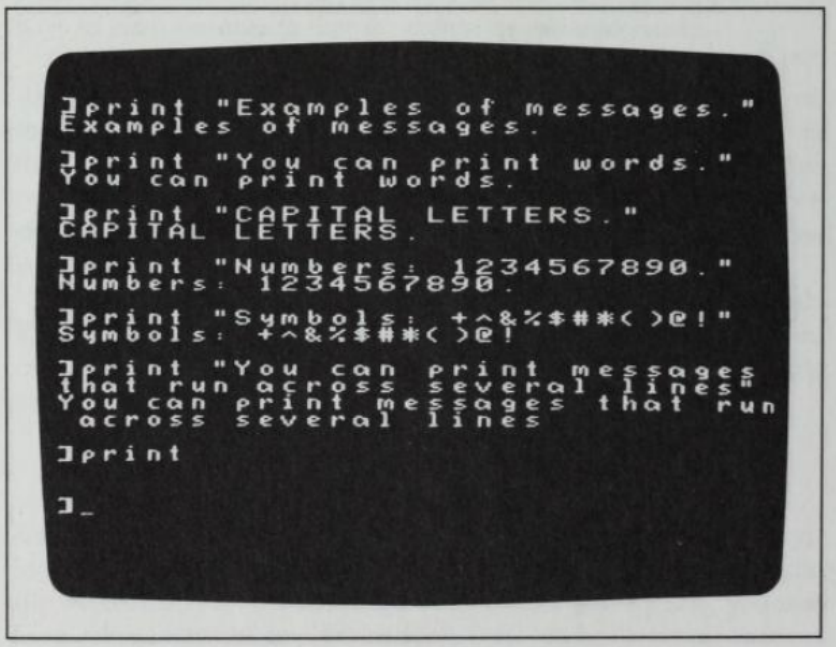


Figure 5.4: *Some examples of the PRINT statement.*

fit within the limit of four and a half screen lines. When you're done, press RETURN and all the commands will be carried out in order.

Figure 5.5 shows how you can use blank lines and multiple statements to improve the appearance of your screen display. The first three PRINT statements with no message tell the computer to put three blank lines on the screen. The fourth PRINT statement contains the first line of a message, which begins with eight blank spaces so that it will be centered on the screen. The fifth adds a blank line so that the message will be double-spaced, then another displays the second line of the message. Two final PRINT statements put some extra blank lines between the message and the bracket that appears below it.

This compound statement is more complicated than anything you'd usually want to do, but it shows how you can organize your screen display. By adding more blank PRINT statements and extra spaces to the messages, you could have the computer move several lines down and display the message in the center of the screen. However, you will learn an easier way to do this in the pages ahead.

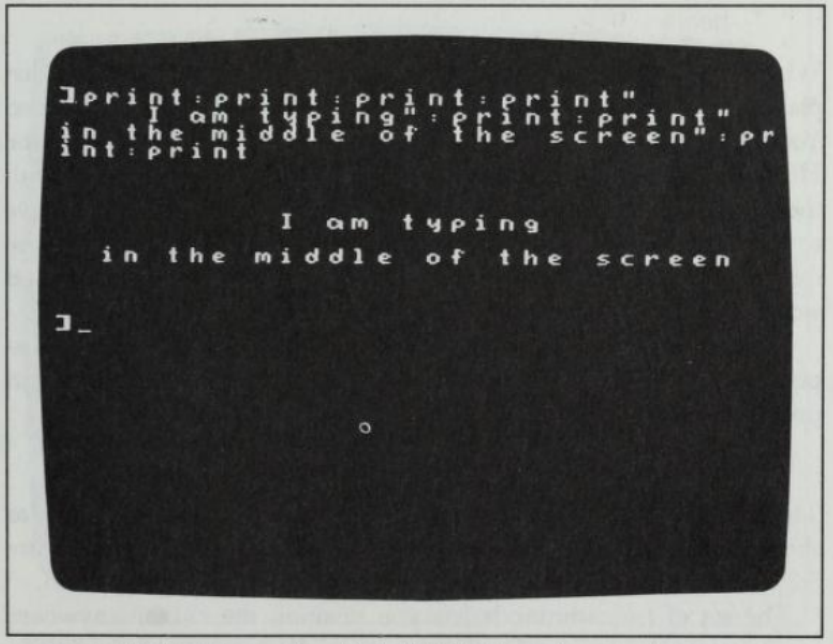


Figure 5.5: *Blank lines and multiple PRINT statements can make your screen more readable.*

PRINT is one of the most important commands because you will always need to display results on the screen. With the PRINT statement, you can do this in any way you wish.

CONTROLLING THE SCREEN

It is quite difficult to produce a readable screen by using the PRINT statement alone. A simple PRINT statement will display your message on the line immediately below your command. For clarity you can insert space between lines by using blank PRINT statements, as in Figure 5.5, but this leads to complicated commands.

Your Adam gives you several other commands that let you clear the screen and position the cursor where you want it. By combining these other commands with PRINT statements, you can create a more readable display.

The HOME command lets you clear the screen. Anytime you find the screen too cluttered, you can type

```
home
```

When you press RETURN, the computer will erase everything displayed on the screen, then place the cursor in the upper-left corner. You can then type or print messages on a clear slate. Note that the HOME command is different from the HOME key that you use with the word processor. If you press the HOME key, the cursor will move to the upper-left corner of the screen, but the display will not be cleared. To erase the screen, you must type HOME as a command word and press RETURN.

You can use the HOME command with a PRINT statement to display a message against a blank screen. Type the following line and press RETURN:

```
home: print "Hello"
```

The computer will clear the screen, then display the message 'Hello,'as shown in Figure 5.6. This greeting is much more readable than Figure 5.3, in which the original PRINT command remains on the screen.

The set of *tab* commands lets you position the cursor anywhere you want on the screen. These commands allow you to move directly to a certain row or column without printing spaces or blank lines.

Think of the text screen as a grid 31 columns wide and 24 rows high, as shown in Figure 5.7. Each row and column is given a number. The columns are numbered 1 to 31, from left to right. The rows run from 0 to 23, top to bottom. Don't be confused by the fact that the columns start their numbers from 1 and the rows from 0. It's just a quirk of the Adam's design.

With the VTAB command, you can set the vertical position of the cursor. To place the cursor halfway down the screen, you could type

```
vtab 12
```

The cursor will immediately move to the twelfth row down from the top of the screen.

VTAB merely moves the cursor; it doesn't print anything. To place a message in row 12 at the left edge of the screen, follow the VTAB command with a PRINT statement:

```
vtab 12: print "Hello"
```

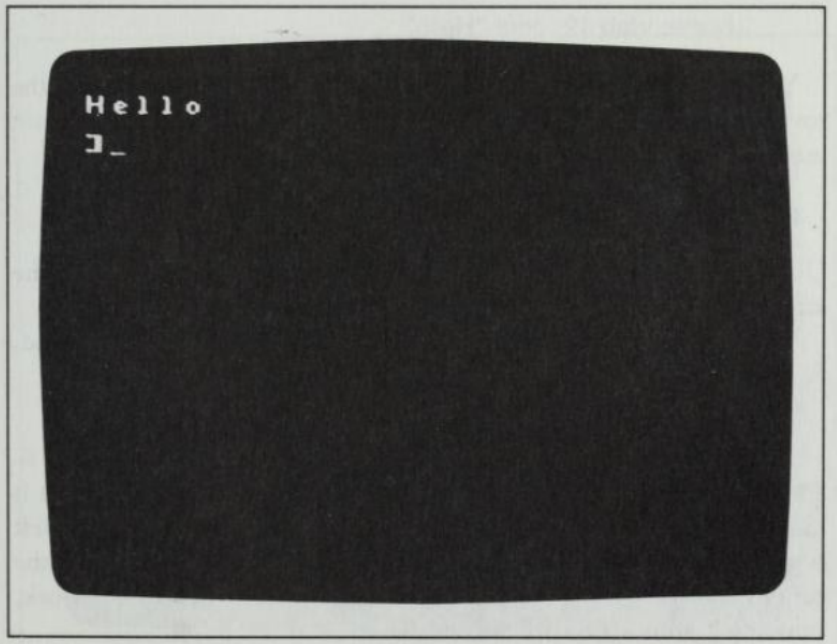


Figure 5.6: *By combining the HOME command with a PRINT statement, you can clear the screen before displaying your message.*

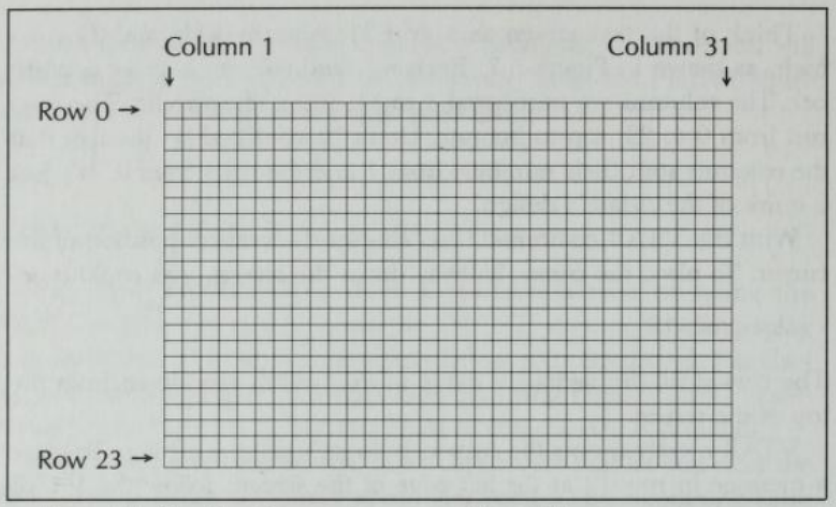


Figure 5.7: *The Adam's screen is arranged like a grid.*

If you want your message displayed against a blank screen, add a HOME command to the beginning of your line:

```
home: vtab 12: print "Hello"
```

You can use the VTAB command to put the cursor on any of the rows from 1 to 23. The top row on the screen is numbered 0, but unfortunately the computer will not accept the command.

```
vtab 0
```

Use the home command to move the cursor to the top line of the screen.

To move the cursor horizontally, you can use the HTAB command. As with VTAB, you can give this command by itself:

```
htab 25
```

The computer will move the cursor to column 25. However, since it has not been given a message, it will drop straight down to the left edge of the next line on the screen. It therefore looks as though the computer didn't do anything. To see the HTAB command work, combine it with a PRINT statement:

```
htab 25: print "Hello"
```


The computer will move the cursor, display the word Hello near the right edge of the screen, and then drop down to the next line.

VTAB and HTAB are often used together in order to move the cursor to a specific point on the screen. Suppose you want to place an X right in the center of the screen. You would need to move the cursor halfway down on the vertical scale of 1 to 23, and halfway across the horizontal scale of 1 to 31. The following command would do that:

```
vtab 12: htab 16: print "X"
```

Note that the tab commands specify only the starting point for the printed message. If you want to center a longer message, change the HTAB command, as in Figure 5.8.

You can use one other formatting feature to arrange your display. With a simple PRINT statement, the computer drops down to the next line after it finishes displaying your message. If you then display another message, it will appear on a fresh line. To review, type the following double command:

```
print "First":print "Second"
```

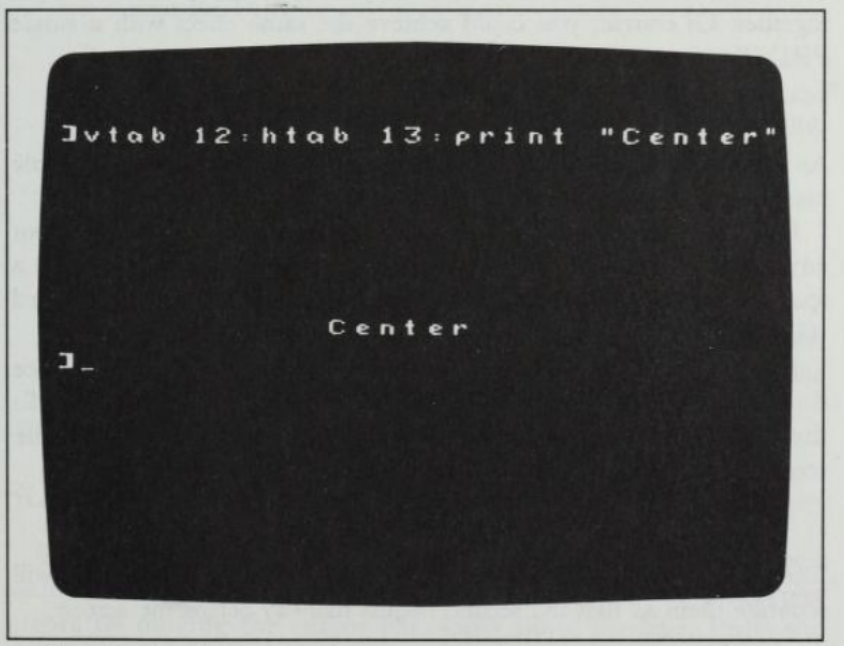


Figure 5.8: *Use the VTAB and HTAB commands to center a word on the screen.*

The words will appear on two different lines, just as if you had typed separate PRINT statements.

If you want to display the second message on the same line, type a semicolon (;) after the first message. Try the following:

```
print "First";:print "Second"
```

The semicolon tells the computer not to start a new line after the first message. Instead, the computer runs the two words together:

```
FirstSecond
```

A semicolon after "Second" would make no difference here, since the computer always drops down to a new line at the end of the last command in a group.

You can also combine several messages in a single print statement. Instead of using the two PRINT statements in the preceding example, you could type:

```
print "First";"Second"
```

As before, the semicolon tells the computer to run the two messages together. Of course, you could achieve the same effect with a single PRINT message:

```
print "FirstSecond"
```

As your commands become more complicated, however, multiple messages will give you more flexibility.

In these examples, the two words have been run together without any space. If you want a space between the two messages, just type a space as a part of the first message, immediately following the word 'First'.

If you want a larger space between the messages, you can replace the semicolon with a comma. In a PRINT statement, a comma tells the computer to separate the messages into two columns on the screen. Type the following command line:

```
print "First","Second"
```

The computer will display the two messages on the same line, but will separate them so that the second begins halfway across the screen:

```
First          Second
```


The second message always starts in the same column, as long as the first message is shorter than 15 letters. This technique is often useful if you need to display a chart in two columns.

Some of these special features of the PRINT command may seem rather silly while you're typing only one command at a time. Rest assured, they will become very useful when you begin writing more complex programs.

OPTIONAL EXERCISES

Now that you have finished this introduction to your computer's keyboard and screen, you might want to try your hand at some optional exercises. I hope you'll try them, but you won't need to have done them to read further in this book.

1. Use the arrow keys to create block letters on the screen. Move the cursor around and place asterisks (*) to form large block letters. Remember to use only the arrow keys as you do this: you will spoil your picture with an error if you press RETURN.
2. On page 74, we put more than one PRINT statement in a single command by using colons (:). Write one command that combines three PRINT statements to produce the following message:

   ```
   ONE
   TWO
   THREE
   ```

3. Use HOME, VTAB, and HTAB commands to center the word Bottom on the last line of the screen. Why does the word get pushed one line up just after it is displayed?

SUMMARY

To type a BASIC command, you use the keyboard just as you would use a typewriter or the Adam's word processor. You can correct errors by moving the cursor backward along the line. Press the RETURN key to have the computer carry out the command.

With the PRINT statement, you can ask the computer to display messages on the screen. You can display several lines at once by using several PRINT commands on the same line, separated by a colon.

Using the HOME, VTAB, and HTAB commands, you can make your screen more attractive and more readable. The HOME command clears the screen and places the cursor in the upper-left corner. The VTAB and HTAB commands move the cursor to a specific point on the screen, so that you can print messages anywhere you want.

Perhaps without realizing it, you have covered a lot of ground in this chapter. You have learned everything you need in order to type BASIC commands, and you know how the computer responds on the screen. Most important, you have given your first command, and the computer gave its response. In the chapters ahead, you will learn many other commands that will increase your control of your computer.

Chapter 6
Graphics

If you have played Buck Rogers: Planet of Zoom or any other Coleco video game, you know that your Adam can display more than just letters and numbers on your television screen. It can draw very colorful pictures and produce impressive animation effects.

This chapter discusses your computer's graphic display system. You will learn how to use coordinates, and how to plot points on the screen. With a few simple commands, you will be able to choose colors, plot points, and draw pictures.

Graphics are one of the most appealing features of your Adam's BASIC language. I hope you will enjoy them.

WHAT ARE GRAPHICS?

Graphics refers to any type of display you can put on the screen other than plain text. Your computer can display pictures such as a box or a cube, and bar or line graphs. In Buck Rogers: Planet of Zoom, you saw your computer display animated scenes.

The Adam has two different modes for displaying pictures. With *low-resolution graphics,* you can compose pictures made up of rectangular blocks and thick lines. With *high-resolution graphics,* you can plot much finer detail and paint intricate line drawings. By choosing the appropriate mode, you can get just the amount of detail you want. For example, to construct a bar graph, you will want to have thick lines, with

relatively little detail. For an intricate picture, you will want finer lines and greater precision. This chapter gives an overview of both modes.

COORDINATES

The Adam uses *coordinates* to set up its graphics screens. Coordinates are a way of labeling points on the screen by numbering their horizontal and vertical positions.

Coordinates always come in pairs. With the first number, you choose the column, telling the computer how far to move in from the left edge of the screen. With the second number, you pick the row, how far you want to go down from the top. The point you choose will be the box where that row and column meet.

The first number in a pair of coordinates is often called the *x-coordinate;* the second is the *y-coordinate.* Programs in this book will often use the letters X and Y to refer to horizontal and vertical locations. When you see a pair of coordinates in the form 'X,Y', you can think of this as "X spaces over and Y spaces down."

In Chapter 5, you saw an example of coordinates in the 24 × 31 grid for the VTAB and HTAB statements. The number in the HTAB statement represents an x-coordinate, since it names the horizontal distance in from the left. The VTAB number is a y-coordinate, determining the vertical distance down from the top. By combining an HTAB statement with a VTAB, you were able to name any point on the grid.

In the two graphics modes, the coordinates are numbered differently, but the concept is the same. The point in the upper-left corner is always 0,0 (0 spaces over and 0 spaces down). In the low-resolution mode, the grid is 40 columns wide and 40 rows high. In high-resolution graphics, there are 256 columns and 159 rows.

LOW-RESOLUTION GRAPHICS

The low-resolution graphics mode is used to plot solid rectangles and thick lines. While it does not allow enough detail for real drawing, the low-resolution mode is easy to use and offers brilliant colors. It is particularly useful for bar charts and other pictures that require thick lines.

To enter the low-resolution graphics mode, type the command

```
gr
```

and press RETURN. The screen will be cleared and the bracket prompt will reappear near the bottom.

The upper part of the screen is now reserved for the graphic designs you are about to create. You cannot use this region for printed messages, but you can use it to draw pictures out of points and lines. This region is called the *graphics screen.*

The graphics screen does not reach all the way to the bottom, because the computer has to leave you some room to enter commands and display text messages. The last four lines are reserved as a *text window,* so that you can still see your commands displayed.

Sometimes you may want to see more text than can fit in the text window. To do so, you have to return to the regular text mode. Type the following command and press RETURN:

```
text
```

This reverses the GR command and lets you type on the full screen. You will need to type another GR command if you later want to return to graphics.

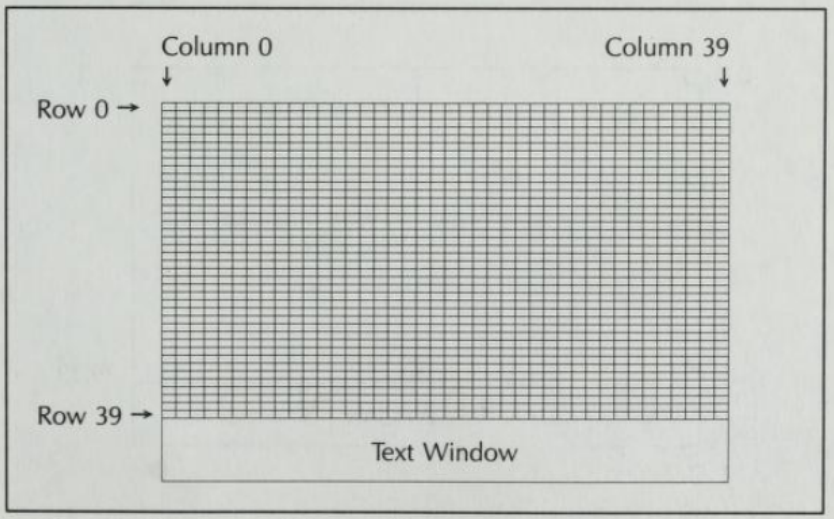


Figure 6.1: *The low-resolution graphics screen.*

Figure 6.1 shows how the screen is laid out in the 40 × 40 low-resolution graphics mode. Columns and rows are each numbered from 0 to 39, with the point in the upper-left corner numbered 0,0. The point in the upper-right corner is 39,0—that is, 39 columns over and 0 rows down. The point at the lower-left corner, which is 0 columns over and 39 rows down, has the coordinates 0,39. Similarly, the point at the lower right has the coordinates 39,39. These four points are shown in Figure 6.2.

To name the point in the center of the screen, we need to choose coordinates halfway between 0 and 39. You could choose either 19 or 20, but we'll use 19 so that we don't crowd the right-hand side and the bottom of the screen. The coordinate 19,19 names the point shown in the middle of Figure 6.2: it is 19 columns over and 19 rows down.

