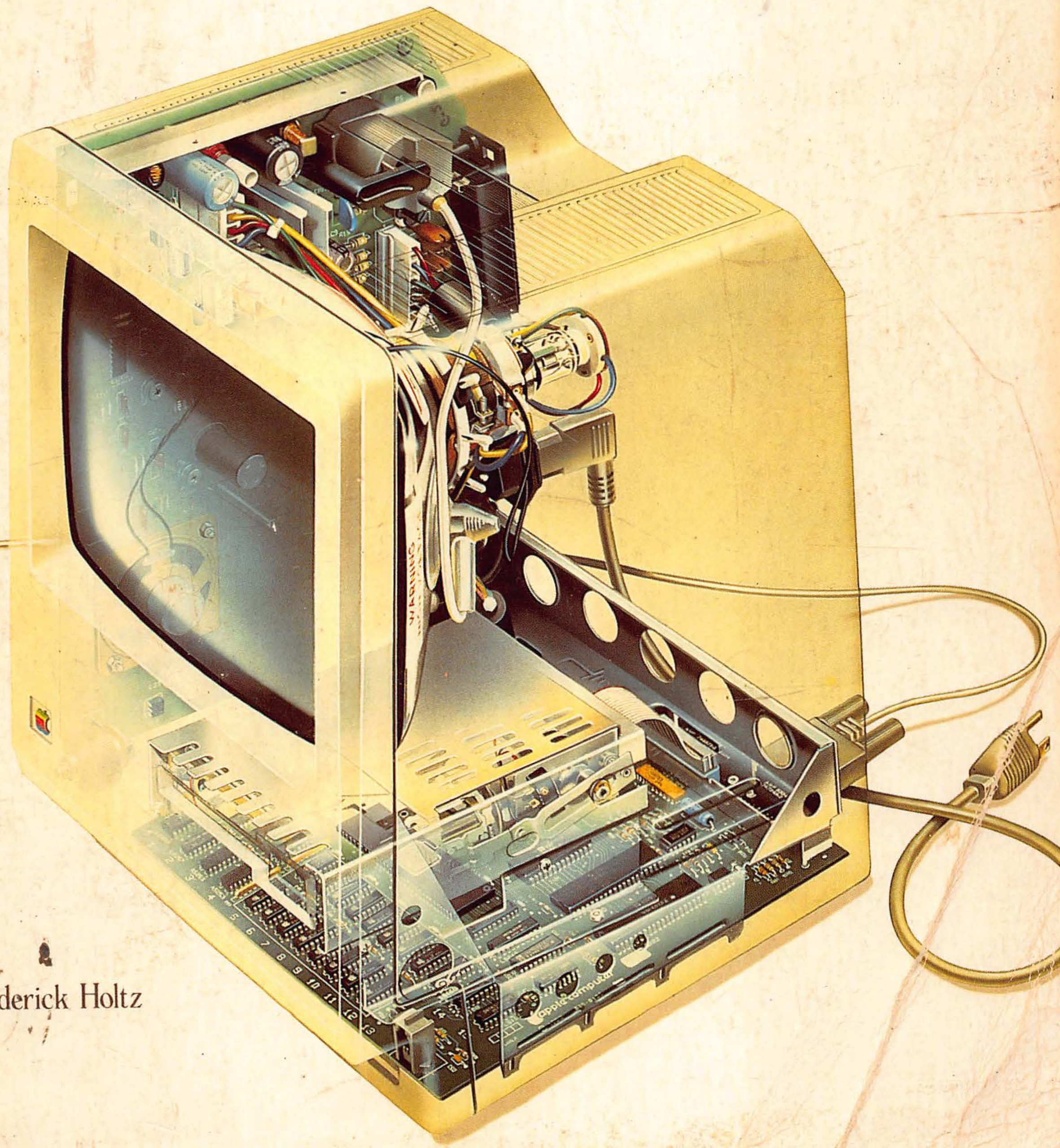
1840

Using & Programming the Macintosh™

including 32 Ready-to-Run Programs



By Frederick Holtz

Using & Programming the Macintosh®
including 32 Ready-to-Run Programs

To my son Robert, whose help on this project was greatly appreciated, and whose faith and trust over the past 16 years have meant more than he could ever know.

Using & Programming the Macintosh®
including 32 Ready-to-Run Programs

By Frederick Holtz

TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

Apple® is a registered trademark of Apple Computer, Inc.

IBM® is a registered trademark of International Business Machines Corporation.

Multiplan™ is a trademark of Microsoft Corporation.

Lotus 1-2-3™ is a trademark of Lotus Development Corporation.

Lisa™ is a trademark of Apple Computer, Inc.

Macintosh™ is a trademark licensed to Apple Computer, Inc.

MacPaint™ is a trademark of Apple Computer, Inc.

MacWrite™ is a trademark of Apple Computer, Inc.

MacTerminal™ is a trademark of Apple Computer, Inc.

Macintosh Assembler/Debugger™ is a trademark of Apple Computer, Inc.

Macintosh Project™ is a trademark of Apple Computer, Inc.

MacDraw™ is a trademark of Apple Computer, Inc.

Property of U.S. Army
Main Post Library
Fort Stewart, GA 31314

FIRST EDITION

THIRD PRINTING

Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Copyright © 1984 by TAB BOOKS Inc.

Library of Congress Cataloging in Publication Data

Holtz, Frederick.
Using and programming the Macintosh, including 32 ready-to-run programs.
Includes index.
1. Macintosh (Computer) 2. Macintosh (Computer)—Programming. 3. Computer programs. I. Title.
QA76.8.M3H65 1984 001.64'2 84-8428
ISBN 0-8306-0840-0
ISBN 0-8306-1840-6 (pbk.)

Cover photograph courtesy of Apple Computer Inc.

Contents

Acknowledgments vii

Introduction viii

1 An Evolution in the Microcomputer Market 1
A New Concept—Apple Versus IBM—Super ROMs—Macintosh Critics—The Computer for Everyone—Features of the Macintosh—Items on the Macintosh Desk Top—Software for the Macintosh—*MacWrite, MacPaint* and More—Specialized Software—Where to Find a Macintosh—The Lisa-Macintosh Connection—Summary

2 Using *MacPaint* 20
The Graphics Tools—The Menu Bar—Patterns, Patterns, Patterns—Drawing Lines—Some Graphic Results—Shrinking and Enlarging—Using a Few More Tools—Summary

3 Using *MacWrite* 43
Spacing and Alignment—The Search Selection—Indenting and Tabbing—Changing the Typeface—Printing Your Document—Saving Your Document—Summary

4 Using The Macintosh Finder 68
The Finder Menu Bar—File Handling—Summary

5 Manipulating the Microsoft BASIC Screen 86
Four Windows—Using the Windows—Editing in BASIC—Multiple LIST Windows—Moving Other Windows—The Menu Bar—Summary

6 Programming the Macintosh in Microsoft BASIC 115
A First Program—Clearing the Screen—Two Modes in BASIC—Creating a Loop—More About Loops—

FOR-NEXT Loops—Going Further with FOR-NEXT Loops—More on PRINT Statements—Variables—Naming Variables—Manipulating Numeric Variables—INPUT Statements—Formulas—Built-in Functions—More String Functions—Another Branch Statement—READ/DATA Statements—Multiple Statement Lines—Logical Operators—Relational Operators—Arrays—Summary

7 Programming Graphics in BASIC 149
The Graphics Screen—CIRCLE Statements—LINE Statements—Use of Color—PSET/PRESET Statements—Animation Techniques—More Ways to Use Graphics Statements—The POINT Function—Summary

8 Macintosh Filekeeping 168
File Reading Program—Advanced File Reading Program—File Writing Program—Another File Writing Program—File Appending Program—Complete Filing Program—File Item Search Program—Partial Item File Search—Summary

9 Programming the Mouse 178
The MOUSE Function—Calling a ROM Subroutine—Putting the Mouse to Work—Summary

10 Ready-to-Run Programs 191
Ohm's Law—Mortgage Payments—Leap Year Calculator—Binary to Decimal Conversion—Checkbook Balancer—Macintosh Typewriter—Alphabetizer—Alarm Clock—ASCII Character Display—Mathematical Circle—Random Lines—Graphic Funnel—Steer Horn—Circle Pattern—Multicircle Pattern—Solar System—Four-pointed Star—Smoking Cigarette—Oriental Prince Picture—Automatic Random Seed—Word Maze—Numbers Guess Game—Computer Numbers Guess—Summary

Appendix A ASCII Character Codes 229

Appendix B Non-ASCII Character Codes 230

Appendix C ImageWriter Printer Specifications 231

Appendix D ASCII, Binary, and Hexadecimal Print Codes 234

Appendix E More *MacPaint* Drawing Samples 236

Appendix F Bar Charts 239

Index 242

Acknowledgments

I wish to thank the staff and management of Computer Solutions in Leesburg, Virginia, for supplying the Macintosh system used to research and write this book.