Before you start drawing, you need to choose a color. Your Adam offers sixteen colors in each of its graphics modes. The colors in both modes are essentially the same, but unfortunately, they are numbered differently. In both cases, the colors are given code numbers between 0 and 15. Figure 6.3 shows the colors codes for the two graphics modes. When you first switch into the low-resolution graphics mode, the color is set to 0, or black. Since black points don't show up against a black background, you need to choose another color by giving a COLOR

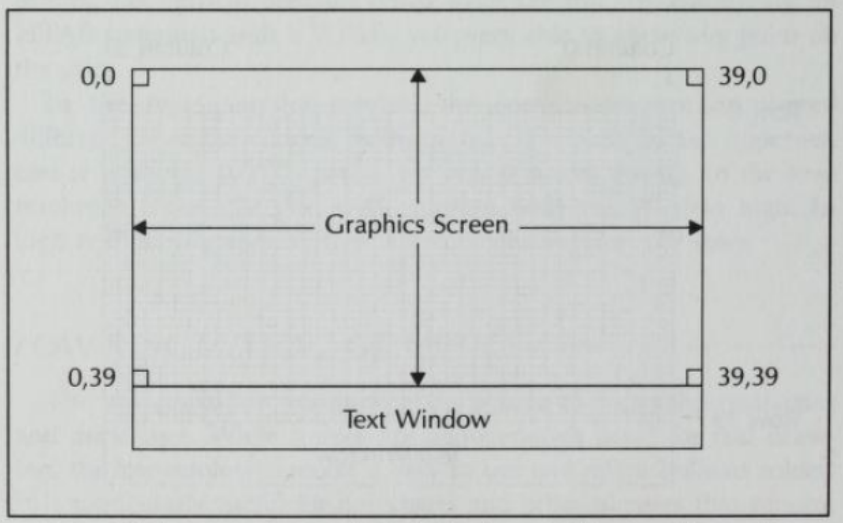


Figure 6.2: *Some sample coordinates in the low-resolution graphics mode.*

command. To draw in a medium green (color number 6), type:

```
color = 6
```

You can give another COLOR command any time you want to paint with a different color.

The numbering of the sixteen low-resolution colors are somewhat jumbled, but they do follow a general order from dark to light. The quality of the colors will depend on your television and how you have it adjusted, but you will probably get the most vivid tones from the colors in the middle: 6 for medium green, 7 for light blue, 8 for orange, and 9 for medium red.

Now let's try some graphics. Type the starting commands if you haven't already done so:

```
gr
color = 6
```

These commands select the low-resolution graphics mode and choose the color green. Any time you want to clear the screen and start over, just retype these two commands.

CODE	COLOR *(low-resolution)*	HCOLOR *(high-resolution)*
0	Black	Black
1	Purple	Green
2	Dark Blue	Dark Red
3	Dark Red	White
4	Dark Green	Black
5	Gray	Medium Red
6	Medium Green	Medium Blue
7	Medium Blue	White
8	Orange	Orange
9	Medium Red	Dark Blue
10	Gray	Gray
11	Light Red	Light Red
12	Light Green	Dark Green
13	Light Yellow	Light Yellow
14	Light Blue	Light Blue
15	White	Purple

Figure 6.3: *The colors for low- and high-resolution graphics.*

The simplest low-resolution graphics command is the PLOT statement, which lets you fill in one block of the grid with the color you have chosen. With this command, you name the coordinates of the point you want to fill. Try typing this line, for example:

```
plot 0,0
```

A small green square should appear in the upper-left corner of the screen.

If it does not, make sure you have given the GR and COLOR commands. Also, make sure that your television is tuned finely enough for you to see this small square. It is way up in the upper-left corner, so you may need to move the image to see it. Use the horizontal- and vertical-hold knobs on your television set to adjust the image.

You can plot as many squares as you wish on the screen. Try a few, to get a feel for the graphics coordinates. Each PLOT command will result in another single square of the last color you have chosen (green, in this case).

In using graphics commands, you may occasionally get the message

```
?Illegal Quantity Error
```

This means that you are trying to paint a box that is off the screen—one of your coordinates is too large. Remember, the maximums for the X and Y coordinates are both 39 in the low-resolution graphics mode.

With the plot command, you get disconnected boxes. If you want lines, you'll need to learn two other commands: HLIN and VLIN. The two work identically: HLIN is for horizontal lines, and VLIN is for vertical. Let's start with HLIN.

The HLIN command lets you draw a horizontal bar of any length. To use it, you need to supply three numbers. First, you must give two x-coordinates, to name the left and right endpoints of the line. Then, you must give a single y-coordinate to tell how many rows down from the top the horizontal line should be. The actual command will be of this form:

```
hlin xleft, xright at y
```

You will type actual numbers to replace the three italicized expressions.

Let's try some examples. Suppose you want to draw a bar all the way across the top of the screen. This bar will begin at the left (xleft = 0) and end at the right (xright = 39), and run across the top row of the graphics screen (y = 0). You therefore type:

```
hlin 0,39 at 0
```

When you press RETURN, the bar will appear, colored green unless you've changed to another color.

Suppose you want a shorter line in the center of the screen. You could type:

```
hlin 10,30 at 20
```

As with the PLOT command, you can use HLIN to draw as many lines as you wish on the screen. Try some experiments with lines of your own.

The VLIN commands follow the same principle to produce vertical lines. But now the first two numbers represent the y-coordinates of the top and bottom endpoints of the line, and the third number names the column in which the vertical line should fall. The general form of the VLIN command is therefore:

```
vlin ytop, ybottom at x
```

To place a vertical line down the center of the screen, try:

```
vlin 0,39 at 20
```

When using HLIN and VLIN, you must be careful to avoid one pitfall: the first two numbers in the command must be different. You must always draw a line *from* someplace *to* someplace else. If you try to draw a line whose endpoints are the same number, the computer becomes confused and draws a bizarre pattern across the entire screen. It may also leave you without any way to type other commands. This is a rather serious defect in the Adam's BASIC language, which you must work around. If you do fall into this trap, you can try to regain control by typing

```
text
```

even though you can't see your letters appear on the screen. If this doesn't get your cursor back, you may need to reload the BASIC cassette, using the instructions in Chapter 5.

To see these concepts in action, let's draw the box shown in Figure 6.4. Start by clearing the screen:

```
gr
```

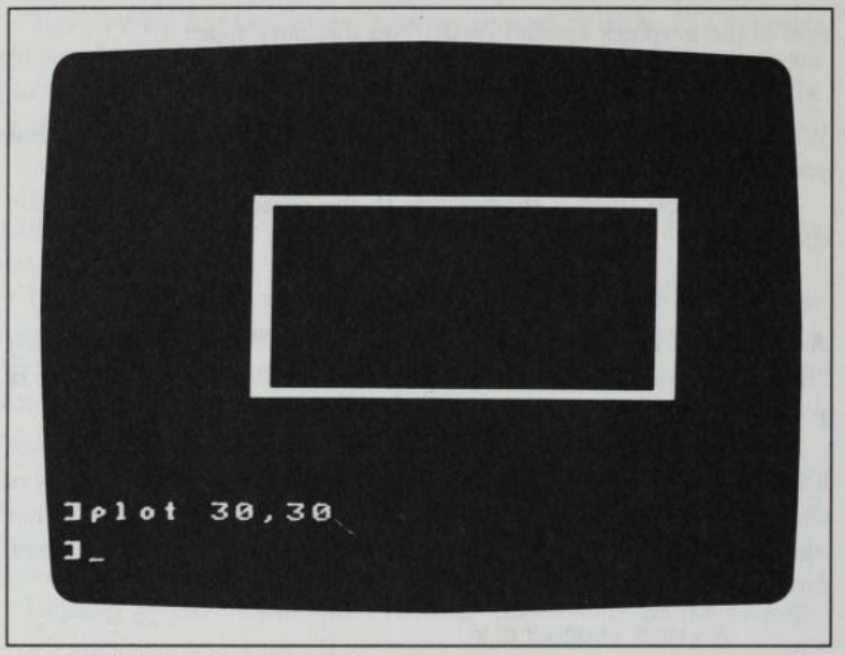


Figure 6.4: *Drawing a box on the screen.*

We could continue using green, but let's choose orange:

```
color = 8
```

Now let's draw the top of the box, from 10 units at the left to 30 at the right, on the tenth row down from the top:

```
hlin 10,30 at 10
```

Then add the bottom of the box on the thirtieth row:

```
hlin 10,30 at 30
```

Now do the same for the two sides:

```
vlin 10,30 at 10
vlin 10,30 at 30
```

You'll notice that the computer does not connect the lines in the lower-right corner. So you'll need to add one more square with a PLOT command:

```
plot 30,30
```

The computer omitted this box because with the HLIN and VLIN commands, it draws the line only up to the second endpoint, but not including it.

There are several ways you can erase portions of your screen. The simplest is to paint with black, the color of the background, by typing

```
color = 0
```

When you do this, your PLOT, HLIN, VLIN statements become *erase* commands, until you choose another color. Try typing the following lines:

```
color = 0
hlin 10,30 at 10
```

The top of the box should disappear. (If it doesn't, you probably mistyped one of the coordinates. Try retyping the last two examples.)

If you want to erase the whole screen, the best thing to do is to type

```
gr
```

This resets the screen and lets you start all over. You now need to type another COLOR command before you can paint again.

HIGH-RESOLUTION GRAPHICS

While the Adam's low-resolution graphics are quite useful, you are limited to plotting squares, rectangles, and thick lines. In the low-resolution mode, you cannot draw a picture with fine details, nor one with diagonal lines. You can draw only vertical and horizontal bars.

As an alternative, your Adam offers a high-resolution graphics mode that lets you plot small points and narrow lines. This allows you to use finer detail and diagonal lines in the pictures you are drawing.

The high-resolution graphics screen is divided into a fine grid 256 dots wide and 159 dots high, as shown in Figure 6.5. The graphics coordinates follow the same system as the coordinates in the low-resolution mode, except that there are more rows and columns. Figure 6.5 shows the coordinates you would use to plot the points in each corner of the graphics screen.

All the high-resolution graphics commands begin with the letter *h*. While the commands are different from the ones you learned for low-resolution graphics, they are very similar. All the low-resolution commands except for HLIN and VLIN have corresponding high-resolution commands.

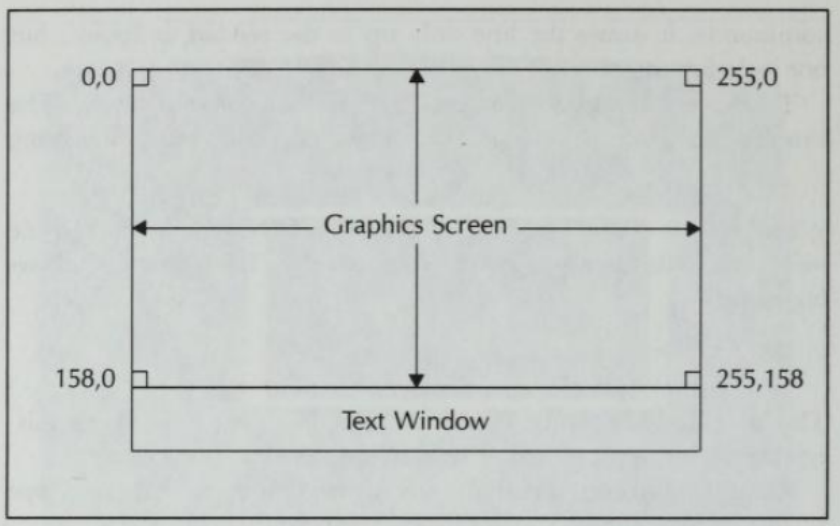


Figure 6.5: *The layout of the high-resolution graphics screen.*

To enter the high-resolution graphics mode, type the command

```
hgr
```

Like the GR command in low-resolution graphics, this command clears the screen and places the cursor down in the text window.

As in the low-resolution mode, you must use a command to choose a color before you start. Otherwise, all the points will be black and invisible against the screen's background.

To select a color, use the command HCOLOR, with the codes shown in the right column of Figure 6.3, on page 87. These are essentially the same 16 colors as in low-resolution graphics, but they are numbered differently. To change the color to medium red, you would type:

```
hcolor = 5
```

On most screens, the sharpest high-resolution colors are medium red (5), medium blue (6), orange (8), and dark green (12).

A single command, HPLOT is used to draw both points and lines in high-resolution graphics. This command tells the computer to paint points and lines on the screen with the color you have last chosen.

To plot a single point on the screen, use the HPLOT command with one set of coordinates. Make sure you're in the high-resolution graphics mode and that you've chosen a color, then type:

```
hplot 20,15
```

A pinpoint of light should appear in the upper-left corner of the screen. If it does not, try typing the HGR and HCOLOR commands again and make sure your set is tuned finely enough to see this isolated point.

The '20,15' in the HPLOT statement names the coordinates of the point: 20 units over from the left, 15 units down from the top. These are relatively small movements on the 256 × 159 high-resolution grid. If you want to see exactly how small, plot the point in the upper-left corner:

```
hplot 0,0
```

Note that this point would be much further from the edges in the low-resolution graphics mode.

On some television sets, this corner point might fall off the left side of the screen. You can usually bring it into view by adjusting the television's horizontal-hold knob, but on some sets even that won't help. If you cannot get this point on your screen even after adjusting your set, you may need to avoid using the first few columns of the graphics screen.

The HPLOT command also lets you draw lines between points. To draw a line, you need to name two sets of coordinates: one for the starting point and one for the destination. The HPLOT command for a line looks like this:

```
hplot 20,20 to 255,158
```

Type this command and press RETURN. The computer will draw a red line from a point near the upper-left corner all the way down to the lower-right corner of the graphics screen.

If you look closely, you will see that this diagonal line is not exactly straight. This "staircase effect" is caused by the limited resolution of a television screen. A line may also appear to have a color other than red, depending on the angle at which it is drawn on the screen. High-resolution lines on the Adam tend to look pale or purplish, and may abruptly change color at a point in the middle. Horizontal lines give the most consistent colors.

In most cases, you need to give the coordinates of two points to draw a line with HPLOT. When you give the HPLOT command, you ask the computer to pick up its graphics paintbrush and plot a point at the first set of coordinates. Then by adding the word 'to' and a second set of coordinates, you ask the computer to draw a connecting line to the second point. Try a few more line commands, such as these:

```
hplot 0,90 to 210,90
hplot 178,15 to 29,109
hplot 255,158 to 0,0
```

Some of these lines may seem to have been drawn backward or to cross other lines. That is okay: the computer always draws from the first point in the command to the second.

Each of these commands produces a line unconnected to the others because the computer lifts its paintbrush each time to move to the starting point of the new line. Often, however, you will want to connect a new line to the endpoint of the preceding one, so that you can produce an unbroken series of lines.

To continue your line from the preceding line's endpoint, start right off with the word 'to' in your HPLOT command, without naming a starting point. Try typing the following command, for example:

```
hplot to 128,0
```

The computer will draw a line from the end point of your last line to the top-center of the screen.

You can also use the HPLOT command to draw several lines at once. Type a single HPLOT command with three or more sets of coordinates, each separated by the word 'to'. For example, try typing

```
hplot 10,45 to 35,60 to 15,140
```

The computer will first draw a line from the starting point 10,45 to the first endpoint, 35,60. Then, it continues and draws another line to the final endpoint, 15,140. In this way, an HPLOT command can draw a whole series of lines. The lines must, however, be connected: you can have only one starting point in each HPLOT statement. You may find it easier, though, to use separate HPLOT commands to draw extended series of lines.

The word 'to' is therefore the key to how the computer responds to your HPLOT commands. When you name a point at the beginning of an HPLOT command without typing 'to' before it, the computer will move to those coordinates without drawing a line. If you then go on to type a set of coordinates preceded by the word 'to', the computer will draw a line to that new point from the previous line's endpoint. You can add as many 'to' points as you want after the initial command word HPLOT.

As an example of the HPLOT command, let's draw a box like the one we made using the low-resolution commands. Start by clearing the screen, with the lines:

```
hgr
hcolor = 8
```

Then give the following four commands:

```
hplot 20,20 to 140,20
hplot to 140,140
hplot to 20,140
hplot to 20,20
```

Each HPLOT command draws an edge of the box, and the fourth one brings the paintbrush back to its starting point. The first command specifies an initial starting point for the line, but the other three do not because each line starts from the endpoint of the line before. The final result should be a box like the one in Figure 6.6.

If you want, you can also combine these four HPLOT commands into one. Clear the screen by typing

```
hgr
hcolor = 8
```

Then type

```
hplot 20,20 to 140,20 to 140,140 to 20,140 to 20,20
```

This extended command will run onto a second line on your television screen, but that doesn't matter. The line will still work.

In most cases, it is better to use shorter commands. The compound HPLOT command does the same thing in fewer words, but you will have to correct the entire command if you make a single mistake. With separate commands, you can usually back up and correct your errors right after you make them.

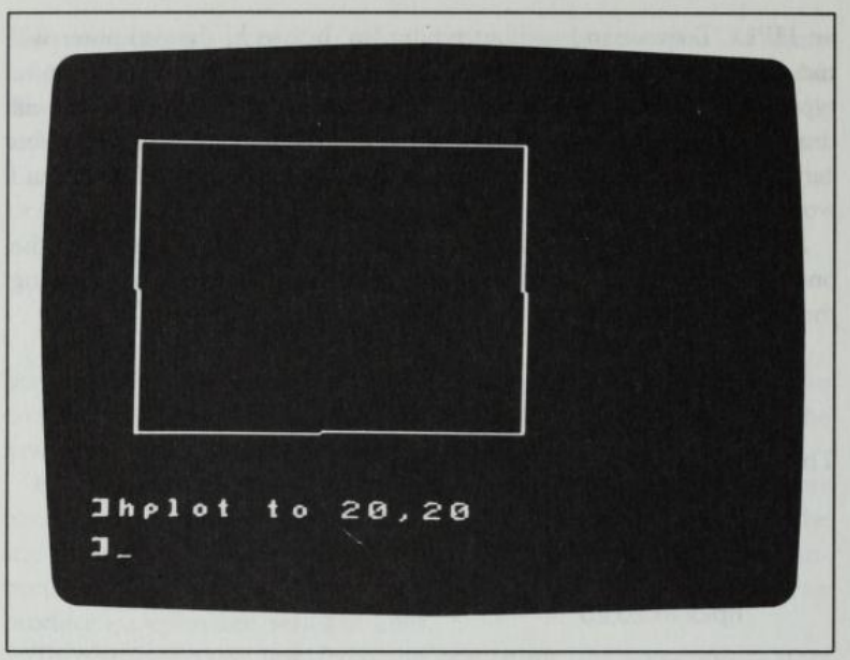


Figure 6.6: *Drawing a box on the screen.*

As with low-resolution graphics, you can change colors at any time. You'll discover, though, that if you plot points or lines of different colors near one another, the colors of some points may change. Your screens will generally look all right as long as you don't try to paint regions of different colors right next to each other. Paint with the color black if you want to erase portions of your screen.

This brings us to the end of our brief introduction to the Adam's graphics. In subsequent chapters, we will use these graphics features in many different ways.

OPTIONAL EXERCISES

1. Draw a series of horizontal bars across the screen using low-resolution graphics. Type a COLOR command before each line, so that each bar is a different color.
2. Rerun the commands you used to draw the orange box in Figure 6.6. Then give a second series of commands to draw a blue box inside.

SUMMARY

In this chapter, you have learned how to draw pictures on the screen using your Adam's two graphics modes. By naming a point with its coordinates, you had the computer plot it or draw a line to it.

The three basic commands in the low-resolution graphics mode let you plot solid rectangles and continuous lines. With the PLOT command, you can fill a single rectangle on the screen. With HLIN and VLIN you can draw an extended horizontal or vertical bar on the screen.

In high-resolution graphics, you can draw detailed lines and points. While the fine detail makes it harder to fill full regions of the screen, the high-resolution mode is generally the more flexible. A single general-purpose command, HPLOT, is used for all high-resolution drawing.

You have now learned a variety of BASIC commands. In Chapter 5 you used the PRINT statement to display messages on the screen, and the HOME, HTAB, and VTAB commands to arrange text messages and clean up your screen display. In this chapter, you have learned to paint pictures on your screen. All these commands will be useful in the chapters to come.

To use these commands more efficiently, you need a way to save them. The procedure for drawing a low-resolution box, for instance, took seven separate commands, each of which had to be entered from the keyboard. Suppose you wanted to draw the box again. Would you need to retype all those steps again? No, there is an easier way, and that is the subject of the next chapter.

Chapter 7

Writing a Program

In Chapters 5 and 6, you learned how to give commands that print messages and draw pictures. As you gave your commands, the computer immediately put them into action.

In those chapters, however, there was one great limitation. You had to type every command from the keyboard. If you made a small mistake in a series of commands, you had to clear the screen and start over from the beginning. Also, when you wanted to repeat a certain task (such as drawing a box), you needed to retype the whole series of commands. These restrictions limited you to fairly short procedures.

With a *program,* you can store a series of commands in the computer's memory. Then, when you are ready, you have the computer run the entire series as a group. The computer will react to the commands as if you had just typed them on the keyboard, but it will store them after it has finished, so that you can reuse them. If one of the steps is wrong or if you want to try a variation, you are free to alter the program and run it again.

Because programs let you design a procedure and reuse it many times, they can be used for complicated tasks. If you had to type your commands one by one, you might never want to try anything very complex. But with a stored program, you can combine commands into powerful procedures that you can use over and over. Many people create programs involving hundreds of stored commands.

Programs also give you full control over your computer. In a program, you can store words and numbers as you go along, and perform calculations. You can take a group of commands and run them

over and over in a loop, or have your computer do something only if a certain condition is fulfilled. With these powerful tools, you can direct your computer to perform complex tasks, then sit back while it does the work.

This chapter will show you how to store a program in the computer's memory and how to run it. A few short hints and examples will help you get started. Also, you will learn how to use your Adam's cassette recorder and printer to save your programs and print your work.

STORING YOUR PROGRAM

A program consists of a series of commands stored in the computer's memory. When you are ready to use the stored commands, you tell the computer to run the program. It then carries out the commands in order.

Before you do anything else, type

```
new
```

and press the RETURN key. This NEW command tells the computer to clear everything from its memory and from the screen.

You should give the NEW command every time you begin a new program. If you don't, program lines may remain in the computer's memory from some other task. These leftover lines could slip into the program you are typing and disrupt it. You should make the NEW command a habit.

If your computer is still in the high-resolution graphics mode from Chapter 6, type the command

```
text
```

Many of the examples in this chapter use the regular text mode, rather than graphics—you'll need to switch back to text so that you can follow.

You can store any BASIC command as a line in a program by placing a number in front of it. This *statement number* signals the computer to store the command rather than respond to it immediately.

In Chapter 5, you typed your first command from the keyboard:

```
print "Hello"
```


When you pressed RETURN, the computer displayed the word Hello on the screen. Now let's try a small variation. Type the following line:

```
10 print "Hello"
```

When you press RETURN, nothing seems to happen. The statement number before the PRINT command told the computer to store the line in its memory. There is nothing special about the number 10; we could have used any number.

You can check to see that the line is stored by typing another command:

```
list
```

In response, the computer will display all the lines it has in its memory. In this case, there is only one such line, number 10.

Now let's ask the computer to carry out the stored command. To do this, type

```
run
```

and press RETURN. You get the same result as you did when you typed the command without a number: the computer prints the word Hello on the screen. The action is delayed, however, until you give the RUN command.

If you again type

```
run
```

the computer will repeat the commands you have stored; here it will again display the word Hello. You can do this as many times as you like: your commands remain stored in the computer's memory until you turn the power off or clear the memory by typing

```
new
```

To add a second line to your program, just type another command with a different statement number in front of it. Since we used 10 for our first command, let's use 20 for our second:

```
20 print "Hello Again"
```

When you press RETURN, the second statement will be stored in the memory along with the first.

You can see how your program is now stored in the computer by typing

```
list
```

The computer will display all the statements it has stored:

```
10 PRINT "Hello"
20 PRINT "Hello Again"
```

Each statement is stored as a unit. The computer stores and lists statements in the order of their statement numbers, regardless of the order in which you typed them.

The computer displays the command words in all-capitals when it lists the program. Even when you type commands in lowercase, the program listings show them as uppercase. The Adam does this to follow the convention used by most other computers, which type all program lines in capital letters.

These capital letters introduce an unfortunate inconsistency between your typed program lines and the computer's program listings. As I explained in Chapter 5, you will find it easier to type your commands in lowercase, so that you don't have to press the LOCK key constantly when typing numbers. But the computer will change these to capitals in its listings. You will simply need to get used to reading the lines both ways.

In this book, I will continue to use lowercase letters for lines that you type on the keyboard, since that is the way you will type your commands. On program listings, I will reproduce the computer's capital letters exactly as they appear on the screen.

You now have two program lines in your computer's memory. When you give the RUN command, the computer will search through its memory and follow the stored commands in numerical sequence. Then it will drop down and show the bracket prompt once again. The results will look like this:

```
Hello
Hello Again

]
```

You can add statements anywhere in your program. Suppose you wanted a message to appear between the two lines you have already printed. You can simply type a command whose statement number

is between 10 and 20. The computer will file the new command in the proper place. You could, for example, type:

```
15 print "Goodbye"
```

If you LIST the program, you will see that the new statement has been placed between the other two:

```
10 PRINT "Hello"
15 PRINT "Goodbye"
20 PRINT "Hello Again"
```

RUN the program. The three commands will be carried out in the order of their statement numbers:

```
Hello
Goodbye
Hello Again
```

You can choose any number from 0 to 65535 for your statement numbers. You would do best, however, to use multiples of ten (10, 20, 30, etc.). This system gives an ordered appearance to your program and leaves plenty of room for insertions.

To delete a line, just type the statement number followed by nothing. Suppose you want to eliminate the last line, "Hello Again." Just type

```
20
```

and press RETURN. The statement numbered 20 will be erased from the computer's memory, as you can tell by listing the program:

```
10 PRINT "Hello"
15 PRINT "Goodbye"
```

If you now run the program, the computer will use only the statements that remain:

```
Hello
Goodbye
```

If you want to change a line, type the new version with the old statement number:

```
10 print "Hello There"
```

When you press RETURN, this new line will replace the earlier

statement number 10. When you now LIST your program it will look like this:

```
10 PRINT "Hello There"
15 PRINT "Goodbye"
```

Remember that you can use each statement number only once. If you duplicate a number, your new statement will replace the previous one.

If you want to make only a minor change in a line of your program, you can edit the line on the screen. The Adam's editing commands are rather clumsy, so you may often find it easier simply to retype the line, but for long statements, you may prefer to use these commands.

As we saw in Chapter 5, you can change and reuse lines that you have already typed. Move the cursor to the line by using the up arrow on the keyboard. Then use the right arrow to position the cursor under the letter you want to change. Type your correction over the old letter, then finish the line. If you are replacing only one letter, you can use the right arrow again to reach the end of the line without changing the letters beyond your correction. If you are adding or removing a character, you will need to type the remainder of the line. When you reach the end of the line, press the RETURN key to enter the new version into the computer's memory.

After making a correction in a line, you must either retype the remainder of the line or use the right arrow to run the cursor past every letter you want to save. If you press the RETURN key before you have run the cursor past the remainder of the line, the computer will erase those remaining characters, even though they appeared to be part of the line on the screen.

You can edit only statements that are displayed on the screen. If you want to change a line that is not currently displayed, you could LIST the entire program, but it is simpler to display just the line you want to change. You can LIST a particular line with a command such as this:

```
list 15
```

With this version of the LIST command, only line 15 will be displayed:

```
15 PRINT "Goodbye"
```

Suppose you want to insert an exclamation point after the word Goodbye. Use the arrow keys to move the flashing cursor under

the second quotation mark, like this:

```
15 PRINT "Goodbye"_
```

Then type the exclamation point: it will replace the closing quotation mark on the screen. Since the closing mark is now missing, you must type another quotation mark to finish the line.

To store the corrected line in the computer's memory, press the RETURN key. The corrected line will replace the previous version and become part of the program. The computer will make the replacement whether you retyped the entire line or simply changed one character and ran the cursor past the rest. Type a LIST command to make sure your change has been incorporated:

```
10 PRINT "Hello There"
15 PRINT "Goodbye!"
```

Don't worry if you find these editing commands difficult to use. The Adam's BASIC editor is so clumsy that many people prefer to retype lines rather than edit them. Also, if you want to make quite a number of changes in a long program, you can store it as a file on a cassette tape, then make your changes using the Adam's word processor. More on this technique later in this chapter.

Let's quickly review the different ways you can use statement numbers as you type your program:

- *Add.* If you enter a line with a statement number that you haven't used before, it will be stored as a new line of your program.
- *Insert.* If you type a line with a statement number that lies between two numbers already stored, the line will be inserted between these two lines in the program.
- *Replace.* If you enter a line with a statement number that you have already used, your new version will replace the old.
- *Delete.* If you enter a statement number with no command following it, any statement stored under that number will be eliminated.

You now know everything you'll need to type programs into the computer. You know how to add, insert, replace, and delete program lines. You know how to list your program, and how to run it.

It is time for a practical example.

THE BOX-DRAWING PROGRAM

At the end of Chapter 6, you typed a sequence of high-resolution graphics commands to draw a box on the screen. To review, try the same series again:

```
hgr
hcolor = 8
hplot 20,20 to 140,20
hplot to 140,140
hplot to 20,140
hplot to 20,20
```

The computer will shift into the four-color graphics mode and produce a picture that resembles Figure 6.6 on page 96.

We're going to write a new program, so type

```
text
```

to return to the text mode. Then type

```
new
```

to clear the computer's memory. Remember: a program line remains in the memory until you delete it, clear the slate with a NEW command, or turn the computer off.

Now we're ready to retype the six commands, but this time we'll give them numbers and store them as a program. Type these lines:

```
10 hgr
20 hcolor = 8
30 hplot 20,20 to 140,20
40 hplot to 140,140
50 hplot to 20,140
60 hplot to 20,20
```

When you're done, check your typing. If you made a mistake, retype the line or correct it with your editing keys.

Now run the program. Almost instantly, your computer will shift to the graphics screen and draw an orange box. The result is the same as Figure 6.6, but the computer did it all in one step, rather than six.

Several advantages of stored programs now become evident. You can correct a mistake by editing a line, rather than retyping the entire procedure. You can redraw the box with a single RUN command, rather than retyping all six steps. As we go on, you will discover other advantages and other uses for programs.

HINTS ON WRITING PROGRAMS

As you write your own programs, you should develop habits that will make your work easier. Clear thinking and good organization are often the difference between successful programming and computer nightmares. Here are three rules to follow:

1. *Think the task through.* You will save yourself a lot of agony if you spend a few minutes planning your program before you start typing it. You don't actually have to write out the program unless you feel the need, but you should sketch the general route before you start.

2. *Be neat.* There is nothing more frustrating than trying to make corrections in a sloppy program. If you organize your program so that it is easy to read and logical, you will be able to follow it more easily when you go back to change it. Use a regular pattern in your statement numbers, such as 10, 20, 30.

3. *Leave notes to yourself.* There is a special command that lets you place comments in your program. You simply start a line with the keyword REM (for "remark"), and the computer will ignore anything you type on the rest of the line.

To place a comment in your program, type a new statement number, the keyword REM, and your comment. For example, at the beginning of the box-drawing program you could add a comment to give a title to the program. Try the following:

```
5 rem Box-Drawing Program
```

When you LIST the program, this remark will appear first, since 5 is the smallest statement number in the program. When you run the program, however, the computer will ignore the remark.

Don't rush to dismiss REM statements as a waste of time. Even if you are writing programs only for yourself, the remarks will remind you of your programming intentions and procedures if you later decide to revise it or reuse a program. Also, if you give your program to a friend, the remarks will explain what you have done.

Figure 7.1 shows how remarks can be used to clarify the box-drawing program. Two remarks that contain rows of asterisks set off

the title of the program. Blank remarks in lines 8, 24, and 34 are separate the program into its major parts, and remarks in lines 9, 25, and 35 name the parts. Line 19 explains the color code in the HCOLOR statement that follows, so that someone reading the program doesn't have to refer to the code list. You may not want to include so many remarks in your own programs, but you should use at least a few.

DEBUGGING

No matter how carefully you write your programs, you will have troubles from time to time. Your program won't work exactly as planned, or you will get an error message when you run it. This is known as a *bug* in your program, and you are faced with the task of *debugging*.

What do you do when you get an error message? Suppose you are trying to run the box-drawing program, but the computer stops with the message

```
?Illegal Quantity Error In 50
```

The computer got part of the way through the program, but couldn't complete statement 50 for some reason. The error message suggests the problem: the computer encountered an "Illegal Quantity"—a number too large for the way it is used in the command; for example, a coordinate too large for the graphics screen.

```
 4 REM  ********************
 5 REM  Box-Drawing Program
 6 REM  ********************
 8 REM
 9 REM  Hi-Res Graphics
10 HGR
19 REM  Choose Orange
20 HCOLOR = 8
24 REM
25 REM  Draw Top of Box
30 HPLOT 20,20 TO 140,20
34 REM
35 REM  Draw Other 3 Sides
40 HPLOT TO 140,140
50 HPLOT TO 20,140
60 HPLOT TO 20,20
```

Figure 7.1: *Use remarks to make your programs more readable.*

Now that you know what the error is, LIST the program and check it over. In this case, the error is probably in line 50, but you should look at the other statements as well. Often a mistake in an early line will cause problems in a perfectly good statement later in the program.

Check the program, line by line. Pay particular attention to details; a missing comma or a misspelled word can block an entire program.

Make sure that no lines are missing. It is easy to leave out a command as you are typing, or to erase one by typing the wrong statement number. Make sure also that no unwanted lines have slipped into your program. For example, you might have forgotten to give a NEW command before you started typing.

If you still can't find your mistake, have a friend check your program. Often another person will be able to spot a small mistake that you missed. Try explaining your program to someone else. In doing so, you may uncover a gap in your reasoning.

After you have checked and rechecked, you may be tempted to blame the computer. Don't—unless it starts behaving erratically on a variety of programs, or on programs that you've already tested. If your machine is defective, the problems will usually be obvious and will affect the computer's overall functioning, rather than just a single program. Check with your dealer if you feel that your machine has gone bad.

Have courage. It can be very difficult to debug a program, so you mustn't become frustrated. Often when you do find the bug, it will seem like a silly mistake, but you shouldn't blame yourself for having overlooked it. This happens to everyone.

CASSETTE STORAGE

You can store only one program at a time in your computer's memory. If you wish to use a different program, you must erase the first by typing

```
new
```

Your program will also be erased when you turn the power off.

If you write a program that you want to use over and over, you will want a more permanent form of storage. With a few simple commands, you can store programs with the Adam's built-in cassette

drive, described in Chapter 4. You can save your program on a tape, turn the computer off, and load the program back in whenever you want to use it again. In this way, you can keep a complicated program permanently available.

Read Chapter 4 of this book, if you haven't already done so. That chapter presents the background information you should have before you use your Adam's cassettes. It also includes some essential precautions about handling cassettes.

To save a program, you will need a blank cassette. You cannot use just any regular blank cassette, though, since the Adam's cassette drive needs a tape that has been coded with special magnetic markings. One blank "digital data pack" comes with your Adam, and you can buy others from your dealer. *Don't use your BASIC tape to store programs!*

You can save many different programs on a single cassette. The computer will keep track of all the files stored on the cassette, so that you can later call a particular program back when you need it.

When you save a program, you must always give it a *file name,* so that the computer can locate it when you want to retrieve it. As described in Chapter 4, you can choose any name of up to 10 characters, as long as it begins with a letter. Capital letters are significant in file names: name, Name, and NAME are treated as different files. Don't type a file name longer than 10 letters—you might ruin your cassette.

As a reminder to yourself, choose a name that explains the file's contents. Use distinctive names, so that you won't repeat them by accident.

Suppose you want to store the box-drawing program that we wrote earlier in this chapter. If you still have the program in your computer's memory, you can store it immediately. If you've turned the power off or done something else since you used the program, you'll need to retype it.

To store a program, type the command word SAVE and then the file name you want to give to the program. Let's choose the name 'box.' The SAVE command would then be:

```
save box
```

When you press the RETURN key, the cassette drive will spin for a time, then rewind and stop. The bracket prompt returns to the screen to show that your program has been stored and that you can go on

with other commands. With your program safely on tape, you can erase it from the computer's memory or turn the computer off.

When you later want to retrieve the program, use the LOAD command. Type

```
load box
```

and press RETURN. The recorder will spin and the computer will load the program back into its memory. When it has finished, the recorder will stop, and the computer will type the bracket prompt. You can now LIST the program to make sure it was loaded correctly, then you can RUN it again. The computer will draw the box, just as before.

When you load a program it replaces any program previously stored in the computer's memory. The computer clears the old program so that leftover lines won't interfere with the program it is loading. If you have been working on something important, you should save it before you load something else.

You can also load and run a program from a cassette in a single step. To do this, type a RUN command and add the name of the cassette file:

```
run box
```

The cassette will spin, and the program will start as soon as the tape has finished loading.

As with the Adam's word processor, you can look at a *directory* of all the files stored on a cassette tape. In BASIC, you do this with the CATALOG command. Simply type

```
catalog
```

and press RETURN. The computer will list the names of each of the programs and files on the tape. If the box-drawing program is the only thing you have stored on your blank cassette, the directory will look something like this:

```
Volume: FIRST DIR

A  1 box

251 Blocks Free
```

The exact appearance of this listing may vary with different cassettes, but the general scheme remains the same.

The first line of the catalog isn't too important. FIRST DIR is simply the name that Coleco gave to the blank cassette when it was prepared in the factory.

Beneath this opening line is the name of the file that you stored: box. The letter A at the left is merely a code for a stored BASIC program. The number 1 tells you how much space the program occupies on the tape. Coleco's cassettes are broken into *blocks,* each of which contains about a thousand characters of information. Since the box program is quite short, storage requires only one block. Beneath the directory, the computer shows you much space is left on the cassette. A blank cassette will usually start with 252 blocks free.

If you have many files stored on a cassette, the directory might fill up the entire screen and even roll some of the names off the top. You can stop the movement of the list by pressing CONTROL and the letter S. Proceed like this: Give the CATALOG command as usual. Then hold down the CONTROL key and press the letter S as the listing appears on the screen. The listing will stop so that you can read it. When you're ready to go on, press CONTROL-S again and the listing will continue to the end.

If you ask the computer to save a program with the same name as a file already on the cassette, it will keep the old file as a backup copy. That way, if you accidentally store a program with the wrong name or if the new file becomes damaged, you can go retrieve the backup version. To see how this works, type

```
save box
```

once again. Then give another CATALOG command. The new directory will read:

```
Volume: FIRST DIR

 a 1 box
 A 1 box

250 Blocks Free
```

There are now two programs on the cassette, both with the name 'box.'

The lowercase *a* in the left column identifies a backup file. The most recent file is always the one preceded by the capital *A,* and that is the version the computer retrieves when you type a LOAD command.

If you want to retrieve a backup file instead of the newest file, you must use the special RESTORE command. You type this new command in the same way as the LOAD command.


```
restore box
```

This command loads the backup version of the 'box' file and uses it to replace the most recent file on the tape. The most recent version is lost when you restore the backup.

You can also restore the backup file with the Adam's word processor. Pull the RESET COMPUTER switch and press ESCAPE/WP to return to the word processor. Then use the GET function to retrieve the file from the backup directory, as described on page 59 in Chapter 4. You can then store the file under another name and return to BASIC to use it. This procedure is a little more complicated, but it is safer since it does not delete any files from the tape.

If you have a very important program or a long one that would take hours to retype, you might want to keep another copy on a second tape. Put a new blank cassette in the drive and type another SAVE command. The computer will store the program on this second tape. If anything ever happens to your original tape, you can load the program from the second.

The Adam's BASIC also has two commands that let you RENAME and DELETE files on the tape. With the RENAME command, you simply give the file a new name, without changing its contents. You could, for example, type

```
rename box, newbox
```

If you now give a CATALOG command, the program formerly named 'box' will be listed under the name 'newbox'. That is the name you would use for the file in all future LOAD commands.

To erase a file from the tape, just type the DELETE command word and the name of the file. Suppose you want to erase the file which you just renamed 'newbox'. You would type

```
delete newbox
```

The file will be erased from the tape, and you can use the space for other storage. *Be careful when using the DELETE command!* Once you have erased a file from the tape, there is no way to recover it.

What can you do if one of your tapes goes bad and cannot be read? First check Chapter 4 of this book to make sure you're using the tape

correctly: it would be a shame to give up on a tape simply because you were doing something wrong.

If the tape is physically damaged, there's not much you can do. Even if you could repair it and make it readable, you could never be sure that the tape wouldn't fail again.

Sometimes, however, the tape itself is undamaged, but its file directory has somehow been erased. In this case, you can *reinitialize* the tape so that it can once again be used. This procedure will, however, erase the information stored on the tape, so you should reinitialize a tape only if you know you won't be able to use it any other way. If the reinitializing works, you will end up with a blank tape.

Before you do this procedure, give a CATALOG command. Does the computer display the cassette's file directory? If so, you can still use the tape, and you should not reinitialize it.

However, if the computer always responds with an error when you ask for the directory, you cannot hurt anything by reinitializing. Don't run this procedure on a valuable program by mistake: make sure you are willing to erase the tape. Then type a command such as this:

```
init Tapename
```

'Tapename' can be any name you want to give to the tape, up to 10 letters long. When you press RETURN, the computer will write a new, empty directory on the cassette, erasing the directory that you were using. All references to files on the tape will be erased, so that you can once again use the tape for data storage.

If you have had a second cassette drive installed in your Adam, you can use it just like the first one. You simply refer to it by the name d2. You can save, load, and run programs on this drive by typing d2 after the file name:

```
save filename, d2
load filename, d2
```

and

```
run filename, d2
```

To see the directory of files on the second data pack, type d2 at the end of your CATALOG command:

```
catalog, d2
```


USING THE PRINTER

By storing your programs on cassette tapes, you can preserve them for future use. Sometimes, however, you also want a written record of your work. A printout is an added safeguard against cassette failures, and it offers a clear record of what you have done.

There are two ways you can use your Adam's printer to record your work. The simpler is to have the printer make an exact copy of everything that appears on the screen. To obtain this *screen dump,* hold down the CONTROL key and press the letter P. The printer will immediately reproduce every letter on the screen, as accurately as it can. You might want to have the computer list a program or display your results on the screen, then make a printed copy with this CONTROL-P key. This feature works only in the text mode: you cannot print graphics.

If you want to list a longer program, you can use the command PR#1, which tells the computer to display the results of all commands on the printer as well. Type the following symbols:

```
pr#1
```

Now type

```
list
```

The printer will type out an entire copy of your program. As it prints, the program will also be displayed on the screen.