Many thanks to the staff and management of Frederick Computer Products in Frederick, Maryland, for providing technical data and support throughout the writing of this book.

Introduction

The Apple Macintosh is here, and it can be accurately described as a jewel of a computer. Rumored for over a year, the Macintosh is more than anyone expected. It's much more. It may eventually change the way we humans interact with computers, when other manufacturers begin to (and they undoubtedly will) emulate its unique interactive style. This can only benefit the consumer.

The Macintosh was first rumored to be a high-level market personal computer, offering an 8088 microprocessor that makes it IBM PC-compatible. This rumor was completely false. Unlike other manufacturers, Apple has chosen to go its own route and has now arrived—with a machine that contains a 32-bit microprocessor, which operates at incredible speed. Hardware is only one factor.

There are already more than 500 software packages on the market for the Macintosh, and they give the user tremendous flexibility. And it contains a good number of bundled programs, too.

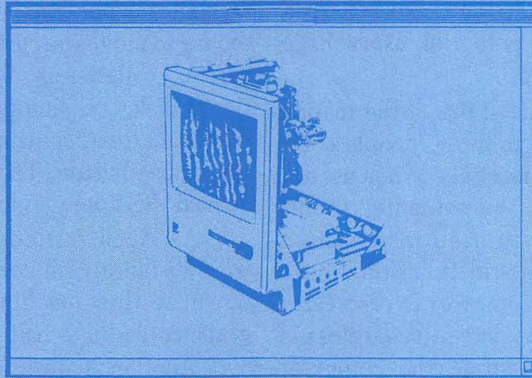
At this writing, the Macintosh is a brand-new phenomenon. Its operational features seem incredible by any standard in the microcomputer industry. For years, other computers have been advertised as being flexible, but none comes close to Macintosh. At long last, there really is a computer that offers the experienced programmer high-powered computational capabilities and at the same time offers the beginner an easy-to-use machine—one that's easier than all the rest.

This book is more than a simple introduction to the Macintosh. Certainly, it overviews the system's features, but it helps tell Macintosh's true story by helping you interact with it. This book is dedicated to explaining how to get the most from your machine. Also, if you are considering purchasing a Macintosh, this text can function as a valuable source of information needed to decide whether or not Macintosh is the right computer for you. Beginner programmers and users who have never even touched a microcomputer will espe-

cially appreciate the detailed step-by-step chapters on using the Macintosh and programming in Microsoft BASIC. If you've never programmed a computer before, you will be able to do so in just a few hours. Additionally, many programs are included that you can type into your computer. You can use them to perform complex mathematical operations in seconds, draw accurate images on the screen, and even play a game or two.

The Macintosh is here. It is a force to be reckoned with by the microcomputer industry and a machine that will become your close partner. Macintosh, more than any other computer, is a true extension of you, the user. It doesn't take the place of an inquisitive, alert mind, but it allows that mind to reach into areas not previously explored. Compact, powerful, and user-friendly are all terms that accurately describe the Macintosh. I think you will agree that Macintosh truly bridges the gap between people and personal computers.

Chapter 1



An Evolution in the Microcomputer Market

The microcomputer revolution has been sprung upon us again! A few years ago, most of us predicted that the hardware revolution was just about over for a decade or so. We also predicted that the industry would turn itself toward the task of revolutionizing microcomputer software. Most of these predictions occurred shortly after IBM announced their Personal Computer.

The IBM PC seemed to be a quantum leap for microcomputers, and indeed, it was. It was touted as the most sophisticated microcomputer ever offered. Most advertisments about the machine centered around the fact that it was the first mass marketed personal computer to contain a microprocessor with 16-bit architecture. For those of you not familiar with computer electronics, the *micro - processor*, or *chip*, is the heart of any microcomputer, and the chips architecture determines the amount of data the machine can handle in any one operation. Most pre-IBM PC microcomputers contained eight-bit microprocessors, and many people were impressed with the increased capabilities of the 16-bit systems when they appeared.

A NEW CONCEPT

Before the 1930s, driving an automobile involved learning transmission shift patterns and synchronizing those patterns with the clutch pedal. That combination of coordination and expertise kept a number of people from using the automobile. But in the late 1930s, automobile manufacturers introduced the automatic transmission, a feature that put more people behind the wheel.

Personal computers have followed a similar course. From the Apple II through the IBM PC, personal computers have forced people to learn unfamiliar ways of doing familiar things. To operate personal computers, users had to learn about operating systems, study programming languages, decipher cryptic messages, and perform tasks in ways very different from the manual tasks the application software was supposed to improve. Learning

to use these personal computers required a large investment of the individual's time. And there was generally a phobia surrounding the computer. As a result, a large number of potential users have stayed away from computers.

The folks at Apple decided that if the market for personal computers was to grow, people could not be forced to acquire special skills. When used to type a letter, for instance, the computer should resemble a typewriter. When used to prepare a diagram, the computer should function like a sketchpad. People want the computer to respond in ways that are familiar and consistent, regardless of what it does. They are awaiting the computer equivalent of the automatic transmission.

The Xerox 8010 Workstation (Star) was the first computer system to embody features that attempted to make computers easy to use and understand. It offered a powerful "user interface" that simplified the system's operation. Symbols and graphics, rather than coded messages, represented the computer's different resources. And the Star used *pointing* technology, where the *mouse*, like a joystick, moved the cursor around on the screen. The Star made great strides toward using, learning, and understanding computers. But its dependence on a large network made Star impractical for small installations.

In January 1983, Apple Computer introduced its Lisa computer shown with the Macintosh in Fig. 1-1, a personal computer that improved the concepts used in Star and increased its simplicity and usefulness. And Lisa could be used as a stand-alone work station, so offices could start with one and add others as needed. Additionally, Lisa used a microprocessor with a 32-bit architecture, allowing it to handle even more data than the 16-bit chips that had only recently been so impressive.

Lisa was to revolutionize the way people interacted with microcomputers. No longer would it be necessary for the user to learn a computer language. All you had to do was turn the computer on and, instead of typing English words to tell the computer what you wanted to do, you used the mouse to move the cursor to picture symbols on the screen, called *icons*. If you wanted to access a floppy disk, you simply used the mouse to place the cursor on the disk icon displayed and then pressed the mouse button. The computer automatically accessed the floppy disk in the drive.

As Apple stated quite accurately, a person with absolutely no computer experience could sit down at this machine and draw sophisticated graphics within a half hour or so.

Another feature that made Lisa unique among microcomputers was that no high-level language was offered for the machine. It was never meant to be *programmed* by the average user. Rather, Apple supplied the programs the user might need, including graphics programs, a word processor, and various business packages.

Lisa, though, was considerably less than a booming success for Apple. The machine simply did not catch on with a large segment of the market. Certainly, one reason it did not was the price, abutting $10,000. Lisa was intended as a businessman's machine and the price reflected this intention. Another reason for its less than spectacular showing was that most software was only available for the IBM PC and similar hot-selling machines. Because the Lisa concept was untested, few companies supported it with software. Undoubtedly, a large number of computer buyers were simply afraid to take a chance on buying such an expensive machine for which there was limited software support.

Rumors were circulating that Apple, who had reportedly spent $10 million dollars on the Lisa project, was in serious trouble. The Apple II series of computers had been a mainstay of the company for many years. At one time, the Apple was *the* microcomputer manufacturer, and even today,

Fig. 1-1. Apple Computer's Macintosh is shown here with mouse and detachable keyboard. Peripheral devices and accessories include a numeric keypad, an external disk drive, the ImageWriter printer, a modem, and the Macintosh carrying case. (Courtesy of Apple Computer, Inc.)

there is still more software available for Apple computers than for any other brand.