Once you have given the PR#1 command, the printer will continue to print anything that would normally be displayed on the screen. If you type:

```
print "Hello"
```

The printer will type the word Hello on the page. Or try:

```
catalog
```

The printer will produce a listing of the cassette's file directory.

To undo the PR#1 command and have the printer stop mimicking the screen, type

```
pr#0
```

If you simply want a printed listing of a program, you can combine the three necessary commands on one line:

```
pr#1: list: pr#0
```

The computer will turn on the printer, print the program, and turn off the printer, all in one step.

OPTIONAL EXERCISES

1. Add a statement to the box-drawing program on page 106 so that the left edge of the box will be blue. You will have to choose a statement number between 50 and 60.
2. Add a line to the end of the same program that will draw a diagonal line from the upper-left corner of the square to the lower right.
3. The following program listing contains four mistakes; find the bugs.

```
10  PRNIT "Hello There"
20  HGR
30  COLOR = 1
40  HPLOT 10,734
50  HPLOT 0,0 TO 10
```

SUMMARY

In this chapter, you have learned how to assemble commands to form a program. You can store as many commands as you like in the computer's memory, and run them all as a group. You can use the LIST command to review or change the program you have stored.

Programs have many uses, as you have already seen. Since they can be revised and reused, programs allow you to undertake complicated operations without having to retype the commands each time. With the Adam's cassette recorder and printer, you can save your programs permanently and keep a written record of your work.

In the next two chapters of this book, you will learn new techniques to increase your control over your computer and discover its true power.

Chapter 8
Using Variables

In the last three chapters, you have been giving very specific instructions to your computer. You have told it to print a message or to plot a point at a certain pair of coordinates. To print a different message or plot a different point, you have had to type a whole new command.

By using *variables,* you can make your commands more flexible. Instead of typing the actual number that you want to use in each command, you can use a name that will stand for any number you choose.

This general name is called a variable, because it can take on different values in the course of a program. You start by storing one number under the name. Later, you can change the number and get a different result from your program. This makes it convenient to use your computer for repetitive calculations.

At the end of this chapter, you will learn how you can make variables represent letters as well as numbers, so that you can manipulate words too.

In this chapter, we will try out several variations on the program that draws a box on the screen. The idea will remain the same as the program we developed in Chapter 7; however, we will use variables to stand for the coordinates, so that we can change the shape and location of the box more easily.

STORING NUMBERS

In most of our commands, we have used one or more numbers to tell the computer what we wanted. To plot a point, for instance, we used a pair of numbers to refer to its coordinates.

So far, we have been using specific numbers whose values never change. When you type a number such as 5 or 46, the computer uses it directly. These are called *fixed numbers,* because nothing inside the program can change their values.

Variables are an alternative to fixed numbers. Instead of typing the exact value of the number that you want to use in a command, you can give the number a *variable name*—a general label that can stand in its place.

A variable is like a storage box in the computer's memory. You *store* a number in the box. From that point on, the computer will use the stored number in any statement where it finds the variable name. You can use the variable name in any place where you normally would have used the fixed value.

The advantage is that you can change the value of the variable merely by storing a different number in the box. When you do this, the first number is discarded and the second replaces it. After that, the variable will stand for the new number. You can change the value as often as you wish.

Let's say, just for fun, that you wanted to use 'Fred' as a variable name. You would start by giving it a value: the number 7, for instance. You simply type:

```
Fred = 7
```

When you press RETURN, your computer sets aside a place in its memory for the variable. It then stores the number 7 in that place, so that you can refer to it by the name you have given.

This statement is really not an equation, though it looks like one. The computer interprets the equal sign as a command to take the value on the *right* of the sign and store it under the variable name on the left. This is called an *assignment statement.*

You cannot turn the equation around, as you can in normal arithmetic. If you type

```
7 = Fred
```

you are saying, "Take the number Fred and store it under the name 7." This makes no sense to the computer, and you will get an error.

Now that you have stored the value, the name Fred represents the number 7. You can see this by asking the computer to print the value:

```
print Fred
```


When you do this, the computer will display the number 7. (If it displays 0 or something else, you have probably mistyped something: try assigning the value again.)

From now on, you can use the name Fred wherever you would normally use the number 7. You could plot the point 7,7 in the low-resolution graphics mode by typing

```
plot Fred,Fred
```

You could even store the number 7 in another variable named Jane by setting Jane equal to Fred:

```
Jane = Fred
```

Let's look a little more carefully at the command

```
print Fred
```

When you pressed RETURN, the computer displayed the value you had assigned to the name Fred. Notice that this command is very different from

```
print "Fred"
```

With the quotes, the computer reads "Fred" as a message to be displayed. It therefore types the word Fred, rather than the number 7. Without the quotes, the computer understands Fred to be the name of a variable and displays its stored value.

You can change the value of a variable merely by storing a different number under its name. To change Fred's value to 19, for example, you could type

```
Fred = 19
```

The old value (7) will be lost, and from now on the variable will represent 19. You can check this by giving the command

```
print Fred
```

Fred will continue to mean 19 until you give it another value or turn the computer off.

Until you store a number in a variable, the computer assumes that its value is 0. We have not assigned a value to the name Irving. If you give the command

```
print Irving
```

The computer will respond with the number 0, since you have not stored any other value.

So far, we have been using people's names, like Fred, Jane, and Irving. It is generally best, however, to use a name that describes what the variable represents. If you have a variable that counts the number of times you have done an operation, you might call it Counter. For graphics, you might want to call the coordinates of a point *x* and *y*, so that you can simply type

```
plot x,y
```

When plotting more than one point at a time, you can give your coordinates different names all starting with *x* and *y*. For example, if you were drawing a line from one point to another, you might call the starting coordinates *xstart, ystart* and the ending coordinates *xend, yend.*

As with command words, it doesn't matter whether you capitalize letters in the name: FRED, Fred, and fred are all the same to the Adam. As before, it is usually easiest to type all of your commands in lowercase letters, so that you don't have to keep pressing the SHIFT and LOCK keys.

When you LIST a program, the Adam displays all variable names in lowercase, even if you typed the name in capitals. This is rather confusing, since it changes other items into capitals when it lists a program. This can make program listings somewhat confusing and unsightly, and it differs from the system used with other machines, in which all words are capitalized. For consistency, the program listings in this book follow the Adam's conventions for capital and lowercase letters, but you should expect to see words capitalized when you read other books and magazines.

There are a few restrictions on the names you can choose for your variables. The name can be as long as you wish, but it must begin with a letter. The rest of the name may contain numbers, but no special symbols. Finally, you may not start your variable name with a BASIC command word, such as PRINT or PLOT: the computer would have no way to tell the difference between your variable name and the command. Within these guidelines, you can use any name you like.

From these experiments with stored values, you have seen how to store a number in any variable you choose and then use the variable's name instead of the number. If you want to change the variable's value, you merely assign a new one. The variable will then represent the new value until you change it again.

VARIABLES IN PROGRAMS

When you're just typing simple commands, variables aren't all that useful. As long as you are typing each command as you go, you might as well use fixed numbers.

In programs, however, variables are helpful. You can write a general procedure for accomplishing a certain task and use variables rather than specific numbers. Then, when you run the program, you can plug in the specific numbers you want. To perform the same task with different numbers, you don't have to retype the program—you simply plug in new values.

To see how this works, let's return to the box-drawing program that we developed in Chapters 6 and 7. To start, give the command

```
new
```

Then type the program back into the computer's memory, exactly as shown:

```
  5  rem Box-Drawing Program
 10  hgr
 20  hcolor = 8
100  hplot 20,20 to 140,20
110  hplot to 140,140
120  hplot to 20,140
130  hplot to 20,20
```

This program is the same as the one shown in Figure 7.1, except that some of the remarks are omitted and the statement numbers have been changed in the second half to make room for some new statements between numbers 20 and 100.

In every version of this program that we have used, our box has always been the same size, and it has always been in the same location. We always started the box at the coordinates 20,20 near the upper-left corner of the screen, and drew each edge to a fixed point. Figure 8.1 shows the fixed coordinates of each corner of the box.

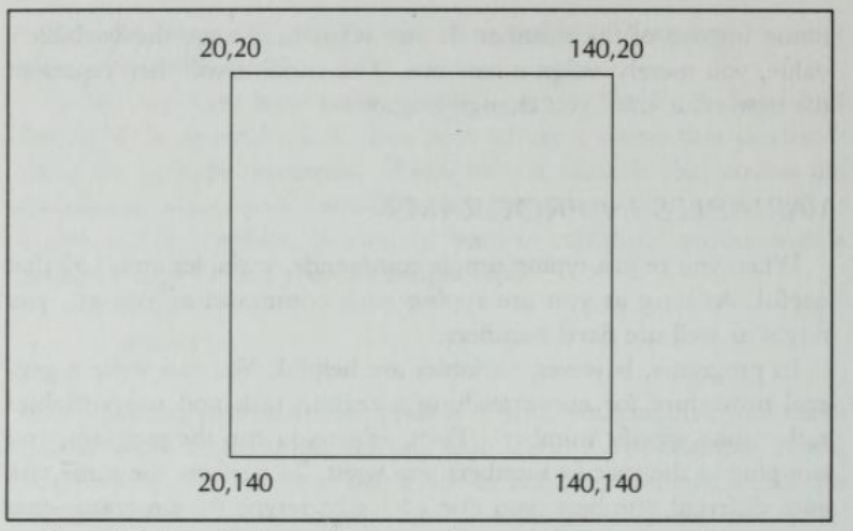


Figure 8.1: *Using fixed coordinates, we always produce the same box.*

We can now make our program more general so that we can draw a box of any size, shape, or location. Instead of using fixed coordinates for the corners, we can use variables.

Figure 8.2 shows the names we'll be using for each of the points. These names were chosen to explain the numbers they will represent. The variable xleft, for example, gives the x-coordinate of the left side of the box. Note that each of the variables is used in the names of two different corners: xleft is the x-coordinate of both the upper-left and lower-left corners.

Let's begin changing the program by replacing the fixed numbers with variables. Instead of giving another NEW command and retyping everything, just type the lines that you want to add. As you may recall, when you type an additional line while you have a program stored in the memory, the new line will be inserted into the program. If its statement number matches one already in the program, the old statement will be replaced. In this way we will be able to change certain parts of the program without retyping the whole thing.

There are times in this chapter when you will run a program that will leave the screen in the graphics mode. You can type commands in the text window at the bottom of the screen, but you will usually find it easier to return to the text mode. You can do this by typing

```
text
```

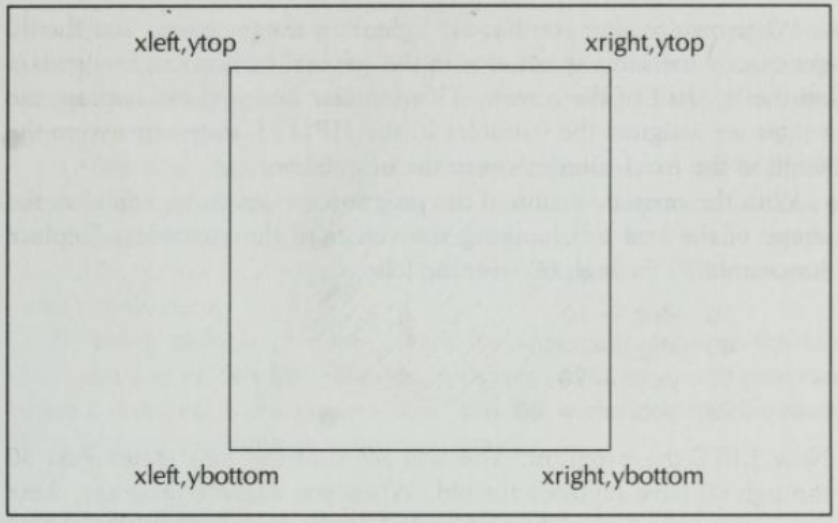



Figure 8.2: *We will give these variable names to the coordinates of the corners of the box.*

Before we can use the four variables to draw the box, we need to store their values. Let's start with the coordinates we've been using all along. Insert the following four lines into your program:

```
30  xleft = 20
40  xright = 140
50  ytop = 20
60  ybottom = 140
```

Now that we have the values stored, we can retype the HPLOT statements using the variables instead of fixed numbers. Type four lines as follows:

```
100  hplot xleft,ytop to xright,ytop
110  hplot to xright,ybottom
120  hplot to xleft,ybottom
130  hplot to xleft,ytop
```

These lines will replace the HPLOT statements in the original program.

LIST the program to make sure that all your new statements have been stored correctly. Check particularly the lines you have just added, to make sure they are accurate. The program should exactly match the listing shown in Figure 8.3.

When you're sure you have it right, run the program. You should get exactly the same result as with the original version: an orange box on the left half of the screen. This familiar box appears because the values we assigned the variables in the HPLOT statements were the same as the fixed coordinates in the original version.

With the revised version of the program, however, we can alter the shape of the box by changing the values of the variables. Replace statements 30 through 60 with the following:

```
30  xleft = 10
40  xright = 250
50  ytop = 70
60  ybottom = 80
```

Now LIST the program. You will see that the new statements 30 through 60 have replaced the old. When you run the program, these new values produce a thin horizontal rectangle across the screen, as shown in Figure 8.4. By choosing other values for these four variables, you can produce rectangles of any dimensions. Type and run these two other combinations:

```
30  xleft = 130
40  xright = 140
50  ytop = 10
60  ybottom = 150
```

```
30  xleft = 0
40  xright = 255
50  ytop = 0
60  ybottom = 158
```

The values on the left will produce a thin vertical box, while the values at the right will draw a rectangle around the entire graphics screen.

Try out some values of your own in this program. You can assign any values to the variables that you want, as long as you don't exceed

```
 5 REM  Box-Drawing Program
10 HGR
20 HCOLOR = 8
30 xleft = 20
40 xright = 140
50 ytop = 20
60 ybottom = 140
100 HPLOT xleft,ytop TO xright,ytop
110 HPLOT TO xright,ybottom
120 HPLOT TO xleft,ybottom
130 HPLOT TO xleft,ytop
```

Figure 8.3: *The revised version of the box-drawing program.*

the limits on the x and y dimensions of the graphics screen: *x* must fall between 0 and 255, and *y* between 0 and 158. When you run the program, the computer will draw a box according to the dimensions you give.

Notice that the variable xleft occurs three times in the HPLOT lines of the program, yet you can use one new assignment statement to change its value in all three places. No matter how many times a variable occurs in a program, one new assignment will change its value throughout.

By using variables, we have made a real improvement on the earlier versions of this box-drawing program. Our new box program gives commands for a general task, and lets us change specific numbers without rewriting the entire program.

THE **READ** AND **DATA** STATEMENTS

When you have wanted to change the value of a variable you are using in a program, you have needed to go back and change the

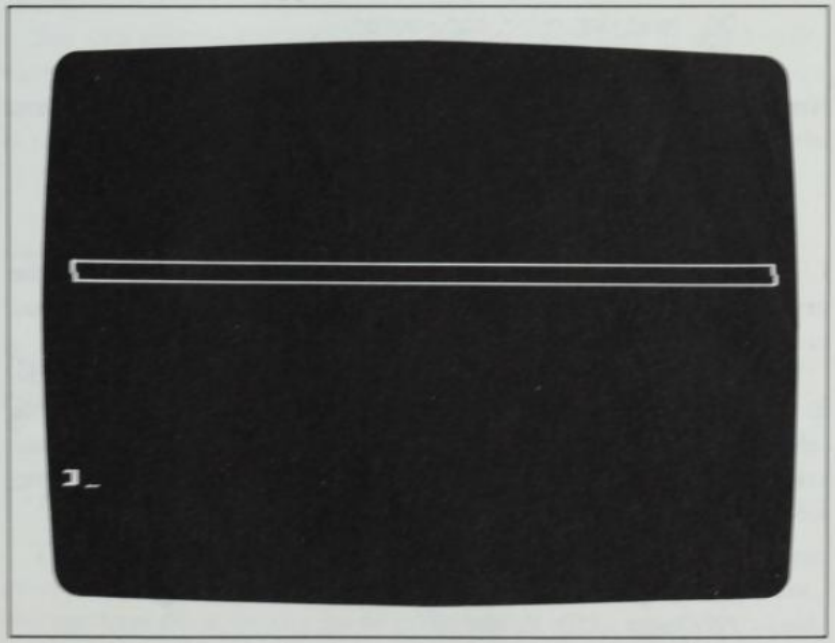


Figure 8.4: *Using variables, you can draw a box of any shape and size.*

statement where you stored it. This is all right if you rarely make changes, or if they only involve one variable. But you frequently need to change many values in a program, you can assign new values to several variables in a single statement. You do this with two new commands: READ and DATA.

The READ and DATA statements always work as a pair, and you cannot use one without the other. In the READ statement, you give the computer a list of variables to which you wish to assign values. You then type a DATA statement with a corresponding list of the values that you want to store. The computer takes the first value from the list in the DATA statement and assigns it to the first variable in the READ statement. Then it returns to the DATA statement and stores the second value in the second variable in the READ statement. This process continues until all the variables are filled. You must have at least enough values in the DATA statement to fill all the variables in the READ statement. If you don't, you will get an error message.

To see how this works, let's revise lines 20 through 60 of the box-writing program by using READ and DATA statements. Type the following lines:

```
30  read xleft,xright,ytop,ybottom
40  data 20,140,20,140
```

Now type two blank statement lines, to delete unwanted statements left over from the previous version:

```
50
60
```

LIST the program: it should match Figure 8.5. When you run the program, you will get the same results as before—a large orange box on the left half of the screen.

When the computer comes to the READ statement, it plugs the first number from the DATA statement (20) into the first variable (xleft). It then continues, with the other variables, reading 140 into xright, 20 into ytop, and 140 into ybottom. These two lines, therefore, have the same effect as the four lines we replaced:

```
30  xleft = 20
40  xright = 140
50  ytop = 20
60  ybottom = 140
```



```
 5 REM  Box-Drawing Program
10 HGR
20 HCOLOR = 8
30 READ xleft,xright,ytop,ybottom
40 DATA 20,140,20,140
100 HPLOT xleft,ytop TO xright,ytop
110 HPLOT TO xright,ybottom
120 HPLOT TO xleft,ybottom
130 HPLOT TO xleft,ytop
```

Figure 8.5: *The box-drawing program, using READ and DATA statements.*

The READ and DATA statements, however, are easier to type and to understand.

The great advantage of READ and DATA statements is that you can easily assign new values. If you want to draw a thin horizontal box, as in Figure 8.4, you need to type only one new DATA statement:

```
40  data 10,250,70,80
```

When the computer draws the box, the corners are set by the new values.

You can experiment with many different numbers in this DATA statement. In the previous version, you had to replace four lines, but here you need to retype only the one DATA statement. Try these sets of values:

```
40  data 130,140,10,150
```

and

```
40  data 0,255,0,158
```

You should get yellow rectangles of other shapes, just as you did on page 124 . Try some other values of your own.

THE INPUT STATEMENTS

There is a third way to assign numbers to variables, and it is even more flexible than the READ statement. With an INPUT statement, you can have the computer pause in the middle of the program and ask you to type a value on the keyboard. This frees you from having to

specify the numbers as you are writing the program. Instead, you can supply the values when you run the program. You can run the same program with different values merely by giving different responses.

To have the computer pause to ask you for the value of the first variable, xleft, you could type this line into the box-drawing program:

```
30  input xleft
```

When you ask the computer to run the program, it will stop when it comes to this statement and put a question mark on the screen. This is a *prompt* to show that the computer is waiting for you to supply some information.

You can add a message to your INPUT statements, so you will immediately know which piece of information the computer is waiting for. If you type a message in quotation marks before the variable name in the INPUT statement, the computer will display that message instead of the question mark when it asks for the value. You must separate the message from the variable with a semicolon (;), just as you separated the messages in the PRINT statements back in Chapter 6. I'd suggest you use a brief message such as this:

```
30  input "Left Edge? ";xleft
```

The space to the right of the question mark will visually separate the question from your response on the screen. If you run a program with this statement, the computer will pause and type the following message:

```
Left Edge?
```

This tells you which value the computer is expecting you to supply.

To see the INPUT statement in action, let's make yet another revision of our box-drawing program. Type the following lines, including the extra spaces after the question marks:

```
30  input "Left Edge?  ";   xleft
40  input "Right Edge? ";   xright
50  input "Top?        ";   ytop
60  input "Bottom?     ";   ybottom
```

The extra spaces will line up your answers in a neat column, so that the questions are easier to read. If you LIST the new version of the program, it should look like Figure 8.6. Note that the variables in this program are never given explicit values.

When you run this program, it will work a little differently from the other versions. The computer first shifts into the high-resolution graphics mode and chooses the color orange, following the instructions in lines 10 and 20. Then, at statement number 30, it prints the message

```
Left Edge?
```

in the text window and stops. This is your cue to type the first number—the coordinate of the left edge of the box (xleft). The computer will wait until you type a number and press RETURN. It will then ask for three more numbers to set the coordinates of the other three edges of the box. Give the same values that we used in the other examples:

```
Left Edge?   20
Right Edge?  140
Top?         20
Bottom?      140
```

When you're typing your values as you run the program, use the keyboard just as you normally would. You can correct mistakes with the BACKSPACE key, and type as many digits as you wish. Once you press the RETURN key, however, the computer stores your answer in its memory, and you cannot go back. If you type anything other than a number, the computer will not accept it and will ask you to reenter the number.

```
  5 REM  Box-Drawing Program
 10 HGR
 20 HCOLOR = 8
 30 INPUT "Left Edge?  ";xleft
 40 INPUT "Right Edge? ";xright
 50 INPUT "Top?        ";ytop
 60 INPUT "Bottom?     ";ybottom
100 HPLOT xleft,ytop TO xright,ytop
110 HPLOT TO xright,ybottom
120 HPLOT TO xleft,ybottom
130 HPLOT TO xleft,ytop
```

Figure 8.6: *The box-drawing program, using input from the keyboard.*

After you give your input for the fourth variable, the computer will draw the box. If you assign the values shown in the preceding example, you will get the same box as in Figure 8.1. By naming other values in response to your program's questions, you can have the computer draw any box.

You have accomplished two important things with these INPUT statements. First, you have developed a program that asks you questions as it is running. The computer waits for your responses, and then acts according to your wishes.

More important, though, you now have a program that you can run over and over again. Each time you use the program it performs the same general task, but the result depends on the values you supply. This is an important step: the computer is now using your program as a general problem-solver, rather than just a set of fixed instructions.

DOING CALCULATIONS

You have now used three different methods to store numbers in variables. When you need to assign a value to a single variable, you can use a simple equals sign. If you have several values that you want to store at one time, you might prefer to use the READ and DATA statements. Finally, if you want to supply the value when you're running the program, you can use an INPUT statement.

Up to this point, however, we have done very few things with the numbers once we stored them. We printed them and used them as coordinates in HPLOT commands, but nothing more.

You can also perform calculations on the numbers you store. Your computer can add, subtract, multiply, and divide, and can handle more complex mathematical functions as well. All the functions are built into the machine: you need only tell it which operations to perform. For arithmetic operations, the computer understands the following symbols:

```
+ Addition
- Subtraction
* Multiplication
/ Division
^ Exponent ("Power")
```


The asterisk (*) is used for multiplication, since the multiplication sign (×) might be mistaken for the letter *x*. For division, the slash (/) is used, because it resembles a fraction bar. All these symbols except the slash are located along the top row of the keyboard.

If you want the computer to add two numbers, type:

```
answer = 2+2
```

Your computer considers this line to be a command. It takes 'answer' to be a new variable and looks at the right side of the equals sign for the value it should assign. In this case, it sees 2 + 2, so it calculates the value: 4. Then, it stores this value for the variable on the left of the equal sign: the variable 'answer.' To see that it has indeed been stored, you can type the command

```
print answer
```

Your expressions can be as complicated as you wish, and can contain variable names as well as numbers. You could type any of the following lines:

```
x = 15+17
pay = wage*hours
area = length*height/2
```

The last of these examples contains two mathematical operations. The computer generally calculates from left to right, except that it gives priority to multiplication and division operations. For example, if you ask the computer to calculate

```
number = 3*4+6/2
```

The computer will perform the multiplication (3*4) and division (6/2) first. Then, it will add the answers together (12 plus 3) and store the result (15).

If you want to override this sequence, or if you simply find this confusing, you can group operations together with parentheses. You could, for instance, write the preceding command as follows:

```
number = (3*4)+(6/2)
```

You will get the same answer. Note that you would get a different answer if you grouped the operations differently:

```
number = ((3*4)+6)/2
```

In this case, the computer will first calculate the multiplication (3*4) in the innermost set of parentheses, then add the answer (12) to the number 6, to solve the outer set of grouping parentheses. Only then will it divide this result by 2 to get the final answer (9). Note that you can put parentheses within a set of parentheses.

If you find these concepts difficult, don't worry. I will stick to simple mathematical operations in this book, so that you won't get lost. If you are comfortable with mathematics, you can certainly learn to write more complicated formulas on your own.

You can use a mathematical expression anywhere you would use a number or a variable. You could, for instance, type

```
print 3*5
```

When you press RETURN, the computer will respond with the answer, 15. As you recall, this is very different from

```
print "3*5"
```

The quotation marks tell the computer to treat 3*5 as a message to be printed as is, rather than a formula to be calculated.

As an example, let's write a program that draws a blue triangle on the screen. (You're getting tired of boxes, aren't you?) Our goal will be to write a program that will draw a triangle of a fixed size, then use a simple calculation to relocate the picture at any point on the screen.

As always, type

```
new
```

before you start. Then type the program shown in Figure 8.7. When you run this program, the computer will ask you for starting x- and y-coordinates. The numbers you give will define the point in the upper-left corner of the triangle. Line 50 plots this point. The rest of the program draws lines to the two other corners, and back to the starting point. The result is a triangle, as shown in Figure 8.8.

Run the program again, and give different values for the coordinates. In this program, you can use any *x* up to 175, and any *y* up to 98. (If you use larger numbers, some of the calculations will yield coordinates too large for the HPLOT statements.) As you try different combinations, you will find that the triangle always remains the same


```
 10 HGR
 20 HCOLOR = 6: REM Blue
 30 INPUT "Starting X? ";x
 40 INPUT "Starting Y? ";y
 50 HPLOT x,y
 60 REM  Move 80 Units to Right
 70 HPLOT TO x+80,y
 80 REM  Move Down and to Left
 90 HPLOT TO x+40,y+60
100 REM  Move Back to Start
110 HPLOT TO x,y
```

Figure 8.7: *A program to draw a triangle.*

size and shape, but is moved to different positions on the graphics screen. This happens because all the coordinates are calculated relative to the point in the upper-left corner of the triangle. When you change that point, the others will change with it.

STRINGS: STORING LETTERS

You can store words and letters in the computer's memory, just as you can store numbers. To do this, you need a special kind of variable, called a *string variable.* This complex subject could easily take up a chapter of its own. I cannot give it that much attention, but this brief introduction will help you understand the concept when you come across it in other books.

To set up a string variable, you must put a dollar sign ($) at the end of the variable's name. These are all string variables:

```
a$
name$
address$
message$
```

You can store words or letters in the string variable, just as you store numbers in regular variables. You can use a standard assignment statement, by enclosing the message in quotes:

```
10  name$ = "Arthur"
```

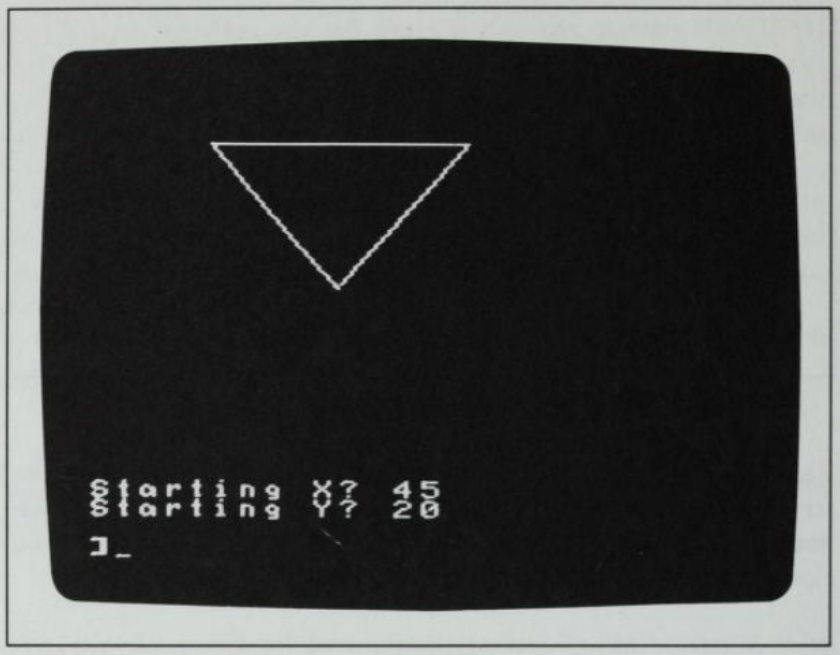


Figure 8.8: *The results of the triangle program.*

You can use READ and DATA statements:

```
20  read name$
30  data Arthur
```

Or, you can use an INPUT statement:

```
40  input "Which Name? "; name$
```

In the last case, the computer will ask 'Which Name?' and you can type

```
Arthur
```

from the keyboard.

When you have stored the message in the string variable, you can use it in several ways. You can print it:

```
100  print name$
```

You can store it in another variable:

```
110  copy$ = name$
```


You can even join it to another variable or to itself, by using a plus sign:

```
120  print name$ + copy$
```

When a plus sign links two string variables like this, it does not mean addition. Instead, it tells the computer to run the contents of the two string variables together one after the other. When you run a program that includes statements 110 and 120, the computer will print the name Arthur twice, like this:

```
ArthurArthur
```

BASIC gives you a number of *string functions* that let you slice a string variable's value into sections. If you just want the first three letters of the variable name$, you can use the LEFT$ function:

```
200  print left$(name$,3)
```

You must always put two elements within the parentheses following LEFT$: the string variable you want to slice, and the number of letters, starting from the left, that you want to include. When you run this statement, the result will be:

```
Art
```

Likewise, you can ask for the last three letters of the characters strored in name$, by typing

```
210  print right$(name$,3)
```

Finally, you can ask for a series of letters from the middle of the string. For this, you must supply two numbers. The first names the starting character, counting from the left. The second gives the number of characters to be displayed. You might type, for example:

```
220  print mid$(name$,2,3)
```

With this, you are asking the computer to begin with the second letter of the variable name$ and display three characters. When you run the program, the computer will display:

```
rth
```

You can do many things with strings. You can splice strings together, or pull them apart. You can use them to display messages in special ways on the screen, or store them for future use. The applications of string variables are many, but they are a subject for another book.

OPTIONAL EXERCISES

1. In the version of the box-drawing program shown in Figure 8.6, add an INPUT statement before line 20, so that you can choose a color other than orange. You will need to define a new variable, and will need to modify statement 20.
2. Add statements to the end of the triangle program (Figure 8.7) so that the computer will draw a Star of David. You will need to add HPLOT statements to draw a second triangle on top of the first, with the point facing upward.
3. Store the number 10 in the variable *j*. Then type the following command:

   ```
   j = j+1
   ```

   What value will *j* now have? Check your answer by asking the computer to

   ```
   print j
   ```

SUMMARY

In this chapter, you have learned how to use variables so that you can store a number and reuse it later. Variables let you simplify programs, and allow you to write programs that perform general operations.

The simplest way to store a number is to set the variable name equal to it. The number will remain stored in that variable until you replace it or turn the computer off.

If you have many numbers to store, or constantly need to change the values you are using, you can read the values from a DATA statement. Each variable in the READ statement is assigned the corresponding value in the DATA statement. To change the values, you need only change the DATA statement.

The most flexible way to store values is with an INPUT statement. The computer will stop each time it runs the program and ask you to type the value at the keyboard. When you press RETURN, the computer stores the value and continues with the program.

You have also seen how your computer can perform calculations. The arithmetic operations can be simple or complex, and you can easily print the results.

With string variables, you can also store lettering and words. These let you manipulate messages and print them out in various ways.

We have almost reached the end of our exploration of the Adam's BASIC programming language. In the final chapter, you will learn other commands that make your computer even more flexible.

Chapter 9

Controlling Your Program

In the last two chapters, you began to write programs. You can now type commands into the computer and store them in its memory as an organized program. With variables, you can make your programs more flexible, so that they will perform a general task using the values you supply.

Until now, however, you have been locked into the start-to-finish flow of the program. The computer began to run the program with the first statement, then went through each of the others in order. When it reached the end of the program, the computer stopped and returned control to the keyboard. It saw each statement only once.

In this chapter, you will learn ways to alter the flow of the program. You can ask the computer to jump from one part of the program to another, or to return to statements it has already used. You can also have the computer do a tedious task many times, or make decisions based on information you supply. Finally, you will learn how to break your program up into smaller *subroutines,* which you can use as building blocks.

In the programs throughout this chapter, the command words are capitalized, just as the computer lists them on the screen. Variable names and some text messages are still printed in lowercase, since the Adam does not capitalize them. You can continue to type everything in lowercase, but you should get into the habit of seeing commands in full capitals, since that is the way they are usually printed in books and magazines.

GOTO: *JUMPING TO ANOTHER STATEMENT*

The simplest form of program control is the GOTO statement, which tells the computer to jump to another statement in the program and continue from that point. You might, for example, want the computer to jump straight from statement number 20 to 50, skipping any statements in between. You could merely make line 20 a GOTO statement:

```
20  GOTO 50
```

When the computer sees this line, it will immediately skip to line 50 and pick up from there. Any lines that fall between 20 and 50 will not be used, unless another GOTO statement later on sends the computer back to them.

The most common use of GOTO is to send the computer back to the beginning of a group of statements. Try the following program:

```
10  PRINT "Hello"
20  GOTO 10
```

When you run this, your computer will display a vertical column of Hellos along the left side of the screen.

The computer first encounters line 10, which tells it to display a single word, Hello. The computer then moves on to line 20. That, however, sends it right back to line 10, so it prints the word again. When it has finished line 10 for the second time, the computer once more proceeds to the next statement. That is line 20 again, which, of course, sends it back to 10. The process goes on and on.

The computer doesn't even stop when it runs out of lines on the screen. If you look carefully at the Hello on the bottom line, you will see that it is flickering. The computer is still printing the word Hello repeatedly, and it keeps moving all the other lines up and off the screen to make room for it.

If you try to type a letter on the keyboard at this point, the computer will not accept it. The computer is still running your program, and all its attention is directed toward that task. Until the computer finishes a program and displays the bracket prompt, you cannot type any new commands.

This program, however, will never end—it is an *infinite loop*. The computer will continue repeating the task until you turn the power off or do something else to stop it. You could stop the program by pulling the RESET COMPUTER switch, but then you would need to reload the BASIC tape and start all over.

CONTROL-C is the best way to stop a program while it is running. Hold down the CONTROL key along the left side of the keyboard, then press the letter C. The computer will stop in its tracks and display:

```
?Break In 10
```

The bracket prompt and the cursor will reappear to show that you can type letters again.

If you wish, you can restart the program from wherever you stopped it, by typing the command

```
CONT
```

In most cases, though, you will probably want to use a RUN command to start over from the beginning.

You can use CONTROL-C to stop almost any program. Use it if you get stuck in an infinite loop, or if you simply want to cut the program short. You may need to follow the CONTROL-C with RETURN, if the program happens to be waiting at an INPUT statement. (If CONTROL-C doesn't work, you can always pull the RESET COMPUTER switch, reload the BASIC tape, and start again.)

Infinite loops are not necessarily bad. When you want to do a task over and over, it is often easiest simply to write the program as an infinite loop and stop it manually when you have had enough.

As an example, let's write a program that will take any number you type in, multiply it by 2, and print the result. Type the following program:

```
10  PRINT
20  INPUT "Type a Number -- "; n
30  PRINT n;" Times 2 = ";n*2
40  GOTO 10
```

Line 10 has the computer print a blank line on the screen, to make the display more readable. Line 20 is an input statement that asks you to type in a number from the keyboard. The computer waits for your response, then puts your number in the variable *n*. Line 30 displays first the number *n*, then the phrase "Times 2 = ", and then the

answer that it calculates from, $n*2$. The GOTO statement sends the computer back to the beginning of the program to ask you for another number.

The PRINT statement in line 30 may look a little unfamiliar. Up till now, we have used only one variable or one message after the word PRINT. As you may recall from Chapter 6, however, you can use several messages in a single PRINT statement. Here we have three: the variable n, the message " Times 2 = ", and the answer $n*2$. The computer understands this perfectly well, and will display the three items one after the other. The semicolons (;) separating the items are important, since they tell the computer to display the three items on the same line with no extra spaces in between.

When you run this program, the computer will start by asking you to

```
Type a Number --
```

Type a 3 and press RETURN. The computer will respond with the answer:

```
3 Times 2 = 6
```

It will then ask you to type another number. You can type numbers and have the computer double them as many times as you wish. When you're done, press CONTROL-C and RETURN to stop the program.

You can make any program repeat in this way. A good example is the box-drawing program we developed in Chapter 8. Think about what would happen if you added the following line to the program in Figure 8.6:

```
140  GOTO 30
```

The new program, shown in Figure 9.1, would run just like the old one, asking you to supply the coordinates for the corners of the box. Once you type in four numbers, the computer will immediately draw an orange box.

With the added GOTO statement, however, the computer won't stop when it finishes the first box. Instead, it will ask you for a new set of four numbers. If you type four more coordinates for the corners, the computer will draw a second box without erasing the first. You can draw as many boxes as you like. Figure 9.2 shows what your screen might look like after a few times through the loop.


```
  5 REM  Box-Drawing Program
 10 HGR
 20 HCOLOR = 8
 30 INPUT "Left Edge?  ";xleft
 40 INPUT "Right Edge? ";xright
 50 INPUT "Top?        ";ytop
 60 INPUT "Bottom?     ";ybottom
100 HPLOT xleft,ytop TO xright,ytop
110 HPLOT TO xright,ybottom
120 HPLOT TO xleft,ybottom
130 HPLOT TO xleft,ytop
140 GOTO 30
```

Figure 9.1: *By adding a GOTO statement to the end, you can have the box-drawing program repeat itself.*

If you look carefully at the program listing in Figure 9.1, you will notice that the added GOTO statement does not send the computer all the way back to the beginning of the program. We do not want to repeat the HGR and HCOLOR statements in lines 10 and 20 because the HGR statement would clear the screen each time around and erase the picture we are building. By jumping to statement 30, we can repeat only those lines that are necessary.

The GOTO statement is so convenient that many people overuse it. Programs that jump back and forth too much can be very difficult to follow, so you should use this statement only when there is a clear need. People normally expect to read a program from top to bottom, and too many departures from that order can be confusing.

FOR/NEXT *LOOPS*

With the GOTO statement, you can make the computer repeat a part of your program, but your loops must be infinite: the computer will repeat the program until you press CONTROL-C to make it stop.

Very often, however, you want to have the computer repeat a group of statements a fixed number of times, then go on to another part of the program. To do this, you must use two new statements: FOR and NEXT.

FOR and NEXT statements always come in pairs. The FOR statement precedes the set of lines you want to repeat; the NEXT statement comes at the end. When the computer comes to the FOR statement,

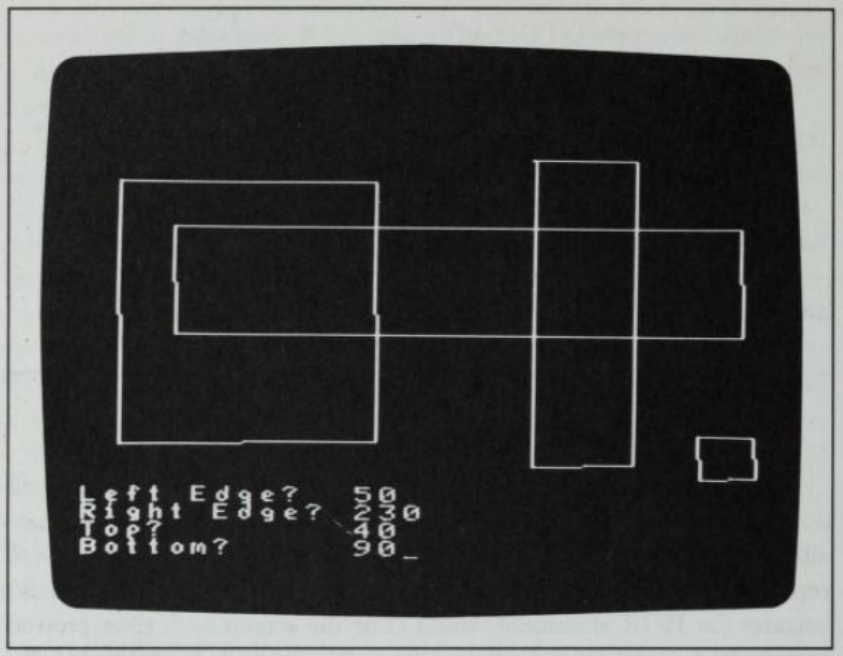


Figure 9.2: *The modified box-drawing program lets you draw more than one box on the screen.*

it starts up a counter to keep track of the number of repetitions it has performed. It then keeps repeating all the statements between the FOR and the NEXT until it reaches the limit you have set. Then, when it again reaches the NEXT, it passes through and continues with the rest of the program.

To see this in action, type the following program:

```
10 PRINT "The Loop Is Beginning."
20 FOR cntr = 1 TO 10
30 PRINT "Repetition No. ";cntr
40 NEXT cntr
50 PRINT "The Loop Is Finished."
```

The messages in this program's PRINT statements are arbitrary—chosen for the purposes of explanation. The counter variable is given the name 'cntr.' This is the variable with which the computer keeps track of the repetitions. When you run the program, your results should look like Figure 9.3.

The FOR statement in line 20 tells the computer to repeat the loop a number of times, keeping track with the counter variable named

cntr. The '1 TO 10' in the FOR statement sets the starting and ending values of the counter, and therefore the number of repetitions. As you can see from the results of your program, the counter variable starts at 1 on the first pass through the loop, and increases by one each time through. When it reaches the upper limit (10), the counter tells the computer that the loop is done, and the computer proceeds to the statement following the NEXT.

Line 30 shows a useful feature of FOR/NEXT loops:

```
30 PRINT "Repetition No. ";cntr
```

Within the loop, you can use the counter as a normal variable. Each time through, cntr will contain a different value, depending on the number of times the computer has repeated the loop. With careful planning, you can often use this to your advantage.