APPLE VERSUS IBM

While the Apple II was and is a good basic machine, many persons feel that its limitations render it relatively obsolete when compared with other personal computers and indeed, with many home computers. A more sophisticated buying public is expecting more and more of their machines, and the Apple II series is struggling to keep up. Also, Microsoft BASIC, used by the IBM PC and many other computers, has become established as somewhat universal, and the Applesoft BASIC, used with the Apple II series, is considerably different.

Apple's step forward with Lisa was a much needed move, to set it apart from competitors in the fast changing marketplace. The Macintosh now offers many of Lisa's features at a much more competitive price. Until Macintosh was released, the IBM PC was considered king of microcomputers, because of the numbers sold. It's so popular that the market abounds with many IBM PC workalikes. Right now, more software is being written for the IBM PC than any other microcomputer. While the PC is an excellent microcomputer, and it has tremendous support both in software and peripherals, it does have its shortcomings. Once considered an extremely fast microcomputer because of its ability to receive and display data, it is rather slow by comparison with the Macintosh and the Lisa. Then Macintosh and the Lisa have the capability of handling data significantly faster than the IBM PC or the IBM PCjr, the PCs family computer.

The Macintosh was released soon after the IBM PCjr was. Both machines had been planned for about a year, and the speculation about them had been endless. As always, IBM got the best part of the word-of-mouth campaign. The PCjr was officially released with tremendous advertising campaigns, press showings, and all of the other hoopla that goes along with a worldwide gala event. IBM expended tremendous energy on the PCjr campaign, while Apple was putting its energy into another area altogether.

The Macintosh project was nearing completion but Apple was keeping a low profile. Apple's machine would promote itself. Apple delivered a "Pocket Rolls Royce" at Volkswagen prices, a hard buy to pass up. In early January of 1984, demo units were sent to Apple dealers across the country. Dealers found them extremely easy to market. All the dealer has to do is set up the unit, activate it, load one of the application programs, and let the potential buyer have a free hand with it—because he needs almost no explanation at all. If dealers tried this with any other computer on the market, they would end up frustrating potential buyers.

The Macintosh is also easy to store and easy to transport, because it's so tiny. The main unit contains its own high-resolution monitor in a package that measures a little over 13 inches in height, 10 inches in width, and approximately 11 inches in depth. Additionally, the whole system, including keyboard and mouse, weighs in at a little less than 20 pounds. Dealers won't have to allot a great deal of shelf space to the machine, and users won't have to acquire much muscle to carry it home.

SUPER ROMS

As sophisticated as the Macintosh hardware is, one cannot say enough about the internal programs contained in the ROM chips. Unlike most computers whose operating systems are on disk and must be loaded into the machine, the Apple Macintosh's operating system is stored in the ROM. ROM stands for *read-only memory* and consists of a tiny integrated circuit device that can store thousands of bits of information.

The Macintosh ROMs contain over 500 programs that set up an extravaganza of functions it is

capable of performing. Unlike most computers, when you activate the Apple Macintosh without a disk, because of its ROM programs, it can still do quite a bit. Most microcomputers today contain between 8K and 32K of ROM; the Apple Macintosh contains 64K of ROM. This is more memory than some computers contain in both ROM *and* RAM, (*random-access memory*) combined. RAM is integrated circuit storage as well, but RAM chips receive information from the user (usually via the keyboard) or from disk storage. RAM chips can be reprogrammed at will, while ROM is fixed and cannot be changed.

For those of us who have been around computers for quite some time, a machine that contains 64K of ROM is extremely impressive, because a tremendous amount of program data can be contained in 64K of memory. Still, what you get from this 64K memory module is equivalent to at least twice what you would expect. The Apple engineers and programmers must have worked themselves sick to use memory so efficiently. The programs contained in this 64K ROM have the capability of what one would expect from 128K or more. Admittedly, almost any program that is written in any language can be shortened by making better use of the language, but undoubtedly, thousands of hours have gone into reducing *code* (programming) length to allow such extensive capabilities to be contained in such a small amount of memory.

MACINTOSH CRITICS

Although the Macintosh has so much ROM and so much in general, most professional programmers tend to shy away from its "menu-driven" concept. Certainly, most of the programs they write for individuals to use will contain menus, but they find using a menu while in the conceptual stage of writing a program can definitely slow down the process. However, many of these complaints may be coming from individuals who are fighting a change simply because it is a change. I overheard one program development manager remark that he would never buy a computer that displayed a trash can on the screen. This is the case with Lisa as well as Macintosh. If I could have asked him a question, it would have been "Why not?" I can think of no reason why someone would prefer to type an ERASE or KILL command at the keyboard in order to delete a disk file when it's far simpler to use the mouse to drag the file (by name) down to the trash can and "throw it away."

One must realize that computers and the people who program them are all part of a new industry. Some people consider themselves above all the average individuals who don't know anything about computers. Many of these persons have added to their mystique by implying that their work with computers is so complex that the average individual could never understand it. This is not true. Computer programming is no more mysterious than operating a restaurant, building a house, or being a plumber. All of these fields are different, and each requires its own type of basic training. Therefore, if a computer can be made that allows more people to use it (including programmers as well as the other users), then I say all the better.

THE COMPUTER FOR EVERYONE

Reasonably priced at $2,495, the Macintosh computer is certain to sell well, because it is within the price range of a large number of individuals. And the Apple ImageWriter printer can be purchased with the Macintosh for a total of $2,995 for the entire system, and its available audience is even larger. There would be no sense whatsoever in offering a computer for everyone that almost no one could afford.

Unlike the original Lisa, the Macintosh is going to be supported by a bevy of commercial software and peripheral companies. Over the next few years, a lot of software will be offered and, of

course, the peripheral selection should become equally plentiful. There is always hesitance to buy a new product we cannot be sure will last. This is especially true of microcomputers, since if there is no software or peripheral support, a great machine can be rendered totally useless. Many thousands of persons who purchased very expensive business computers in the middle and late seventies are now finding it necessary to try and unload them as scrap because the machines are obsolete and no longer supported.

I don't believe this will be the case with the Macintosh. The machine itself is certainly state-of-the-art and would seem to offer what other popular personal computers do, plus a lot more. What makes me feel most secure about the Macintosh is the overall concept of an inexpensive, high-level microcomputer that provides extremely efficient human interaction. In other words, a person with no computer expertise can be given a few minutes of instruction on the use of *MacPaint*, for instance, and then proceed to draw and learn more about this machine through using it. This is not true of many other computers, although programs could be written to make it so. Let me also say that programs like *MacWrite* and *MacPaint* may be the best features of all. A computer can be high-level and maybe even ahead of its time, but it's the software available that really counts. Any other high-level computer could be almost as easy to learn if such comprehensive, easy-to-use programs as *MacPaint* and *MacWrite* were offered. The mouse also makes the human computer interaction easy, but it is not so unique that they're not available for other computers as well. However, no other software, peripheral, or computer manufacturer has ever made available the combination of software and hardware necessary to accomplish such easy interface. The people at Apple have, and they are to be commended for it. Unlike many other companies who develop a high-level product and then thrust it upon the market, Apple developed a high-level product and a high-level user interface before offering it for sale. If other companies would think of the inexperienced user and equate this person's importance with the electronic design of the machine itself, everyone would be much better off.

FEATURES OF THE MACINTOSH

Shown in Fig. 1-2, the Apple Macintosh is a new kind of personal computer. It pretends the screen is a desk top and uses those familiar working methods to establish a new, more comfortable relationship between the user and the computer.

Personal computers that use current technologies present various and confusing interfaces each time the user changes applications. Macintosh does not. Familiarity and consistency are its keys. Macintosh applications offer methods the user already knows to perform tasks. Regardless of which task, the user approaches each application in the same manner.

At the heart of Macintosh is a 32-bit microprocessor, the Motorola MC68000. The 32-bit architecture and speed support the Macintosh graphics-based user interaction system. Macintosh uses a 9-inch, high-resolution (512 by 342 pixels) bit-mapped display. The entire system is extremely compact. See Fig. 1-3.

The Macintosh toolbox, the built-in software that controls its user interface system, has more than 480 separate software routines and is stored in the 64K of ROM discussed earlier. It contains 128K of RAM—enough room for more than an eighth of a million characters.

Application software and information are stored on the small disk in the Macintosh computer's built-in 3½-inch disk drive. The disk can store up to 400K of data, or about 100 double-spaced typewritten pages of information. In addition, the disk is small enough to fit in a shirt pocket

Fig. 1-2. Apple Computer's new Macintosh: extraordinary computing power and exceptional ease of use. Macintosh's main unit, which weighs less than 17 pounds, contains a 32-bit microprocessor, a built-in 3½-inch disk drive, a 9-inch black-on-white display, 64K of ROM and 128K of RAM. Equipped with a detachable keyboard and a mouse pointing device, Macintosh fits quickly and easily into the work style of business people, professionals, and students. The suggested retail price for Macintosh is only $2,495. (Courtesy of Apple Computer, Inc.)

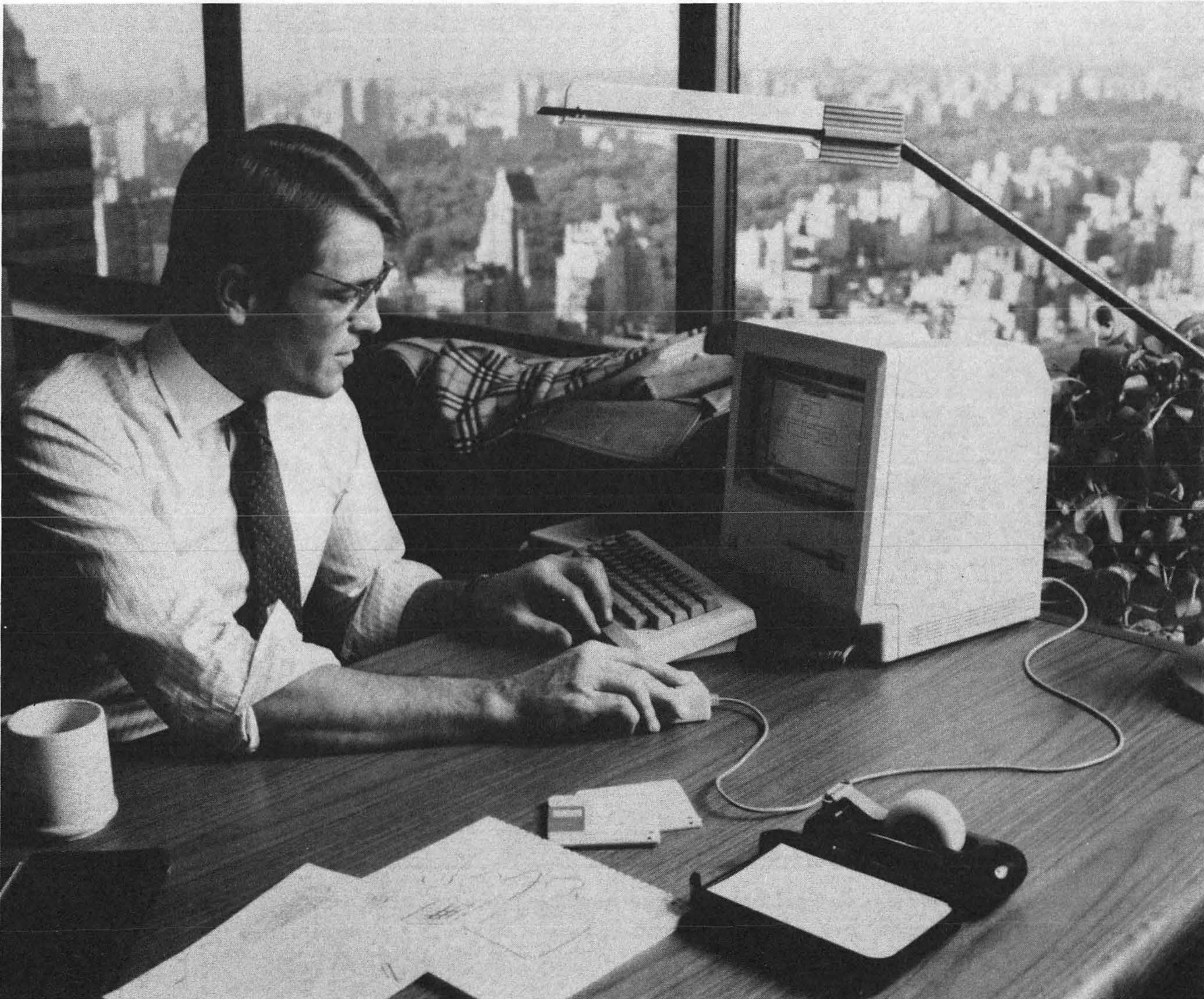


Fig. 1-3. Macintosh can be mastered in just a few hours and takes up only a small amount of desk space. It allows businesspeople, professionals and students to work in familiar ways, but more quickly and efficiently. (Courtesy of Apple Computer, Inc.)

and is protected by a hard plastic case. Another 400K of storage are available with an optional external floppy disk drive.

Macintosh has two RS 232C/RS 422 serial ports for attaching a printer and peripheral communications devices such as a modem; another port for connecting the optional external disk drive; and an audio system that has a range of more than 12 octaves, is capable of producing polyphonic pitches, and can replicate human speech.

ITEMS ON THE MACINTOSH DESK TOP

The Macintosh user works with icons of familiar office objects such as file folders, trash cans, and stacks of paper displayed on the Macintosh screen. Some icons represent tasks such as writing, moving documents, storing documents, or throwing documents away. Using the mouse, the user points to a desired function and selects it by pushing the button on the mouse.

The *Finder* is the application that allows users to organize their data, documents and tasks just as the user would on a desk in the office. Icons appear as documents, folders, or disks with their titles next to them. To gain access to a document, the user need only select the appropriate icon and the document is opened. Later, when the document is saved, it reverts back to icon form and is placed in its original location on the screen.

Users may organize the desk top in any way they choose. It includes such desk accessories as a calculator and a notepad. These are mini-applications that run concurrently with any other application. The machine is capable of running several more of these accessory programs, and Apple or third-party software developers may offer additional desk accessories in the future.

Each application places a menu bar at the top of the display to the user of the various operations that can be performed with it and the tools available in that particular application to perform them. Once an application has been selected, options for performing specific tasks appear through a pull-down menu. For example, before printing a document, a user can change such features as font style and size, margin justification and line spacing.

Macintosh can have several *window* display areas on the screen at one time. Each window may display different information. Using several Macintosh windows, a user can view several files at once. The information in a window can be a user's document, an error message or a request for more information. When several file folders are stacked on top of each other to conserve space, and as more windows are placed on the Macintosh desk, the windows overlap. Those in front partially cover those behind them. The sizes of the windows can be changed and windows can be split to show different parts of one document.

Information from one window or application can be moved to another. This *cut and paste* feature (Fig. 1-4) is useful for preparing reports that require both text and graphic data, each of which may have originally been prepared in another application.

Macintosh also supports Apple's interconnect equipment, AppleBus. A small, local equipment-sharing bus, AppleBus lets Macintosh and Lisa 2 users share such peripherals as printers and disk drives via a standard twisted-pair cable. (See Fig. 1-5.)

Macintosh's data communications capabilities give its users low-cost access to information stored in mainframe computer data bases. To do this, Macintosh can emulate a terminal in a mainframe communications network. Macintosh can currently emulate a variety of popular terminals, including the VT-100, VT-52 and Teletype for nonIBM mainframe communications, and the IBM 3277 and 3278 terminals via the Apple-Line product.