The counter variable does not need to start from 1. You can choose any numbers you want for the starting and ending values—even negative numbers. Try some other values in line 20 of this program, such as the following:

```
20 FOR cntr = 7 TO 14
20 FOR cntr = 329 TO 364
20 FOR cntr = -3 TO 6
20 FOR cntr = 8 TO 5
```

Note that in the last of these examples, the upper limit on the counter is less than the starting value. When this happens, the computer goes

```
The Loop is Beginning.
Repetition No. 1
Repetition No. 2
Repetition No. 3
Repetition No. 4
Repetition No. 5
Repetition No. 6
Repetition No. 7
Repetition No. 8
Repetition No. 9
Repetition No. 10
The Loop is Finished.
```

Figure 9.3: *An example of how a FOR/NEXT loop can be used to repeat a PRINT statement.*

through the loop only once, then continues past the NEXT statement.

You can also ask the computer to increase the counter each time by some number other than 1. You do this by adding a STEP number to the end of the FOR statement, like this:

```
20  FOR cntr = 1 TO 10 STEP 2
```

If you run the program with this statement, you will find that the counter increases by 2 each time: 1, 3, 5, 7, 9. Note that the counter's value is never exactly 10 in this case, but it stops the loop at 9, just before it goes over the limit.

You can also use the step to make the loop counter decrease instead of increase. To do this, give the step a negative value:

```
20  FOR cntr = 10 TO 1 STEP -1
```

Now the counter will start with the value 10. When it reaches the end of the loop, it will add −1 to the counter, effectively subtracting one from it. The loop will then repeat with the values 9, 8, and so forth. When it reaches 1, the loop will end and the program will continue.

You can give the counter variable any name you want. Make sure, however, that the counter's name in the FOR statement exactly matches the name you use in the NEXT statement to end the loop. If it doesn't, you get an error. Many people choose to give simple names to their FOR/NEXT counters—especially one-letter names such as *i*, *j*, and *k*.

You can have any group of statements within your loop—even another loop. Try out the following program:

```
10  PRINT "The Loops Are Beginning"
20  FOR i = 1 TO 3
30  PRINT " Outer Loop -- ";i
40  FOR j = 1 TO 4
50  PRINT " Inner Loop -- ";j
60  NEXT j
70  PRINT " Inner Loop Is Finished"
80  NEXT i
90  PRINT "Both Loops Are Finished"
```

Spaces were added to the messages in lines 30, 50, and 70 so that the screen display will reflect the structure of the program when you run it.

The results are shown in Figure 9.4. As you can see, the outer loop is repeated three times in the course of the program, each time printing the value of its counter variable, *i*. Each time through this outer loop, however, the computer encounters the inner loop, with a different counter variable, *j*. This tells it to repeat the print statement in line 50 four times on each of its three passes through the outer loop.

There is only one restriction on the use of a loop within a loop: the inner loop must be entirely contained within the outer. You must make very sure that the NEXT of the inner loop comes before the NEXT of the outer loop, as in the above program. Otherwise you will get an error.

FOR/NEXT loops are useful for almost every type of program. In most programs you will need to repeat statements, and for many tasks repetition is essential. Here are some examples of ways you can use FOR/NEXT loops:

Calculations Many computations have to be done over and over again. Often, you need to have a whole list of answers to the same simple problem, and you'd rather not have to calculate it each time.

Suppose, for instance, that you wanted to buy a number of gadgets

```
The Loops Are Beginning.
 Outer Loop -- 1
  Inner Loop -- 1
  Inner Loop -- 2
  Inner Loop -- 3
  Inner Loop -- 4
 Inner Loop is Finished.
 Outer Loop -- 2
  Inner Loop -- 1
  Inner Loop -- 2
  Inner Loop -- 3
  Inner Loop -- 4
 Inner Loop is Finished.
 Outer Loop -- 3
  Inner Loop -- 1
  Inner Loop -- 2
  Inner Loop -- 3
  Inner Loop -- 4
 Inner Loop is Finished.
Both Loops Are Finished.
```

Figure 9.4: *A loop within a loop can give interesting results.*

at $11.39 apiece. You'd like to buy several, but the number you buy will depend on the cost. So, you write a program to make up a table:

```
10 REM Table of Gadget Costs
20 price = 11.39
30 REM Print Headings
40 PRINT "Number","Cost"
50 PRINT
60 FOR i = 1 TO 20
70 cost = i*price
80 PRINT i,cost
90 NEXT i
```

When you run this program, you will get a table, as in Figure 9.5, that shows the cost for any number of gadgets between 1 and 20. Notice that in lines 40 and 80 a comma is used instead of a semicolon. As you may recall, a comma between messages in a PRINT statement tells the computer to arrange numbers neatly in columns.

Graphics For many types of pictures, you must use some controlled form of repetition, so that you don't have to plot every point and line

```
Number          Cost

1               11.39
2               22.78
3               34.17
4               45.56
5               56.95
6               68.34
7               79.73
8               91.12
9               102.51
10              113.9
11              125.29
12              136.68
13              148.07
14              159.46
15              170.85
16              182.24
17              193.63
18              205.02
19              216.41
20              227.8
```

Figure 9.5: *A FOR/NEXT loop will let you create a table.*

manually. You can use a FOR/NEXT loop for any figure that cannot be drawn with a few straight lines.

You will need to use a FOR/NEXT loop if you want to draw a solid figure. To see how this works, try the following program:

```
10 REM Solid Triangle
20 HGR
30 HCOLOR = 12
40 FOR y = 0 TO 158
50 HPLOT 0,y TO y,y
60 NEXT y
```

This program has the computer draw a line at every value of *y* from 0 to 158. At each *y*, it draws a horizontal line y units in from the left edge of the screen (coordinate 0,*y*). The result is a solid triangular region from the top of the screen to the bottom, as shown in Figure 9.6.

Delays Each time the computer has to repeat a FOR/NEXT loop, it pauses for a fraction of a second. In most programs, you will hardly

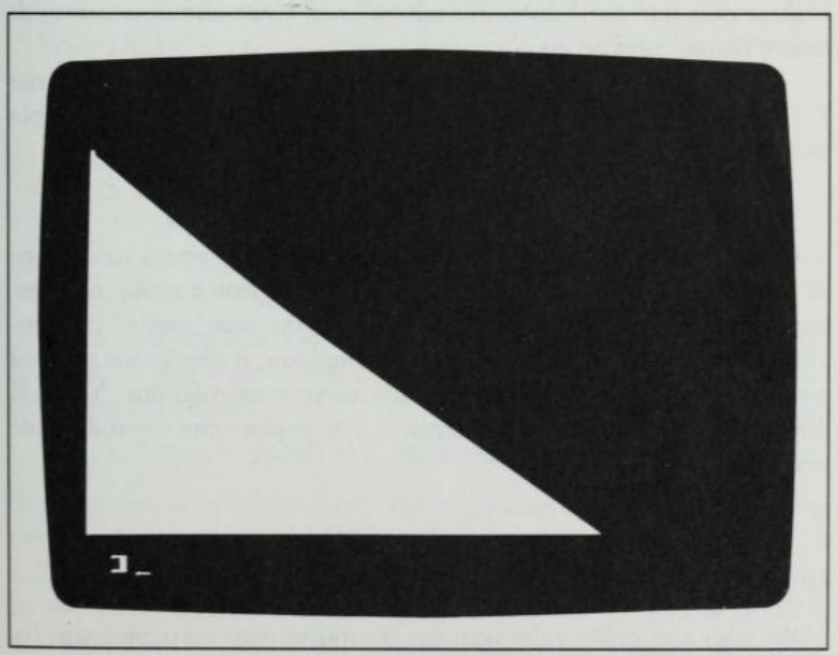


Figure 9.6: *You can draw solid regions on the screen, using FOR/NEXT loops.*

notice this delay. If you ask for a large number of repetitions, though, you can slow down the action by creating a noticeable pause.

Write a FOR/NEXT loop with many repetitions at the place where you want the pause:

```
20 FOR t = 1 TO 5000
30 NEXT t
```

This loop accomplishes nothing as it goes through its 5000 repetitions; however, the computer takes about five seconds to complete the loop. This pause might be helpful in a program such as this:

```
10 PRINT "Patience Test"
20 FOR t = 1 TO 5000
30 NEXT t
40 PRINT "You Passed!"
```

When you run this program, the computer immediately displays the words

```
Patience Test
```

It then goes into the "do-nothing loop" and waits about five seconds before congratulating you.

Delay loops are common in many BASIC programs. Since the FOR and NEXT statements must appear together, many people write their delay loops on a single line, separated by a colon:

```
20 FOR t = 1 TO 5000: NEXT t
```

In these and other uses, FOR/NEXT loops give you a new range of control over your computer. They let you repeat a group of statements any number of times, then go on to the next part of the program. While the computer is repeating the loop, it also keeps track of a counter variable, which you can use in your calculations. You will find more uses for these statements as you explore the capabilities of your computer.

IF STATEMENTS: MAKING DECISIONS

You can also control the way the computer runs your program by having it make decisions. With the IF statement, you can have the

machine decide whether or not to do something depending on the values of certain variables.

Of course, the computer can't make an independent decision. You have to tell it to do one thing if a certain condition is met, and another thing if it isn't. The computer then makes its choice in response to the information you give it.

The IF statement consists of two parts: a condition and a response. The general form of the IF statement is

```
10 IF condition THEN response
```

When you actually write this statement, you will replace the words *condition* and *response* with a specific comparison and command.

The condition will usually be a comparison between a variable and a fixed number. You could, for example, test to see if the variable *n* is equal to 0. You would start by writing

```
10 IF n = 0 THEN . . .
```

The computer would check the value stored in *n*. If it is exactly 0, the computer will carry out the response. If *n* is not 0, the computer will skip the response and proceed to the next statement in the program.

You can use a variety of arithmetic comparisons in the IF statement's condition. The most important are:

```
=   equal to
>   greater than
>=  greater than or equal to
<   less than
<=  less than or equal to
<>  not equal to
```

You can also combine two or more conditions with the words AND and OR. If you want to see if *n* is more than 5 and less than 10, you could use the condition

```
10 IF n > 5 AND n < 10 THEN . . .
```

The computer will carry out the response only if *n* has a value between 5 and 10.

The response, which follows the word THEN in the statement, can be any BASIC command. If the condition in the IF part is satisfied, the computer will follow the command; if the condition isn't satisfied, it will ignore the command.

Here are some examples of valid IF statements:

```
10  IF i = 10 THEN PRINT "Number Is Ten"
20  IF mode = 0 THEN HGR
30  IF size > 20 THEN size = 20
```

Line 10 will display a message on the screen if at that point the variable *i* is equal to ten. If *i* has any other value, no message is displayed. Line 20 uses an IF statement to switch the computer into high-resolution graphics, if it discovers the variable mode has been given the value 0.

Line 30 shows a common use of the IF statement. When it arrives at this statement, the computer checks to see if the variable 'size' exceeds 20. If it does, the computer resets it to 20. After this statement, you can be certain that the values of the variable 'size' will not exceed this maximum.

To see the IF statement in action, let's try a simple program that lets you guess at a number from 1 to 100. Type in this program:

```
10  answer = 57
20  PRINT
30  REM Blank Line
40  INPUT "Type a Number -- ";n
50  IF n < answer THEN PRINT "Too Small"
60  IF n > answer THEN PRINT "Too Large"
70  IF n = answer THEN PRINT "Correct"
80  GOTO 20
```

Line 10 of this program gives the correct answer to the computer. The rest of the program is an infinite loop that asks you to type a number, then compares it to the correct answer.

When you run this program, the computer will start by asking you to

```
Type a Number --
```

You know the answer (57), but try some other numbers too. Give your response and press RETURN. The computer will say either Too Small, Too Large, or Correct, depending on the number that you type. The computer makes its choice according to the IF statements in lines 50 through 70, then prints the appropriate response. Line 80 sends the computer back to the beginning of the loop, to ask for another number. Figure 9.7 shows the results of several guesses.

When you have finished, press CONTROL-C and RETURN to stop the program.

You can change the expected answer merely by changing line 10. You might even enjoy playing this as a game with a friend. Store a number in the computer by typing it into line 10, then clear the screen and run the program. You can then have a friend try to guess the number that you typed in.

IF statements are often used to ask the computer to GOTO another part of the program only under a certain condition. You could, for example, have the number-guessing program go back to line 20 only if the wrong answer is typed. Replace the simple GOTO statement in line 80 with the following IF statement:

```
80  IF n <> answer THEN GOTO 20
```

The <> is a "not equals" condition that will send the computer on two different paths through the program, depending on whether the values of the two variables are equal. If they are not equal, the computer will jump directly back to line 20 and continue from there. If the numbers are equal, however, the condition will be false, and the computer will skip the GOTO command. Normally, it would then continue with the next statement, but since statement 80 is the last line, the program simply ends.

An IF statement with a GOTO response provides you with a way to split your program into two parts, with the computer choosing its

```
Type a Number -- 35
Too Small

Type a Number -- 64
Too Big

Type a Number -- 51
Too Small

Type a Number -- 58
Too Big

Type a Number -- 57
Correct

Type a Number --
```

Figure 9.7: *Results of the number-guessing program.*

path depending on the IF statement's condition. You might, for example, write a program such as this:

```
10 PRINT "Type a Number -- "; n
20 IF n = 0 THEN GOTO 50
30 PRINT "Statement 30 is used"
40 GOTO 10
50 PRINT "Statement 50 is used"
60 GOTO 10
```

This program asks for a number, then tests to see if it is zero. If it is, the computer jumps to statement 50 and prints a message. If the number is not zero, the computer bypasses the response and continues with the statement that follows, number 30. In either case, the computer returns to line 10 after it prints the message.

The IF statement lets you use a technique called *data checking.* Often when you write a program that asks for input from the keyboard, you need a specific type of response. You might, for instance, expect a number from 1 to 12, to represent a month of the year. You can guard against meaningless responses by making a simple check:

```
10 INPUT "Which Month? "; month
20 IF month < 1 OR month > 12 THEN GOTO 10
30 PRINT "OK"
```

When you run this program, the computer will ask you to type a number to represent a month. If the value you type is less than 1 or greater than 12, however, the IF statement will send the computer back to line 10 to ask for another value. It will keep doing this until you type a number between 1 and 12. In any statements you might add after line 30, you can be confident that the variable has a value that represents an actual month.

IF statements clearly have many other uses. Whenever you want your computer to do more than a simple task, you will need to give it directions for making choices along the way. With the IF statement, you can easily control the way it makes these choices.

GOSUB: SUBROUTINES

There is another important way you can change the start-to-finish flow of your program. This is with a *subroutine* or *subprogram,* a series of

statements that you set off from the rest of your program and use as a unit whenever you want.

To use a subroutine, you must *call* it with a GOSUB statement, such as this:

```
20 GOSUB 1000
```

This GOSUB statement acts just like a GOTO statement, sending the computer to line number 1000 to continue its work. With GOSUB, however, you tell the computer to return to the program's next line after it has finished using the subroutine.

A subroutine is like any other group of statements, except that it ends with the command RETURN. When the computer gets to this final command, it sees that the subprogram is done, and goes back to the statement following the original GOSUB. In this way, you can tell the computer to go off and follow a group of instructions, then come back to where it left off.

To see how this works, let's try a sample program:

```
  10 PRINT "Main Program Begins"
  20 GOSUB 1000
  30 PRINT "Main Program Continues"
  40 END
1000 REM Sample Subroutine
1010 PRINT "Now Running Subroutine"
1020 RETURN
```

When you run this program, the computer will display the following statements:

```
Main Program Begins
Now Running Subroutine
Main Program Continues
```

The computer began with line 10, the beginning of the main program. At line 20, it jumped to the subroutine and printed the second message. At the RETURN statement, however, the computer finished the subroutine, returned to the main program, and continued with line 30, which immediately follows the GOSUB statement that called the subroutine.

The END command in line 40 is quite important. It signals the end of the main program and tells the computer to stop. If this statement were not there, the computer would continue running the

program, going on to line 1000 of the subroutine. Always use an END statement after the main program and before the subroutine unless you want to rerun the subroutine at the end of the main program.

Your program can have many different subroutines, as long as you end each subroutine with a RETURN and don't repeat any statement numbers. While you can choose any statement numbers for your subprograms, you will find it best to use rather high numbers, such as the 1000 chosen here, in case you later decide to expand your main program. Also, it is generally best to start each subroutine with a remark (REM statement) that describes what the subprogram does.

You can put any kind of statement in a subroutine. You can even use another GOSUB statement to call a second subroutine. If you do this, the RETURN statement from the second subprogram will send the computer back to the GOSUB statement in the first. From there the computer will complete the first subroutine, then return to finish the main program.

You may be tempted to use a GOTO rather than a RETURN to get back to the main program. Don't. Although there is a way to do this, you will usually end up confusing both yourself and your computer. Make your subroutines self-contained, and be careful to have them RETURN when they are done.

Why use subroutines? There are a variety of reasons why you might want to break up your program into smaller parts. Here are a few:

- *To use a group of statements more than once.* In large programs, you often have certain tasks that you need to do over and over. If you just want to repeat a group of commands several times in one place, you might be able to use a FOR/NEXT loop. If, however, you need to reuse the group in another part of your program, you would have to retype the commands each time. Instead, it would be easier to group the statements in a single subroutine. Then, each time you need to do the task, you can call it with a simple GOSUB statement.
- *To make a program modular.* Very long programs can seem terribly daunting if you try to view them all at once. By breaking the program down into more manageable units, you can make it much easier to understand. Some people write their main programs as a series of GOSUB statements, which direct the computer to each of the modules in turn.

- *To develop a subroutine library.* At some point, you might want to put together a collection of subroutines that you often use as building blocks for different kinds of programs. If you store all these routines on a cassette or diskette, you can load any one of them back into the computer whenever you want to use it in a program.

With subroutines, we have reached the end of our exploration of the Adam's BASIC language. With the statements you have learned in this chapter, you are well on your way toward programming your computer effectively.

OPTIONAL EXERCISES

1. Write a program that will ask you for a number, then say whether that number is positive, negative, or zero. You will need an INPUT and an IF statement. Remember that you use a minus sign to type a negative number.
2. Write a subroutine that draws a rectangle around the entire graphics screen. Add this subroutine to one of your other graphics programs so that you have a frame for your picture.
3. Write a FOR/NEXT loop to calculate the sum of the numbers between 1 and 100. Hint: set up a variable named 'sum' and add the counter variable to it each time through the loop.

SUMMARY

In this chapter, you have learned many new ways to control your machine. You are no longer restricted to writing programs that the computer simply marches through top-to-bottom. You can have it jump about, turn loops, make decisions, and run subprograms.

The GOTO statement tells the computer to jump to another statement. You can jump forward or backward within your program, and create infinite loops if you wish. This is often a simple way to do repetitive tasks.

With FOR and NEXT statements, you can make your computer repeat a group of statements a certain number of times, then go on to the rest of the program.

IF statements let your computer make decisions in the course of your program. At any point, you can tell the computer to test a condition. If the condition is true, your IF statement will have the computer take some action. It can print a message, change a value, or go to another part of the program—anything you choose.

The GOSUB statement gives you yet another way to determine the flow of your program. You can break a large program up into smaller blocks, then use the blocks whenever you need them. Subroutines let you simplify your programs and allow you to work with one piece at a time.

With this chapter, you have come to the end of the standard programming techniques covered in this book. You know how to write programs, use variables, and control the order in which the computer follows your commands. You have had a glimpse of your Adam's potential and are ready to explore on your own. The fun is just beginning.

Afterword

In the course of this book, you have done many things with your Adam. You learned how to set it up and how to use preprogrammed software. You learned to use the word processor to type documents. Then, in the third section, you began to give commands of your own and to write programs that control the computer in various ways.

These procedures are only the beginning, however. With these basic skills, you can go on to explore many other possibilities. You can connect optional equipment to your system or write complex programs that solve important problems.

In these final pages, I will try to give you an idea of some of these other possibilities. I cannot cover everything, though. If you want more detailed information, check some of the books in the bibliography, or try some experiments on your own.

ADD-ONS

Coleco is planning a large line of add-on devices to expand its Adam. Some of these may never materialize, but they are worth looking out for.

Two of the most important add-ons are mentioned at the end of Chapter 4. If all goes as planned, you will soon be able to have a second cassette drive installed in your Adam. Or, for greater speed and reliability, you will be able to buy a disk drive for your system. These optional storage devices will let you use a larger variety of software.

Another useful accessory is a *telephone modem,* a device that lets your computer communicate electronically with another computer. The modem converts your computer's electronic signals into a warbling

sound, then sends them directly over your telephone lines. With the help of a special program, your computer can then send information directly to another computer. It can also receive messages and store them in its memory, or communicate with one of the electronic news and information services.

You will also be able to buy a *memory expansion* for your Adam. This plug-in card will add another 64K bytes of memory to the 80K that comes with the computer. Make sure you really need the extra memory before you buy it, though. The Adam's original memory is quite enough for most people.

Further down the line, Coleco is planning a number of other options that will give access to software written for other computers. One expansion will allow you to use the CP/M operating system, described briefly in Chapter 1. Another planned expansion will let the Adam run software designed for the IBM Personal Computer. Such expansions would let you choose programs from the two largest lines of software currently available. Don't hold your breath, though: these expansions will be a long time coming, if they ever do move into production.

Although all these add-ons have been officially announced by Coleco, some may never be marketed. Check with your dealer for more information and for detailed descriptions.

OTHER DIRECTIONS

The chapters in the third section of this book describe some of the ways you can control your computer directly. You learned how to create a program, use variables to perform calculations, and control the program's flow with loops and decisions. With special graphics commands, you can paint various types of pictures on the screen.