Macintosh was designed to conform easily to individual countries' local requirements and is marketed internationally. For example, the Macintosh

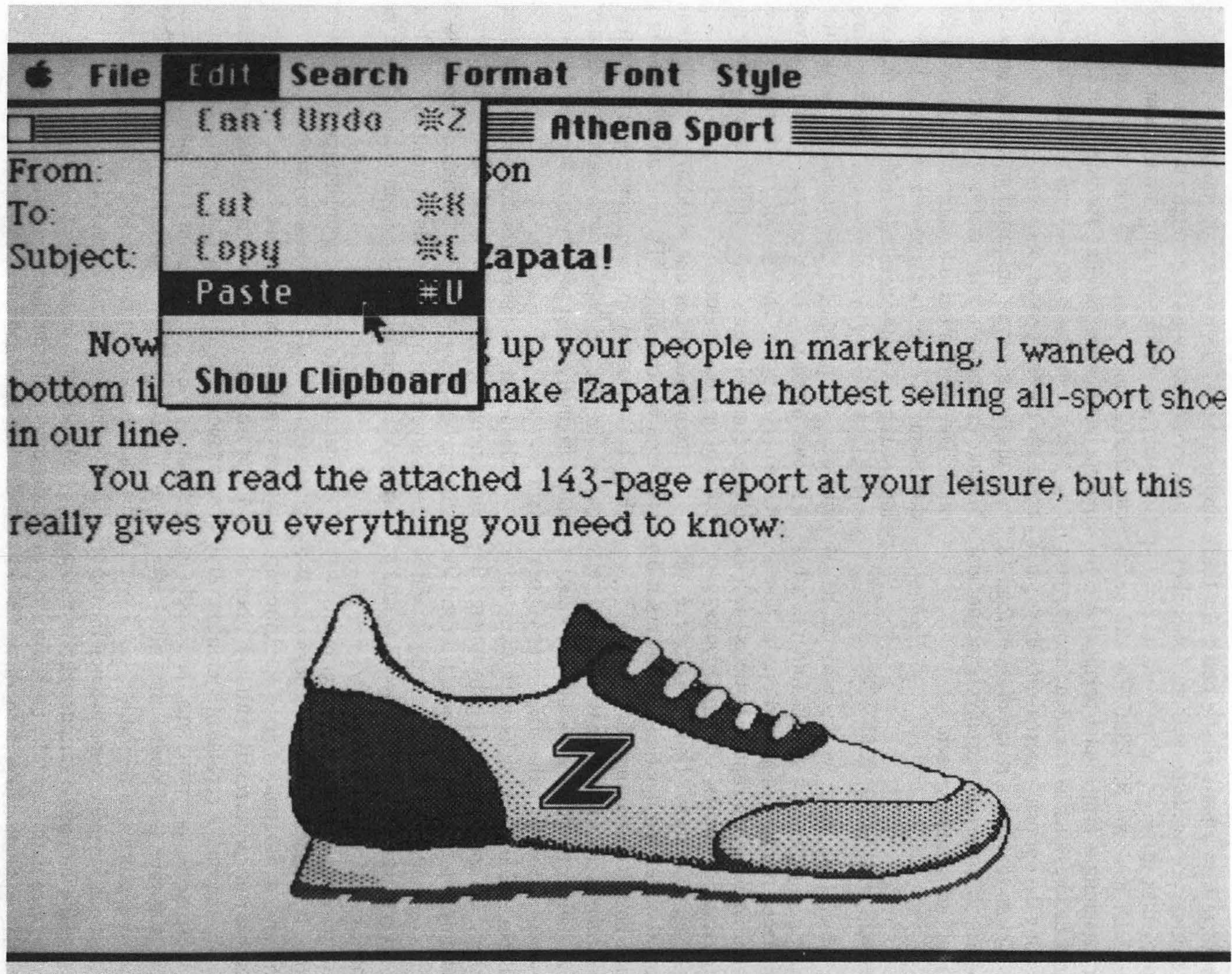


Fig. 1-4. Memos takes on a new look with Apple Computer's Macintosh software. Now along with typing your memo, you can illustrate it with graphics. Portions of text or graphics can easily be moved around within one document or "cut" from one document and "pasted" into another one.

Fig. 1-5. Apple Computer's new Lisa 2 (left) and Macintosh are part of an expanded family of compatible products called the Apple 32 Supermicro System. Products in the family incorporate Lisa Technology, 32-bit microprocessors, high-resolution bit-mapped graphics and mouse pointing devices, and all run Macintosh software. (Courtesy of Apple Computer, Inc.)

MACINTOSH SPECIFICATIONS

PROCESSOR: MC68000, 32-bit architecture, 7.8336 MHz clock frequency

MEMORY:
128K bytes RAM
64K bytes ROM

DISK CAPACITY: 400K bytes per formatted disk, 3½-inch diameter hard-shell media

SCREEN:
9-inch diagonal, high-resolution,
512-pixel by 342-pixel bit-mapped display

INTERFACES:
Synchronous serial keyboard bus
Two RS232/RS422 serial ports, 230.4K baud maximum (up to 0.920 megabit per second if clocked externally)
Mouse interface
External disk interface

SOUND GENERATOR: 4-voice sound with 8-bit digital-analog conversion using 22 kHz sampling rate

INPUT:
Line voltage: 105 to 125 volts ac, rms
Frequency: 50 to 60 Hz
Power: 60 watts

KEYBOARD:
58 key, 2-key rollover, software mapped
optional numeric keypad

MOUSE:
Mechanical tracking, optical shaft encoding
3.54 pulse per mm (90 pulse per inch) of travel

CLOCK/CALENDAR: CMOS custom chip with 4.5 volt (Eveready No. 523 or equivalent) user-replaceable battery backup

SIZE & WEIGHT:	Weight	Max. Height	Max. Width	Max. Depth
Main Unit	7.5 kg (16 lb, 8 oz.)	344 mm 13.5 inches	246 mm 9.7 inches	276 mm 10.9 inches
Keyboard	1.2 kg (2 lb, 8.5 oz.)	65 mm 2.6 inches	336 mm 13.2 inches	146 mm 5.8 inches
Mouse	.2 kg (7 oz.)	37 mm 1.5 inches	60 mm 2.4 inches	109 mm 4.3 inches

ENVIRONMENT: Temperature:operating: 10<<>>C. to 40<<>>C. (50<<>>F. to 104<<>>F.)
storing: −40<<>>C. to 50<<>>C. (−40<<>>F. to 122<<>>9 F.)

Humidity, all conditions: 5% to 90% relative humidity
Altitude: 0 to 4615 (0 to 15,000 feet)

Fig. 1-6. The Apple Macintosh specifications. (Courtesy of Apple Computer, Inc.)

user interface can display the date in a month-day-year format for the United States, or it can be changed to day-month-year for Europe. Additionally, all connectors on the Macintosh cabinet are labeled with internationally recognized symbols, rather than with words. This simplifies attaching printers or additional mass-storage devices. Further, Macintosh ROM uses no English code, making it easy for a translator to adapt the software to any language. This can be accomplished within a few hours.

A complete outline of the Macintosh specifications is shown in the chart in Fig. 1-6.

Everytime you insert a disk in your Macintosh and turn it on, a *window* will appear on the screen. In the bottom right border of the window you will see a shaded area containing arrow pointers. There's also another arrow on the screen. This is your mouse pointer and it is moved across the screen by sliding the mouse across your desk or tabletop. The arrows contained in the special border section are for scrolling the window left and right or up and down. If you want to scroll your window from bottom to top, simply place the mouse pointer on the arrow icon at the bottom right border. When you click the mouse, your file list is scrolled upward (Fig. 1-7). To move it back to the beginning again, place the mouse pointer on the top right arrow and click again. To scroll from right to left, place the pointer on the right-hand arrow in the bottom margin. The other arrow will reverse this direction (Fig. 1-8).

To actually enter the program, place the mouse arrow in the block in the upper left corner of the window. Now, click twice. Your disk drive will begin to whir and after a few seconds, you'll be in the program.

SOFTWARE FOR THE MACINTOSH

Apple is introducing the Macintosh with what it believes are the two software packages that make hardware immediately useful—a word processor and a graphics ilustration package. Microsoft Corp.'s *Multiplan* spreadsheet program is also immediately available.

Apple is also supporting the work of independent software vendors to develop Macintosh programs. Apple began to provide support more than two years ago and continues strongly. Apple expects that its developer support program will make at least 500 software packages available for Macintosh by the end of the year. The software should include productivity applications, communications packages, educational tools, other special applications, and games. Currently, Lotus Development is rewriting its popular *Lotus 1-2-3* for Macintosh, and Microsoft Corp. executives say they expect nearly 50 percent of their company's application revenues in 1984 to come from Macintosh software. Currently, more than 100 major software and peripheral vendors are developing packages for Macintosh, including Software Publishing Corp., Continental Software, Ashton-Tate, and Sorcim Corp.

Software from these major developers will make Macintosh one of the best machines available to business customers as well as students and home users. To reach business customers, Apple plans to sell Macintosh through the 3,000 dealerships already selling Apple products. The company expects that nearly 85 percent of Macintosh sales will come through retail channels.

To supply software to Apple's second major market for the Macintosh, college students, Apple is entering into sales relationships with prominent U.S. colleges and universities where colleges will develop courseware on Macintosh. Software and other key developments made at each institution will be shared with the others and with Apple. Twenty-four leading institutions, including Harvard, Brown, and Stanford universities, the University of Michigan, and the University of Chicago, have joined the Apple University Consortium.

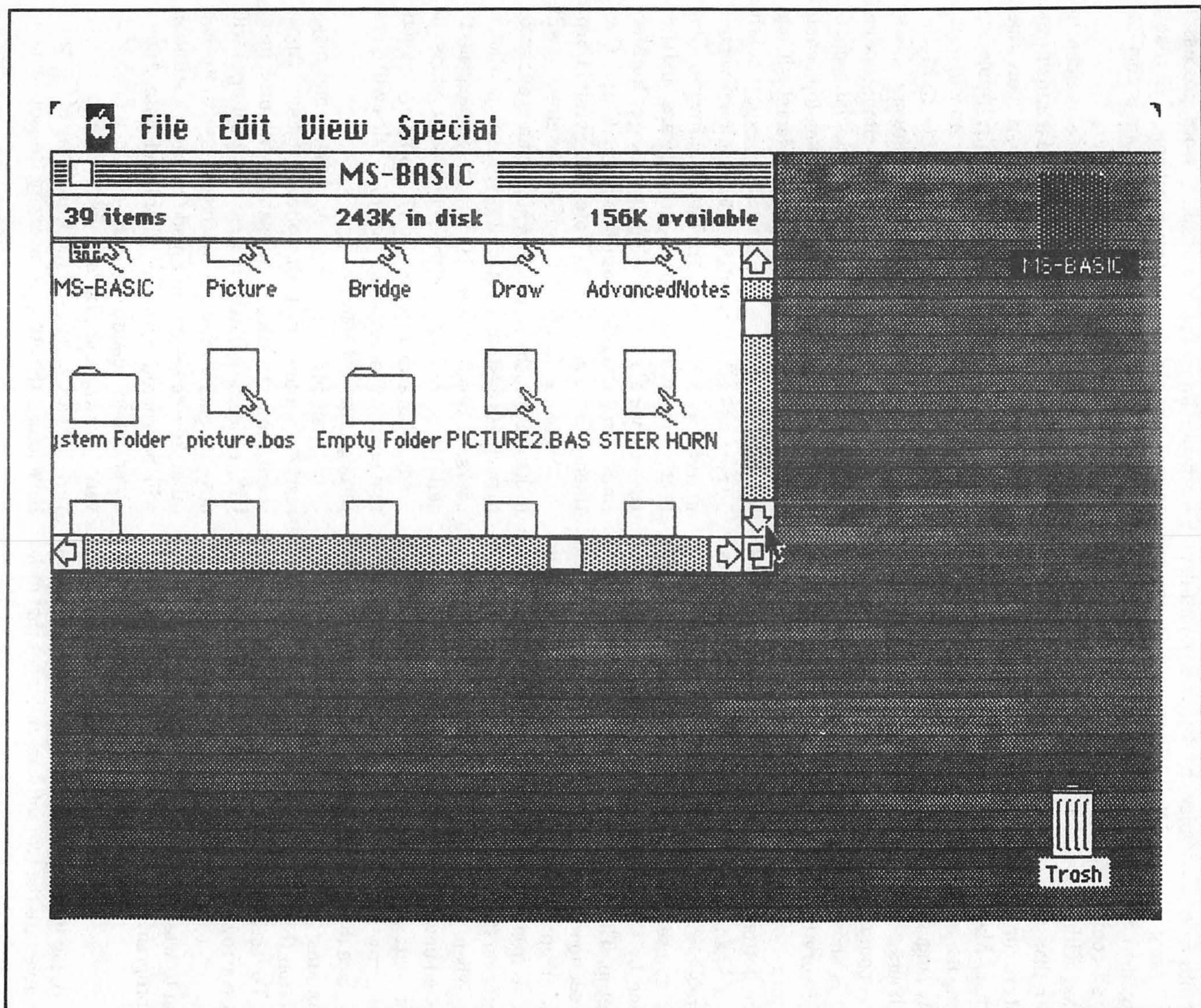


Fig. 1-7. Scrolling the window upward.

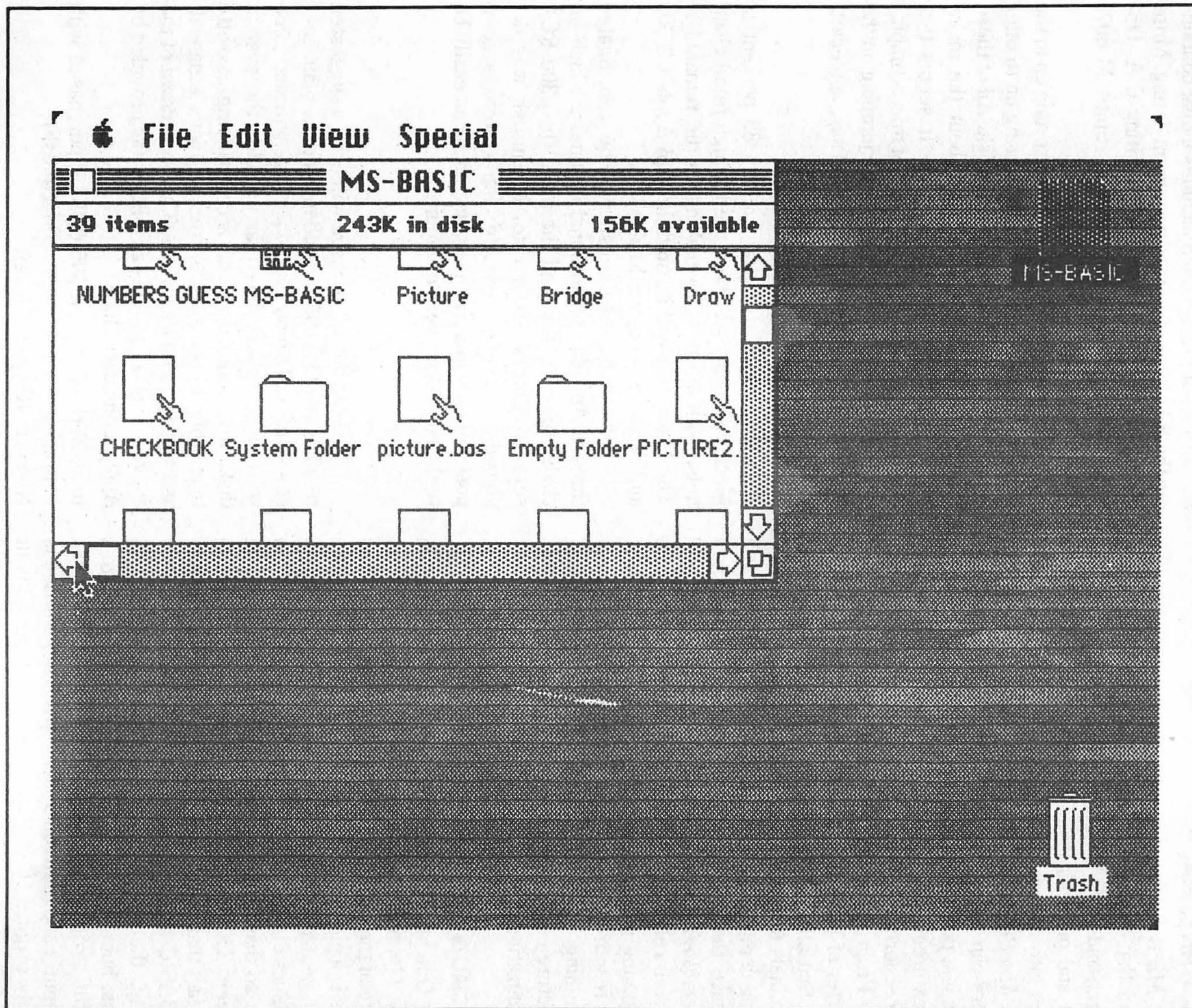


Fig. 1-8. Scrolling from left to right.