While we have covered all the fundamental concepts of programming, there are many other things you can do. You can, for example, use *arrays* that let you store an entire table of numbers in the memory without giving each number an individual variable name. You can save words or numbers as *data* on cassette or diskette. You can then later retrieve this information and reuse it in other programs.

In principle, you can also use BASIC programs written by other people. Many books and magazines print BASIC programs that you can simply type into your computer and run directly.

Unfortunately, computer manufacturers do not all use the same version of the BASIC language. Unless a program is written specifically for the Adam, you cannot be certain that you can simply run it. Since the Adam is so new, few books and magazines offer programs specifically designed for it. For now, your choices may be limited.

However, the Adam's BASIC is virtually identical to the BASIC programming language of the Apple II computer. While there are a few differences, many Apple BASIC programs can be run on the Adam with only minor changes. Since the Apple II is one of the most popular computers, you can readily find BASIC programs written for it.

If you try to run Apple II programs, you will find that the minor differences between the two machines do cause some problems, especially for a novice at programming. For example, the Adam's BASIC will display only 31 characters on each line, compared with the Apple's 40. As a result, you may find the Adam's display to be almost unreadable for complex Apple programs. Other differences between the more technical aspects of the two computers can require complex changes in programs. The simpler programs, however, should work almost identically on the two machines.

There are several other programming languages that you can use with your Adam. One of the most popular is Coleco's *Logo,* a graphics language designed for children. In Logo, you give commands to move a small *turtle* around the screen. As it moves, the turtle lays down a trail, which creates a picture. Logo is an excellent instructional tool for introducing children to the basic principles of programming.

With any programming language, the computer must translate the command words into the specific codes that it can use. This process is slow and inefficient for complex programs that must run quickly. One can bypass the translation stage completely by writing programs in *machine language.*

With machine language, you are actually giving commands at the deepest level of the computer, and can exert direct control over its operation. Certain of the advanced graphics features of the Adam can only be controlled with direct machine language commands. Machine language also lets the computer run much more quickly, since inefficient operations are reduced.

Unfortunately, machine language is very technical and difficult to use, since it requires you to keep track of every single operation the computer must do. You must know a lot about the internal workings of

the machine and must be willing to do tedious programming. Even professional programmers usually avoid machine language unless there is a clear need for it.

Machine language is closely related to *assembly language,* which is slightly easier to use. While you must still give meticulous commands for every operation you want the computer to do, you can use labels and verbal commands, rather than numerical codes. The computer then translates assembly language directly into machine language.

Some books and magazines take a middle ground between BASIC and machine language by using two special BASIC statements: PEEK and POKE. These commands let you read and store values directly in the computer's memory without giving them a variable name. This gives you some direct control over the internal operation of the computer. The process is, however, quite cumbersome and hard to understand.

Whatever you do, I hope that you will continue your explorations. Many books and resources can help you learn more than I have been able to cover in these pages.

Your main resource, however, is yourself. Be creative with your computer; play around with it. There are many wonderful things you can do with your machine, whatever approach you choose to take.

Appendix A
Further Reading

Because the Adam is such a new machine, very few books have been written specifically about it. I have seen only one so far:

Word Processing with Your Coleco Adam

by Carole Alden
(SYBEX, 1984).

This book offers an in-depth guide to the Adam's built-in word processor and provides far greater detail than I could in this volume. It is well organized and clearly written, and it contains useful tips on how to get the best results from your Adam's word processor.

Although few books are devoted exclusively to the new Adam, there are many general resources. The Adam's BASIC programming language is an almost exact duplicate of the BASIC for the popular Apple II computer, for which hundreds of books are available. While you may need to make a few allowances for the differences between the Adam and the Apple II, you can use most of these books right off the shelf. Here are a few of the best:

Your First Apple II Program

by Rodnay Zaks
(SYBEX, 1983).

This fine introduction to fundamental techniques of BASIC programming focuses on the thought processes you need for writing a program. The delightful illustrations by Daniel Le Noury help to clarify the major concepts.

The Apple II BASIC Handbook

by Douglas Hergert
(SYBEX, 1983).

This is the best reference guide available for the Apple II BASIC language. Arranged as an alphabetical dictionary, this book gives detailed, readable descriptions of all Apple BASIC commands, as well as common computer words.

Apple II User Guide

by Lon Poole
(Osborne/McGraw-Hill, 1982).

This 400-page book gives every piece of information that you would ever want to know about the Apple II computer. This is an essential reference book for anyone who wants to do serious programming in BASIC on the Adam. It is not, however, light reading. The descriptions are fairly technical and assume some previous knowledge of computers.

Apple II BASIC Programs in Minutes

by Stanley R. Trost
(SYBEX, 1984).

This is a collection of short BASIC programs which you can type directly into your Adam and use, with no prior knowledge of BASIC programming. Since the programs were designed for the Apple II computer, some may require slight modifications to work on the Adam. The programs cover a range of financial, home management, and educational applications.

Appendix **B**

Reference Guide to BASIC

This reference list of the BASIC language summarizes the commands described in this book. For a complete reference guide, use the SmartBASIC manual included with your Adam, or buy Douglas Hergert's excellent book, *The Apple II BASIC Handbook.* The Adam's BASIC is virtually identical to the Apple II's language.

CATALOG

Displays the directory of all the files that are currently stored on the tape in the cassette drive. This command also shows how much space is remaining on the tape. Backup files are indicated by a lowercase *a* to the left.

If you have a second cassette drive, you can see its directory by typing

```
CATALOG, D2
```

COLOR = n

In the low-resolution graphics mode: chooses the color that will be used to draw in future PLOT, HLIN, and VLIN statements. The computer will continue to use that color until it is given another COLOR command. The Adam's low-resolution graphics colors are shown in Figure 6.3 on page 87.

CONT

Restarts a program that has been stopped with the CONTROL-C key. The program will continue from the statement it was using when it was stopped.

CONTROL keys

To use these special function keys, hold down the CONTROL key and press the appropriate letter on the keyboard.

CONTROL-C: Stops a program as it is running. This procedure is used to break out of an infinite loop or to stop a program before it has finished. You may also need to press the RETURN key before the program will stop.

CONTROL-P: Makes an exact printed copy of whatever is currently displayed on the screen. (Text mode only.)

CONTROL-S: Stops a listing that is being displayed on the screen. Most often used when a LIST or CATALOG command produces more than a full screen of displayed text. Press CONTROL-S again to restart the display.

DATA

Used with the READ statement to assign numbers to variables. The numbers in the list after the word DATA are assigned in sequence to the corresponding variables in the READ statement. If numbers remain in the DATA statement after the READ statement's variables have been filled, further READ statements will continue to use them, starting with the first unused number.

END

Marks the last statement in a program. An END statement is unnecessary if a program proceeds strictly from start to finish. If, however, subprograms or other statements follow the end of the main program, an END statement must follow the main program.

FOR/NEXT

The FOR statement marks the beginning of a loop, and tells the computer how many times to repeat the loop. The general form is:

FOR *i*=*a* TO *b* STEP *c*

The FOR statement must specify a counter variable (*i*), its starting value (*a*), and its ending value (*b*). The statement can also designate the step (*c*)—the increment by which the counter is to increase each time through the loop. If no step is designated, the computer assumes it is 1. FOR must always be paired with a NEXT statement, which marks the end of the loop.

GOSUB *n*

Asks the computer to run the subprogram that starts at statement number *n*. The computer will run the subprogram until it reaches the RETURN statement that marks the subprogram's end. The computer then resumes the main program with the statement following the GOSUB.

GOTO *n*

Tells the computer to jump directly to statement number *n* and continue from there.

GR

Clears the screen and shifts into low-resolution graphics. If the computer is already in low-resolution graphics, the mode will not change, but the graphics screen will be cleared. Each GR command must be followed by a COLOR command.

HCOLOR = n

In the high-resolution graphics mode: chooses the color that will be used to plot all points and lines in hplot statements. The codes for high-resolution colors are different from the codes for the low-resolution graphics mode: see Figure 6.3 on page 87 for the correct values. This color remains in effect until you give another HCOLOR command or reset the graphics mode with a GR or HGR command.

Don't confuse HCOLOR with COLOR, a low-resolution graphics command.

HGR

Clears the screen and chooses the high-resolution graphics mode. You must choose a color with the HCOLOR command each time you use HGR.

HLIN *xleft, xright* AT *y*

In the low-resolution graphics mode: draws a horizontal line across the screen, from column *xleft* to column *xright*. The entire line will be drawn on row *y*, counting down from the top of the screen, and will have the color chosen in the most recent COLOR statement. The coordinates must be within the allowable range for the graphics: both *x* and *y* must be between 0 and 39. Also, *xleft* and *xright* must be different.

HOME

Clears the screen in the text mode and places the cursor in the upper-left corner. Don't confuse the typed HOME command with the HOME key on the keyboard.

HPLOT

In the high-resolution graphics mode: plots a point or draws a line between two points, using the color selected by the most recent HCOLOR statement. To plot an individual point, give one set of coordinates:

HPLOT x,y

For a line between two points, use the word 'to':

HPLOT x_1,y_1 TO x_2,y_2

To continue a line on to other points, add more 'to'-points at the end of this statement or type another HPLOT statement:

HPLOT TO x_3,y_3

(See Chapter 6 for details.)

HTAB x

In the text mode: moves the cursor to column *x*. The next PRINT statement will then display its message starting at that point. See also VTAB.

IF/THEN

Tells the computer to make a decision. If the condition following the IF is met, the computer carries out the command that follows the THEN. If the condition is not met, the computer skips the command and goes directly to the next statement.

INPUT

Directs the computer to stop each time it runs the program so that you can supply a value for a variable. The computer displays a question mark, then waits for you to type your answer and press RETURN. Your response is stored in the variable named in the INPUT statement.

LIST

Used by itself, LIST directs the computer to display the stored program on the screen. LIST followed by a statement number displays

only that particular statement. LIST followed by two statement numbers tells the computer to display those two statements and all those in between.

LOAD

Retrieves a program that has been stored on a cassette tape. You must supply the name of the file you want to retrieve:

LOAD *filename*

The computer will erase any program currently in its memory before it loads the new one. To load a file from the second cassette drive, type:

LOAD *filename*, D2

NEW

Clears the computer's program memory so that you can type a new program. If you forget to use this command, lines from your old program may be mixed in with your new statements.

NEXT

Marks the end of a FOR/NEXT loop. This statement must always be paired with a preceding FOR statement. The variable name after the word NEXT must exactly match the name of the counter variable in the FOR statement.

PLOT x,y

In the low-resolution graphics mode: paints a single point at the coordinates *x,y*. Both *x* and *y* must have values between 0 and 39.

PRINT

Displays a message on the screen. If PRINT is followed by a message in quotation marks, the computer prints that message. If PRINT is followed by a word not in quotation marks, the computer treats the word as a variable name and displays its current value. Several messages may be combined in a single PRINT statement, using a semicolon (;) to separate them. A semicolon at the end of the statement keeps the computer from dropping down to the next line before displaying the next message.

PR#

PR#1 instructs the printer to print the results of commands and programs that the computer is running. Any text that would normally be displayed on the screen will also be printed on paper. Type PR#0 to stop the printing.

READ

Works with the DATA statement to assign a list of values to a series of variables. Each variable name in the READ statement is assigned the corresponding value in the DATA statement. The first READ statement in a program loads its numbers starting with the first DATA statement; additional READ statements continue with the first unused number in the DATA statement's list.

RETURN

Marks the end of a subroutine and instructs the computer to resume the main program. The computer returns to the statement following the GOSUB that called the subroutine.

RUN

Asks the computer to execute the program stored in its memory. Program lines are executed in order of their statement numbers,

unless a GOTO, GOSUB, or FOR/NEXT statement alters the flow. At the end of the program, the computer stops and displays a bracket prompt. The program remains stored in the memory until the power is turned off or a NEW command is given.

SAVE

Records the program stored in the computer's memory onto a cassette tape. Each saved program must be given a name so that the computer can later locate it on the cassette:

SAVE *filename*

To save programs on a second cassette drive, type:

SAVE *filename*, D2

TEXT

Returns the computer to the text mode. This command is used to reset the computer from the low-resolution (GR) or high-resolution (HGR) graphics modes.

VLIN *ytop, ybottom* AT *x*

In the low-resolution graphics mode: draws a vertical line in column *x*. The line will be of the color last chosen in a COLOR statement, and will extend from row *ytop* down to row *ybottom*. All three numbers must have values between 0 and 39. Also *ytop* and *ybottom* must be assigned different values.

VTAB *y*

In the text mode: VTAB moves the cursor down to row *y*, counting from the top of the screen. The value of *y* must be between 1 and 23. To move to row 0, use the HOME command. See also HTAB.

=

The equal sign acts as a command, assigning a value to a variable. The computer first carries out any calculations involved in the expression to the right of the equal sign, then stores the result in the variable to the left. For example, the statement

```
I = 2+2
```

would assign the value 4 to the variable *I*.

The equal sign is also used in the COLOR and HCOLOR commands.

Answers to Selected Exercises

5.2: Give the command:

```
print "ONE":print "TWO":print "THREE"
```

5.3: Type this command:

```
home:vtab 23:htab 13:print "Bottom"
```

This command has the computer display the word 'Bottom' on the last row on the screen. After the computer has finished printing the message, however, it automatically drops down to the next line on the screen to print its bracket prompt on a new line. Since the next line would take the computer off the bottom of the screen, the computer moves the rest of the display up.

6.1: There are many possible answers to this problem. This is one series of commands that would work:

```
gr
color = 2
hlin 0,39 at 5
color = 9
hlin 5,20 at 15
color = 14
hlin 0,25 at 30
```

6.2: Type the six commands shown at the top of page 95. Then add your commands to draw the smaller blue box. Many sequences are possible; for example:

```
color = 7
hplot 40,40 to 120,40
hplot to 120,120
hplot to 40,120
hplot to 40,40
```

7.1: Add this line to the program:

```
55  hcolor = 6
```

7.2: Add this line:

```
70  hplot 20,20 to 140,140
```

If you have just done exercise 1, the diagonal line will be blue.

7.3: The four bugs are:

1. The word PRINT is misspelled in line 10.
2. COLOR in line 30 is not a high-resolution graphics command. HCOLOR is what is needed.
3. The second coordinate (734) in line 40 is outside the allowed range.
4. The last point in the HPLOT statement in line 60 has only one coordinate.

8.1: Add a statement number 15, and replace statement 20, as follows:

```
15  input "Color Number? "; n
20  hcolor = n
```

8.2: Add these statements to the program on page 133:

```
120  rem Inverted Triangle
130  hplot x,y+40
140  hplot to x+40,y-20
150  hplot to x+80,y+40
160  hplot to x,y+40
```


8.3: It will set the new value of *j* to equal the old value (10) plus 1, or 11.

9.1:

```
10  INPUT "What Number? "; n
20  IF n>0 THEN PRINT "Positive"
30  IF n<0 THEN PRINT "Negative"
40  IF n=0 THEN PRINT "Zero"
```

9.2:

```
1000  REM Subroutine to Draw Frame
1010  HPLOT 0,0 TO 255,0
1020  HPLOT TO 255,158
1030  HPLOT TO 0,158
1040  HPLOT TO 0,0
1050  RETURN
```

Note that the GOSUB statement that calls this subroutine must come *after* the initial HGR and HCOLOR statements, or the frame will not show up.

9.3: Try this program:

```
10  sum = 0
20  FOR i = 1 TO 100
30  sum = sum+i
40  NEXT i
50  PRINT sum
```

The answer should be 5050.

Index

Selections from The SYBEX Library

Buyer's Guides

THE BEST OF TI 99/4A™ CARTRIDGES

by Thomas Blackadar

150 pp., illustr., Ref. 0-137

Save yourself time and frustration when buying TI 99/4A software. This buyer's guide gives an overview of the best available programs, with information on how to set up the computer to run them.

FAMILY COMPUTERS UNDER $200

by Doug Mosher

160 pp., illustr., Ref. 0-149

Find out what these inexpensive machines can do for you and your family. "If you're just getting started . . . this is the book to read before you buy."—Richard O'Reilly, Los Angeles newspaper columnist

PORTABLE COMPUTERS

by Sheldon Crop and Doug Mosher

128 pp., illustr., Ref. 0-144

"This book provides a clear and concise introduction to the expanding new world of personal computers."—Mark Powelson, Editor, *San Francisco Focus Magazine*

THE BEST OF VIC-20™ SOFTWARE

by Thomas Blackadar

150 pp., illustr., Ref. 0-139

Save yourself time and frustration with this buyer's guide to VIC-20 software. Find the best game, music, education, and home management programs on the market today.

SELECTING THE RIGHT DATA BASE SOFTWARE

SELECTING THE RIGHT WORD PROCESSING SOFTWARE

SELECTING THE RIGHT SPREADSHEET SOFTWARE

by Kathy McHugh and Veronica Corchado

80 pp., illustr., Ref. 0-174, 0-177, 0-178

This series on selecting the right business software offers the busy professional concise, informative reviews of the best available software packages.

Introduction to Computers

OVERCOMING COMPUTER FEAR

by Jeff Berner

112 pp., illustr., Ref. 0-145

This easy-going introduction to computers helps you separate the facts from the myths.

COMPUTER ABC'S

by Daniel Le Noury and Rodnay Zaks

64 pp., illustr., Ref. 0-167

This beautifully illustrated, colorful book for parents and children takes you alphabetically through the world of computers, explaining each concept in simple language.

PARENTS, KIDS, AND COMPUTERS

by Lynne Alper and Meg Holmberg

208 pp., illustr., Ref. 0-151

This book answers your questions about the educational possibilities of home computers.

THE COLLEGE STUDENT'S COMPUTER HANDBOOK

by Bryan Pfaffenberger

350 pp., illustr., Ref. 0-170

This friendly guide will aid students in selecting a computer system for college

study, managing information in a college course, and writing research papers.

COMPUTER CRAZY

by Daniel Le Noury

100 pp., illustr., Ref. 0-173

No matter how you feel about computers, these cartoons will have you laughing about them.

PROTECTING YOUR COMPUTER

by Rodnay Zaks

214pp., 100 illustr., Ref. 0-239

The correct way to handle and care for all elements of a computer system, including what to do when something doesn't work.

YOUR FIRST COMPUTER

by Rodnay Zaks

258 pp., 150 illustr., Ref. 0-045

The most popular introduction to small computers and their peripherals: what they do and how to buy one.

SYBEX PERSONAL COMPUTER DICTIONARY

120 pp., Ref. 0-067

All the definitions and acronyms of microcomputer jargon defined in a handy pocket-sized edition. Includes translations of the most popular terms into ten languages.

FROM CHIPS TO SYSTEMS: AN INTRODUCTION TO MICROPROCESSORS

by Rodnay Zaks

552 pp., 400 illustr., Ref. 0-063

A simple and comprehensive introduction to microprocessors from both a hardware and software standpoint: what they are, how they operate, how to assemble them into a complete system.

Personal Computers

ATARI

YOUR FIRST ATARI® PROGRAM

by Rodnay Zaks

150 pp., illustr., Ref. 0-130

A fully illustrated, easy-to-use introduction to ATARI BASIC programming. Will have the reader programming in a matter of hours.

BASIC EXERCISES FOR THE ATARI®

by J.P. Lamoitier

251 pp., illustr., Ref. 0-101

Teaches ATARI BASIC through actual practice using graduated exercises drawn from everyday applications.

THE EASY GUIDE TO YOUR ATARI® 600XL/800XL

by Thomas Blackadar

175 pp., illustr., Ref. 0-125

This jargon-free companion will help you get started on the right foot with your new 600XL or 800XL ATARI computer.

ATARI® BASIC PROGRAMS IN MINUTES

by Stanley R. Trost

170 pp., illustr., Ref. 0-143

You can use this practical set of programs without any prior knowledge of BASIC! Application examples are taken from a wide variety of fields, including business, home management, and real estate.

Commodore 64/VIC-20

THE COMMODORE 64™/VIC-20™ BASIC HANDBOOK

by Douglas Hergert

144 pp., illustr., Ref. 0-116

A complete listing with descriptions and instructive examples of each of the Commodore 64 BASIC keywords and functions. A handy reference guide, organized like a dictionary.

THE EASY GUIDE TO YOUR COMMODORE 64™

by Joseph Kascmer

160 pp., illustr., Ref. 0-129

A friendly introduction to using the Commodore 64.

THE BEST OF COMMODORE 64™ SOFTWARE

by Thomas Blackadar

150pp., illustr., Ref. 0-194

Save yourself time and frustration with this buyer's guide to Commodore 64 software. Find the best game, music, education, and home management programs on the market today.

YOUR FIRST VIC-20™ PROGRAM

by Rodnay Zaks

150 pp., illustr., Ref. 0-129

A fully illustrated, easy-to-use introduction to VIC-20 BASIC programming. Will have the reader programming in a matter of hours.

THE VIC-20™ CONNECTION

by James W. Coffron

260 pp., 120 illustr., Ref. 0-128

Teaches elementary interfacing and BASIC programming of the VIC-20 for connection to external devices and household appliances.

YOUR FIRST COMMODORE 64™ PROGRAM

by Rodnay Zaks

182 pp., illustr., Ref. 0-172

You can learn to write simple programs without any prior knowledge of mathematics or computers! Guided by colorful illustrations and step-by-step instructions, you'll be constructing programs within an hour or two.

COMMODORE 64™ BASIC PROGRAMS IN MINUTES

by Stanley R. Trost

170 pp., illustr., Ref. 0-154

Here is a practical set of programs for business, finance, real estate, data analysis, record keeping and educational applications.