MACWRITE, MACPAINT AND MORE

MacWrite is a versatile word processing program that features multiple fonts and font sizes, search-and-replace functions and the ability to cut text and pictures from other programs and paste them into memos or reports.

MacPaint is a powerful graphics program. Users can choose from an array of tools, such as brushes, pencils and erasers, and a large selection of textures and shapes to create an endless variety of free-form or structured images.

Programs to be released by Apple in the first quarter of 1984 include *MacTerminal*, which is the one that allows Macintosh to emulate a variety of terminals for access to a variety of mini and mainframe computers. In the second quarter Apple will release the *Macintosh Pascal* and the *Macintosh Assembler/Debugger* programming packages. Third quarter releases are the *Macintosh BASIC* programming package; *MacProject*, for creating flow and resource usage charts; *Macintosh Logo* programming language; and *MacDraw*, for creating structured graphics such as line graphs, pie charts, organizational diagrams, and flowcharts.

SPECIALIZED SOFTWARE

One interesting software offering comes from C.A. Optographics. Figure 1-9 shows a screen write of the highly specialized programs they offer. This is a picture of the author that was made from a black-and-white glossy photograph. C.A. Optographics digitized this picture and then loaded it on disk so that it may be displayed on the Macintosh screen. The unusual feature of this service lies in the fact that the picture program is written entirely in BASIC and can be modified by the user. Certainly, there is nothing new about digitized pictures, but a digitized picture that is converted to a BASIC program that can be input to any Macintosh computer and then run to display the same image is highly original. C.A. Optographics charges about $35 for this service, which includes a disk containing the BASIC program and a program listing. More information can be obtained by writing C.A. Optographics, 720 Virginia Avenue, Suite M-607, Front Royal, VA 22630.

Also, several companies have sprung up in the Midwest that are reputed to be gearing up to offer some unusual programs for Macintosh. All of these rumors are in addition to those about the major software and peripheral suppliers. It seems that Macintosh is ushering in a new era of microcomputers and a lot of reputable people are jumping on the bandwagon . . . or, perhaps I should say, applecart.

WHERE TO FIND A MACINTOSH

Apple estimates that initially 85 percent of Macintosh sales will be made through retail channels, with direct sales making up the remainder. The Macintosh will be sold through Apple's 3,000 authorized dealers worldwide.

Service for Macintosh will be coordinated through Apple's conventional channels, which include Apple dealers and the more than 300 RCA service centers nationwide. Macintosh was designed for simple servicing; the system is composed of only four modules, and each can easily be replaced in the event of failure.

Price

The Macintosh package will have a suggested retail price of $2,495 and will include the main unit, keyboard, and a mouse. The package will also come with an accessory box that includes the system disk, a learning disk and accompanying cassette tape, a blank disk, a power cord, an owner's manual, and a programmer's switch. The breakdown of the suggested retail price is as follows (as provided by Apple Computer, Inc.):

ImageWriter printer	$ 595 ($495 if purchased with Macintosh)
Numeric Keypad	$ 129

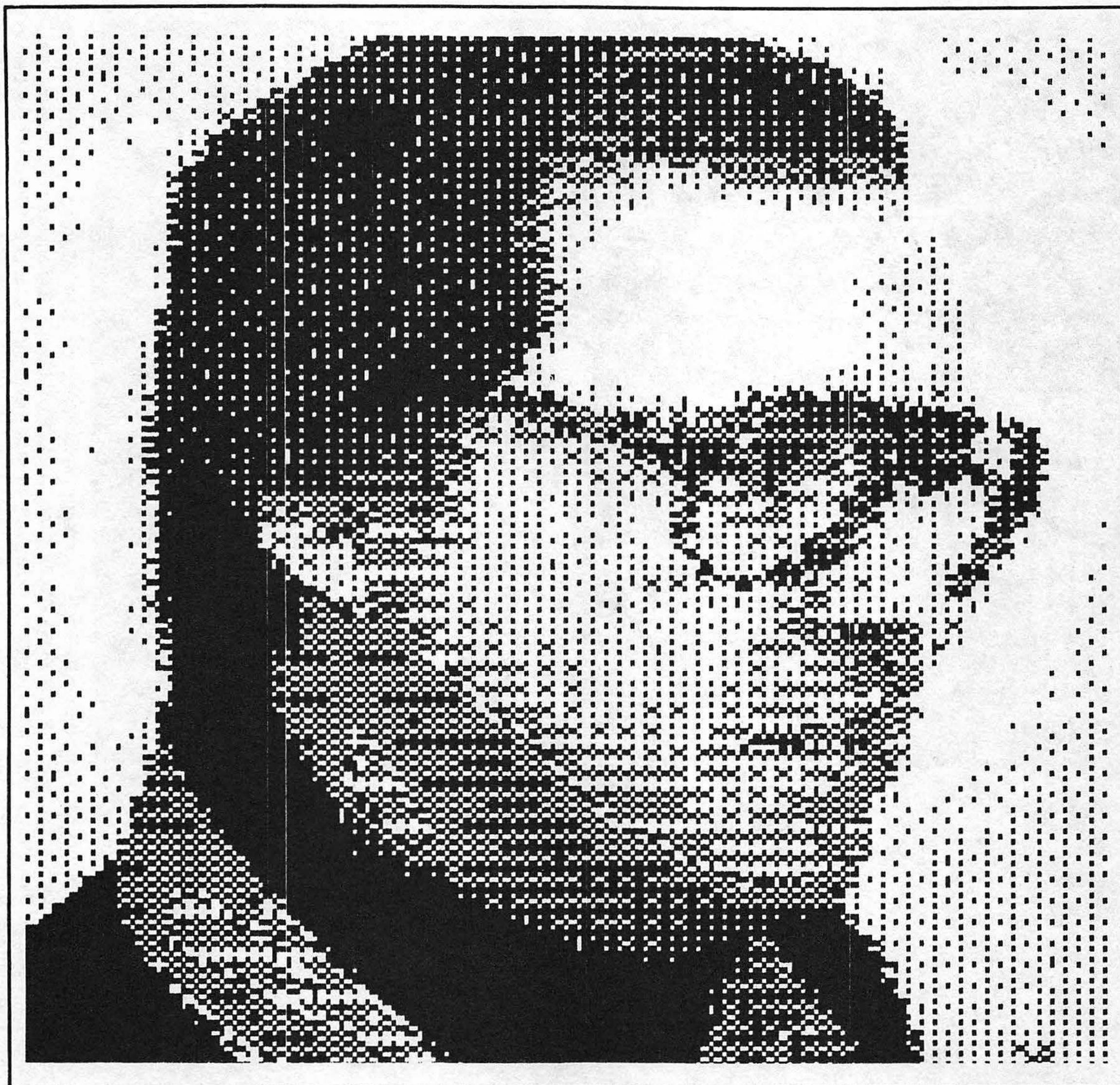


Fig. 1-9. Digitized image of the author. (Courtesy C.A. Optographics, Inc.)

Modem 300	$ 225
Modem 1200	$ 495
Carrying Case	$ 99
3½-inch disk box (10 disks)	$ 49
MacWrite/MacPaint	$ 195 (included free with each Macintosh during the introductory period)
External Drive	$ 495

Fig. 1-10. Three new Lisa (TM) personal computers were introduced by Apple Computer, Inc. in January 1984. All three systems share the same look and feature advanced microdisk technology. The Lisa 2, the Lisa 2/5, and the Lisa 2/10 all have 512 kilobytes of internal memory, expandable to one megabyte. In addition, the Lisa 2/5 has a 5-megabyte external hard disk drive and the Lisa 2/10 features a built-in 10-megabyte hard disk drive. (Courtesy of Apple Computer, Inc.)

THE LISA-MACINTOSH CONNECTION

Since the original announcement of the Macintosh many persons have tried to predict what Macintosh will do for the Lisa 2 (the new version of Lisa that was released by Apple in 1984), and what Lisa 2 will do for Macintosh. The original Lisa from Apple, as pointed out earlier, was not a success; but Lisa 2, shown in Fig. 1-10, is an upgraded version that offers everything the old Lisa did and more. Also, it costs less. Many have speculated that the success of Macintosh, which is really a cut-down version of Lisa 2, will spur sales of Lisa. Also, since the two machines are compatible in many ways, software support for one will often mean software support for the other. During the old Lisa era some persons felt that Apple might scrap Lisa altogether, but instead, this company is offering a new version along with Macintosh.