GRAPHICS GUIDE TO THE COMMODORE 64™

by Charles Platt

192 pp., illustr., Ref. 0-138

This easy-to-understand book will appeal to anyone who wants to master the Commodore 64's powerful graphics features.

IBM

THE ABC'S OF THE IBM® PC

by Joan Lasselle and Carol Ramsay

100 pp., illustr., Ref. 0-102

This is the book that will take you through the first crucial steps in learning to use the IBM PC.

THE BEST OF IBM® PC SOFTWARE

by Stanley R. Trost

144 pp., illustr., Ref. 0-104

Separates the wheat from the chaff in the world of IBM PC software. Tells you what to expect from the best available IBM PC programs.

THE IBM® PC-DOS HANDBOOK

by Richard Allen King

144 pp., illustr., Ref. 0-103

Explains the PC disk operating system, giving the user better control over the system. Get the most out of your PC by adapting its capabilities to your specific needs.

BUSINESS GRAPHICS FOR THE IBM® PC

by Nelson Ford

200 pp., illustr., Ref. 0-124

Ready-to-run programs for creating line graphs, complex illustrative multiple bar graphs, picture graphs, and more. An ideal way to use your PC's business capabilities!

THE IBM® PC CONNECTION

by James W. Coffron

200 pp., illustr., Ref. 0-127

Teaches elementary interfacing and BASIC programming of the IBM PC for connection to external devices and household appliances.

BASIC EXERCISES FOR THE IBM® PERSONAL COMPUTER

by J.P. Lamoitier

252 pp., 90 illustr., Ref. 0-088

Teaches IBM BASIC through actual practice, using graduated exercises drawn from everyday applications.

USEFUL BASIC PROGRAMS FOR THE IBM® PC

by Stanley R. Trost

144 pp., Ref. 0-111

This collection of programs takes full advantage of the interactive capabilities of your IBM Personal Computer. Financial calculations, investment analysis, record keeping, and math practice—made easier on your IBM PC.

YOUR FIRST IBM® PC PROGRAM

by Rodnay Zaks

182 pp., illustr., Ref. 0-171

This well-illustrated book makes programming easy for children and adults.

THE COMPLETE GUIDE TO YOUR IBM® PC JUNIOR

by Douglas Hergert

250 pp., illustr., Ref. 0-179

This comprehensive reference guide to IBM's most economical microcomputer offers many practical applications and all the helpful information you'll need to get started with your IBM PC Junior.

DATA FILE PROGRAMMING ON YOUR IBM® PC

by Alan Simpson

275 pp., illustr., Ref. 0-146

This book provides instructions and examples of managing data files in BASIC. Programming designs and developments are extensively discussed.

Apple

THE EASY GUIDE TO YOUR APPLE II®

by Joseph Kascmer

160 pp., illustr., Ref. 0-122

A friendly introduction to using the Apple II, II plus and the new IIe.

BASIC EXERCISES FOR THE APPLE®

by J.P. Lamoitier

250 pp., 90 illustr., Ref. 0-084

Teaches Apple BASIC through actual practice, using graduated exercises drawn from everyday applications.

APPLE II® BASIC HANDBOOK

by Douglas Hergert

144 pp., illustr., Ref. 0-155

A complete listing with descriptions and instructive examples of each of the Apple II BASIC keywords and functions. A handy reference guide, organized like a dictionary.

APPLE II® BASIC PROGRAMS IN MINUTES

by Stanley R. Trost

150 pp., illustr., Ref. 0-121

A collection of ready-to-run programs for financial calculations, investment analysis, record keeping, and many more home and office applications. These programs can be entered on your Apple II plus or IIe in minutes!

YOUR FIRST APPLE II® PROGRAM

by Rodnay Zaks

150 pp., illustr., Ref. 0-136

A fully illustrated, easy-to-use introduction to APPLE BASIC programming. Will have the reader programming in a matter of hours.

THE APPLE® CONNECTION

by James W. Coffron

264 pp., 120 illustr., Ref. 0-085

Teaches elementary interfacing and BASIC programming of the Apple for connection to external devices and household appliances.

TRS-80

YOUR COLOR COMPUTER

by Doug Mosher

350 pp., illustr., Ref. 0-097

Patience and humor guide the reader through purchasing, setting up, programming, and using the Radio Shack TRS-80/TDP Series 100 Color Computer. A complete introduction.

THE FOOLPROOF GUIDE TO SCRIPSIT™ WORD PROCESSING

by Jeff Berner

225 pp., illustr., Ref. 0-098

Everything you need to know about SCRIPSIT—from starting out, to mastering document editing. This user-friendly guide is written in plain English, with a touch of wit.

Timex/Sinclair 1000/ZX81

YOUR TIMEX/SINCLAIR 1000 AND ZX81™

by Douglas Hergert

159 pp., illustr., Ref. 0-099

This book explains the set-up, operation, and capabilities of the Timex/Sinclair 1000 and ZX81. Includes how to interface peripheral devices, and introduces BASIC programming.

THE TIMEX/SINCLAIR 1000™ BASIC HANDBOOK

by Douglas Hergert

170 pp., illustr., Ref. 0-113

A complete alphabetical listing with explanations and examples of each word in the T/S 1000 BASIC vocabulary; will allow you quick, error-free programming of your T/S 1000.

TIMEX/SINCLAIR 1000™ BASIC PROGRAMS IN MINUTES

by Stanley R. Trost

150 pp., illustr., Ref. 0-119

A collection of ready-to-run programs for financial calculations, investment analysis, record keeping, and many more home and office applications. These programs can be entered on your T/S 1000 in minutes!

MORE USES FOR YOUR TIMEX/SINCLAIR 1000™ Astronomy on Your Computer

by Eric Burgess

176 pp., illustr., Ref. 0-112

Ready-to-run programs that turn your TV into a planetarium.

Other Popular Computers

YOUR FIRST TI 99/4A™ PROGRAM

by Rodnay Zaks

182 pp., illustr., Ref. 0-157

Colorfully illustrated, this book concentrates on the essentials of programming in a clear, entertaining fashion.

THE RADIO SHACK® NOTEBOOK COMPUTER

by Orson Kellogg

128 pp., illustr., Ref. 0-150

Whether you already have the Radio Shack Model 100 notebook computer, or are interested in buying one, this book will clearly explain what it can do for you.

THE EASY GUIDE TO YOUR COLECO ADAM™

by Thomas Blackadar

175 pp., illustr., Ref. 0-181

This quick reference guide shows you how to get started on your Coleco Adam with a minimum of technical jargon.

Software and Applications

Operating Systems

THE CP/M® HANDBOOK

by Rodnay Zaks

320 pp., 100 illustr., Ref 0-048

An indispensable reference and guide to CP/M—the most widely-used operating system for small computers.

MASTERING CP/M®

by Alan R. Miller

398 pp., illustr., Ref. 0-068

For advanced CP/M users or systems programmers who want maximum use of the CP/M operating system . . . takes up where our *CP/M Handbook* leaves off.

THE BEST OF CP/M® SOFTWARE

by John D. Halamka

250 pp., illustr., Ref. 0-100

This book reviews tried-and-tested, commercially available software for your CP/M system.

REAL WORLD UNIX™

by John D. Halamka

250 pp., illustr., Ref. 0-093

This book is written for the beginning and intermediate UNIX user in a practical, straightforward manner, with specific instructions given for many special applications.

THE CP/M PLUS™ HANDBOOK

by Alan R. Miller

250 pp., illustr., Ref. 0-158

This guide is easy for the beginner to understand, yet contains valuable information for advanced users of CP/M Plus (Version 3).

Business Software

INTRODUCTION TO WORDSTAR™

by Arthur Naiman

202 pp., 30 illustr., Ref. 0-077

Makes it easy to learn how to use WordStar, a powerful word processing program for personal computers.

PRACTICAL WORDSTAR™ USES

by Julie Anne Arca

200 pp., illustr., Ref. 0-107

Pick your most time-consuming office tasks and this book will show you how to streamline them with WordStar.

MASTERING VISICALC®

by Douglas Hergert

217 pp., 140 illustr., Ref. 0-090

Explains how to use the VisiCalc "electronic spreadsheet" functions and provides examples of each. Makes using this powerful program simple.

DOING BUSINESS WITH VISICALC®

by Stanley R. Trost

260 pp., Ref. 0-086

Presents accounting and management planning applications—from financial statements to master budgets; from pricing models to investment strategies.

DOING BUSINESS WITH SUPERCALC™

by Stanley R. Trost

248 pp., illustr., Ref. 0-095

Presents accounting and management planning applications—from financial statements to master budgets; from pricing models to investment strategies.

VISICALC® FOR SCIENCE AND ENGINEERING

by Stanley R. Trost and Charles Pomernacki

225 pp., illustr., Ref. 0-096

More than 50 programs for solving technical problems in the science and engineering fields. Applications range from math and statistics to electrical and electronic engineering.

DOING BUSINESS WITH 1-2-3™

by Stanley R. Trost

250 pp., illustr., Ref. 0-159

If you are a business professional using the 1-2-3 software package, you will find the spreadsheet and graphics models provided in this book easy to use "as is" in everyday business situations.

THE ABC'S OF 1-2-3™

by Chris Gilbert

225 pp., illustr., Ref. 0-168

For those new to the LOTUS 1-2-3 program, this book offers step-by-step instructions in mastering its spreadsheet, data base, and graphing capabilities.

UNDERSTANDING dBASE II™

by Alan Simpson

220 pp., illustr., Ref. 0-147

Learn programming techniques for mailing label systems, bookkeeping and data base management, as well as ways to interface dBASE II with other software systems.

DOING BUSINESS WITH dBASE II™

by Stanley R. Trost

250 pp., illustr., Ref. 0-160

Learn to use dBASE II for accounts receivable, recording business income and expenses, keeping personal records and mailing lists, and much more.

DOING BUSINESS WITH MULTIPLAN™

by Richard Allen King and Stanley R. Trost

250 pp., illustr., Ref. 0-148

This book will show you how using Multiplan can be nearly as easy as learning to use a pocket calculator. It presents a collection of templates that can be applied "as is" to business situations.

DOING BUSINESS WITH PFS®

by Stanley R. Trost

250 pp., illustr., Ref. 0-161

This practical guide describes specific business and personal applications in detail. Learn to use PFS for accounting, data analysis, mailing lists and more.

INFOPOWER: PRACTICAL INFOSTAR™ USES

by Jule Anne Arca and Charles F. Pirro

275 pp., illustr., Ref. 0-108

This book gives you an overview of InfoStar, including DataStar and ReportStar, WordStar, MailMerge, and SuperSort. Hands on exercises take you step-by-step through real life business applications.

WRITING WITH EASYWRITER II™

by Douglas W. Topham

250 pp., illustr., Ref. 0-141

Friendly style, handy illustrations, and numerous sample exercises make it easy to learn the EasyWriter II word processing system.

Business Applications

INTRODUCTION TO WORD PROCESSING

by Hal Glatzer

205 pp., 140 illustr., Ref. 0-076

Explains in plain language what a word processor can do, how it improves productivity, how to use a word processor and how to buy one wisely.

COMPUTER POWER FOR YOUR LAW OFFICE

by Daniel Remer

225 pp., Ref. 0-109

How to use computers to reach peak productivity in your law office, simply and inexpensively.

OFFICE EFFICIENCY WITH PERSONAL COMPUTERS

by Sheldon Crop

175 pp., illustr., Ref. 0-165

Planning for computerization of your office? This book provides a simplified discussion of the challenges involved for everyone from business owner to clerical worker.

COMPUTER POWER FOR YOUR ACCOUNTING OFFICE

by James Morgan

250 pp., illustr., Ref. 0-164

This book is a convenient source of information about computerizing you accounting office, with an emphasis on hardware and software options.

Languages

C

UNDERSTANDING C

by Bruce Hunter

200 pp., Ref 0-123

Explains how to use the powerful C language for a variety of applications. Some programming experience assumed.

FIFTY C PROGRAMS
by Bruce Hunter
200 pp., illustr., Ref. 0-155
Beginning as well as intermediate C programmers will find this a useful guide to programming techniques and specific applications.

BASIC

YOUR FIRST BASIC PROGRAM
by Rodnay Zaks
150pp. illustr. in color, Ref. 0-129
A "how-to-program" book for the first time computer user, aged 8 to 88.

FIFTY BASIC EXERCISES
by J. P. Lamoitier
232 pp., 90 illustr., Ref. 0-056
Teaches BASIC by actual practice, using graduated exercises drawn from everyday applications. All programs written in Microsoft BASIC.

INSIDE BASIC GAMES
by Richard Mateosian
348 pp., 120 illustr., Ref. 0-055
Teaches interactive BASIC programming through games. Games are written in Microsoft BASIC and can run on the TRS-80, Apple II and PET/CBM.

BASIC FOR BUSINESS
by Douglas Hergert
224 pp., 15 illustr., Ref. 0-080
A logically organized, no-nonsense introduction to BASIC programming for business applications. Includes many fully-explained accounting programs, and shows you how to write them.

EXECUTIVE PLANNING WITH BASIC
by X. T. Bui
196 pp., 19 illustr., Ref. 0-083
An important collection of business management decision models in BASIC, including Inventory Management (EOQ), Critical Path Analysis and PERT, Financial Ratio Analysis, Portfolio Management, and much more.

BASIC PROGRAMS FOR SCIENTISTS AND ENGINEERS
by Alan R. Miller
318 pp., 120 illustr., Ref. 0-073
This book from the "Programs for Scientists and Engineers" series provides a library of problem-solving programs while developing proficiency in BASIC.

CELESTIAL BASIC
by Eric Burgess
300 pp., 65 illustr., Ref. 0-087
A collection of BASIC programs that rapidly complete the chores of typical astronomical computations. It's like having a planetarium in your own home! Displays apparent movement of stars, planets and meteor showers.

YOUR SECOND BASIC PROGRAM
by Gary Lippman
250 pp., illustr., Ref. 0-152
A sequel to *Your First BASIC Program*, this book follows the same patient, detailed approach and brings you to the next level of programming skill.

Pascal

INTRODUCTION TO PASCAL (Including UCSD Pascal™)
by Rodnay Zaks
420 pp., 130 illustr., Ref. 0-066
A step-by-step introduction for anyone wanting to learn the Pascal language. Describes UCSD and Standard Pascals. No technical background is assumed.

THE PASCAL HANDBOOK
by Jacques Tiberghien
486 pp., 270 illustr., Ref. 0-053
A dictionary of the Pascal language, defining every reserved word, operator, procedure and function found in all major versions of Pascal.

APPLE® PASCAL GAMES
by Douglas Hergert and Joseph T. Kalash
372 pp., 40 illustr., Ref. 0-074
A collection of the most popular computer games in Pascal, challenging the reader not only to play but to investigate how games are implemented on the computer.

INTRODUCTION TO THE UCSD p-SYSTEM™
by Charles W. Grant and Jon Butah
300 pp., 10 illustr., Ref. 0-061
A simple, clear introduction to the UCSD Pascal Operating System; for beginners through experienced programmers.

PASCAL PROGRAMS FOR SCIENTISTS AND ENGINEERS
by Alan R. Miller
374 pp., 120 illustr., Ref. 0-058
A comprehensive collection of frequently used algorithms for scientific and technical applications, programmed in Pascal. Includes such programs as curve-fitting, integrals and statistical techniques.

DOING BUSINESS WITH PASCAL
by Richard Hergert and Douglas Hergert
371 pp., illustr., Ref. 0-091
Practical tips for using Pascal in business programming. Includes design considerations, language extensions, and applications examples.

Assembly Language Programming

PROGRAMMING THE 6502
by Rodnay Zaks
386 pp., 160 illustr., Ref. 0-046
Assembly language programming for the 6502, from basic concepts to advanced data structures.

6502 APPLICATIONS
by Rodnay Zaks
278 pp., 200 illustr., Ref. 0-015
Real-life application techniques: the input/output book for the 6502.

ADVANCED 6502 PROGRAMMING
by Rodnay Zaks
292 pp., 140 illustr., Ref. 0-089
Third in the 6502 series. Teaches more advanced programming techniques, using games as a framework for learning.

PROGRAMMING THE Z80
by Rodnay Zaks
624 pp., 200 illustr., Ref. 0-069
A complete course in programming the Z80 microprocessor and a thorough introduction to assembly language.

Z80 APPLICATIONS
by James W. Coffron
288 pp., illustr., Ref. 0-094
Covers techniques and applications for using peripheral devices with a Z80 based system.

PROGRAMMING THE 6809
by Rodnay Zaks and William Labiak
362 pp., 150 illustr., Ref. 0-078
This book explains how to program the 6809 in assembly language. No prior programming knowledge required.

PROGRAMMING THE Z8000
by Richard Mateosian
298 pp., 124 illustr., Ref. 0-032
How to program the Z8000 16-bit microprocessor. Includes a description of the architecture and function of the Z8000 and its family of support chips.

PROGRAMMING THE 8086/8088
by James W. Coffron
300 pp., illustr., Ref. 0-120
This book explains how to program the 8086 and 8088 in assembly language. No prior programming knowledge required.

Other Languages

FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS
by Alan R. Miller
280 pp., 120 illustr., Ref. 0-082
In the "Programs for Scientists and Engineers" series, this book provides specific scientific and engineering application programs written in FORTRAN.

A MICROPROGRAMMED APL IMPLEMENTATION
by Rodnay Zaks
350 pp., Ref. 0-005
An expert-level text presenting the complete conceptual analysis and design of an APL interpreter, and actual listing of the microcode.

Hardware and Peripherals

MICROPROCESSOR INTERFACING TECHNIQUES
by Rodnay Zaks and Austin Lesea
456 pp., 400 illustr., Ref. 0-029
Complete hardware and software interconnect techniques, including D to A conversion, peripherals, standard buses and troubleshooting.

THE RS-232 SOLUTION
by Joe Campbell
225 pp., illustr., Ref. 0-140
Finally, a book that will show you how to correctly interface your computer to any RS-232-C peripheral.

USING CASSETTE RECORDERS WITH COMPUTERS
by James Richard Cook
175 pp., illustr., Ref. 0-169
Whatever your computer or application, you will find this book helpful in explaining details of cassette care and maintenance.

For a complete catalog of our publications please contact:

U.S.A.
SYBEX, Inc.
2344 Sixth Street
Berkeley,
California 94710
Tel: (800) 227-2346
(415) 848-8233
Telex: 336311

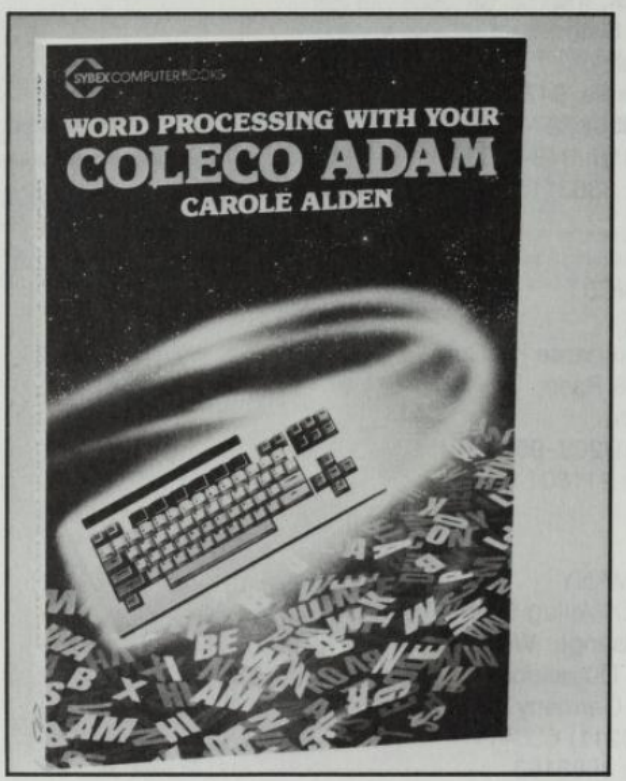
FRANCE
SYBEX
6-8 Impasse du Curé
75018 Paris
France
Tel: 01/203-9595
Telex: 211801

GERMANY
SYBEX-Verlag GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
West Germany
Tel: (0211) 626411
Telex: 8588163

SYBEX

DON'T MISS THIS HANDY GUIDE TO

WORD PROCESSING WITH YOUR COLECO ADAM



Now that you have a working knowledge of your Coleco Adam, you'll want to learn how to streamline your writing chores with the word processing features of the Adam. Here is a book to show you how. The friendly, hands-on approach is designed to help the complete beginner master the basics quickly and easily. You'll find many examples of practical applications for every member of the family, as well as tips on caring for the Adam and information about future add-ons.

ISBN: 0-89588-182-9, 175 pp., illustrated, 6"x 9"

THE EASY GUIDE TO YOUR COLECO ADAM

The Easy Guide to Your Coleco Adam is the next best thing to having your own personal computer expert right there to coach you through every step! This book is designed to take you all the way from set-up to a thorough working knowledge of your machine. In just a few hours you'll be off and running, loading software, entering data, and learning to use:

- the word processor
- graphics features
- cassettes
- the Adam printer
- BASIC programming
- and more!

Everything you need to know to get started on your Coleco Adam is revealed here in a friendly, jargon-free style. This is truly computing made easy!

ABOUT THE AUTHOR: Thomas Blackadar is a professional writer who has been using computers for more than ten years. A graduate of Princeton University, he has studied at the University of Fribourg, Switzerland. He is the author of *The Best of TI 99/4A Cartridges, The Easy Guide to Your Atari 600XL/800XL, The Best of VIC-20 Software,* and *The Best of Commodore 64 Software.*

ISBN 0-89588-181-0