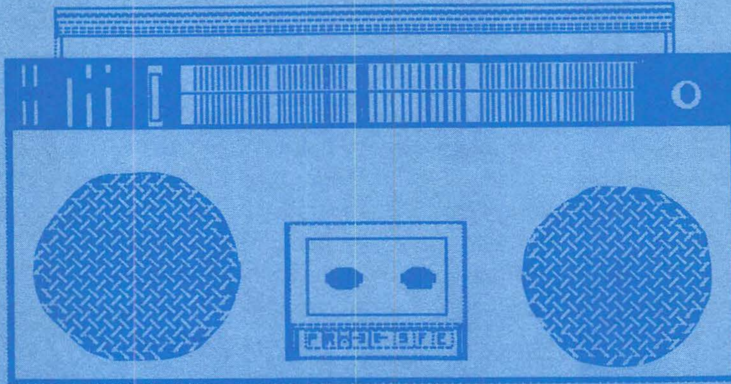
On January 24, 1984, Apple Computer announced its Lisa 2 series, incorporating three higher-performance versions of the original Lisa computer. Ranging in price from $3,495 to $5,495, the expanded Lisa 2 computers give users more flexibility in both memory and mass storage. Together with the new Macintosh, the Lisa series forms the basis of the Apple 32 Supermicro family of advanced desk top computers, a series of computers that have new paths for the future of computers.

SUMMARY

There can be no doubt that the Macintosh and the Lisa computers are tied together. The fact that the Macintosh operating system is available for Lisa indicates that programs that will run on Macintosh will also run on Lisa. Undoubtedly, Apple hopes to introduce many individuals to the computer through its low-priced Macintosh line. Later, many people may decide to move up to a Lisa 2, which can use many of the same peripherals and is not much more expensive than the Macintosh. The less-expensive version of Lisa 2 costs only $1,000 more than Macintosh, whereas the more expensive version costs only $2,000 more. This may mean that many experienced users of computers may elect to go with Lisa rather than Macintosh, although it is almost a certainty that Macintosh will be the bigger seller of the two.

This new series of microcomputers from Apple may mark the turning point for the entire microcomputer industry. If the ease-of-use concept really catches on, as I believe it will, you will see many manufacturers upgrading their machines or at least offering software packages (and undoubtedly, the mouse) that will raise their ease-of-use level.

Chapter 2



Using *MacPaint*

MacPaint is an electronic graphics palette for the Apple Macintosh by Bill Atkinson. It is available on disk as is *MacWrite*, the word processing program. They are both included with the Macintosh during its introductory period.

MacPaint allows you to write, draw, paint, enlarge, shrink, copy, alter, and do almost anything imaginable on the screen—all you need is the mouse. If you can't draw a straight line on paper, you can still draw with *MacPaint*. *MacPaint* makes use of the QuickDraw routines contained in Macintosh ROM. Because they are ROM-based, the routines operate with the kind of speed necessary for a high-level electronic palette program.

As is the case with most Macintosh applications, *MacPaint* displays a menu bar at the top of the screen and a bevy of drawing tools. The *MacPaint* screen is shown in Fig. 2-1. You can see the menu bar at the top, the drawing tools on the left, and at the bottom, a number of patterns that can be used to "color in" outlined objects.

All you have to do is place the cursor on the drawing tool you wish to use. For instance, you may want the pencil icon that is located on the fourth row in the graphics tool column in order to write on the screen as you would on a piece of paper. Move the mouse to place the cursor on the pencil; then hold the mouse button down while you roll the mouse in the direction you want the pencil to move on the screen. Let go of the mouse button once when you want to begin writing, then press it again and hold it while moving the mouse as you would a pencil. As the pencil travels across the screen, it writes a thin black line. With a little practice, you can write legibly. You can even sign your name as shown in the sample in Fig. 2-2. Each time you desire a different drawing tool, you simply move the pointer to that tool, click the button and hold it, and then move the mouse to bring the tool to the screen. You'll notice that, for most drawing functions, the arrow will convert to a crosshair when it is brought out to the writing area on the screen.

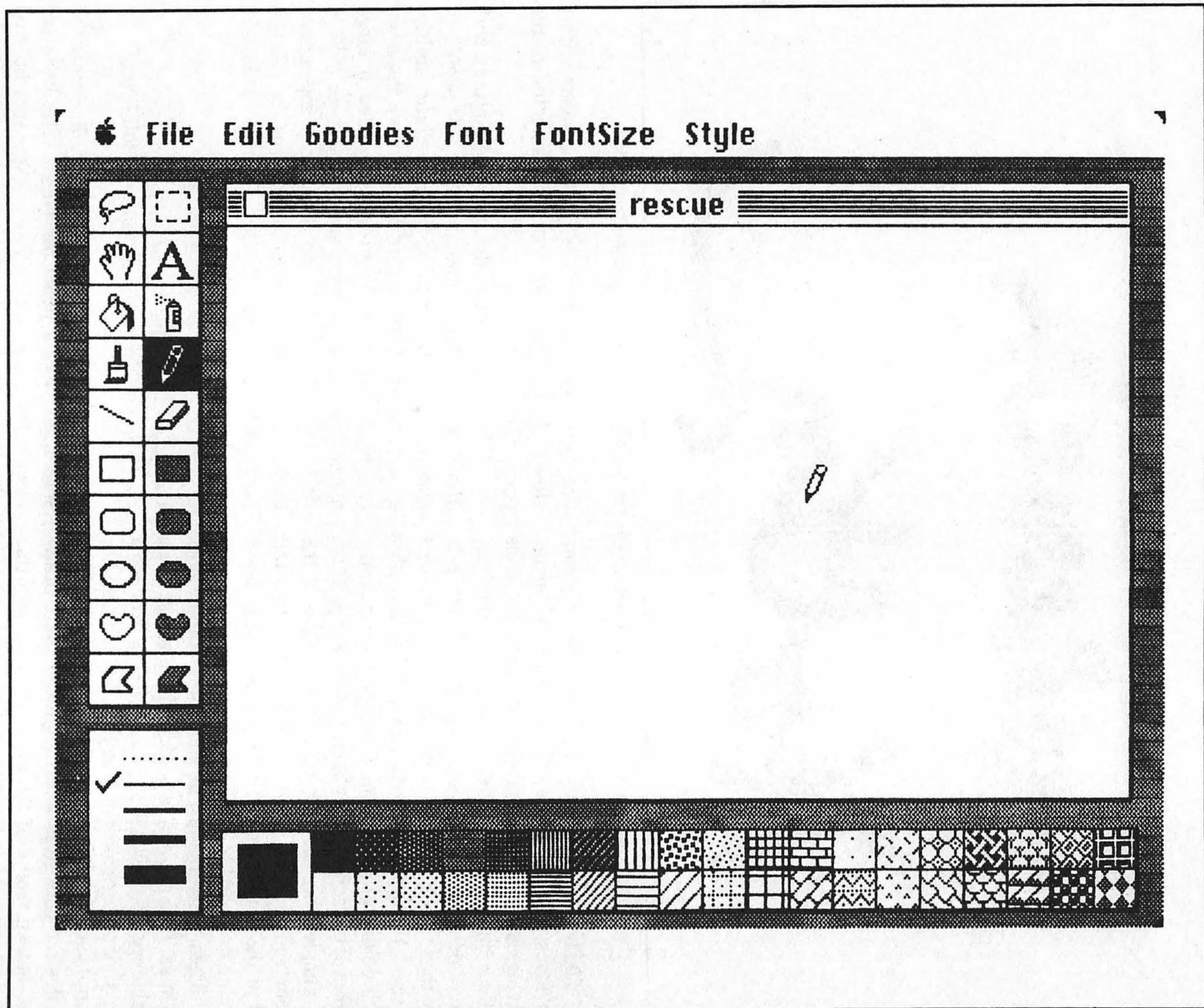


Fig. 2-1. The *MacPaint* Screen. The Title Window contains the name "rescue," the name of the image that is to be drawn.

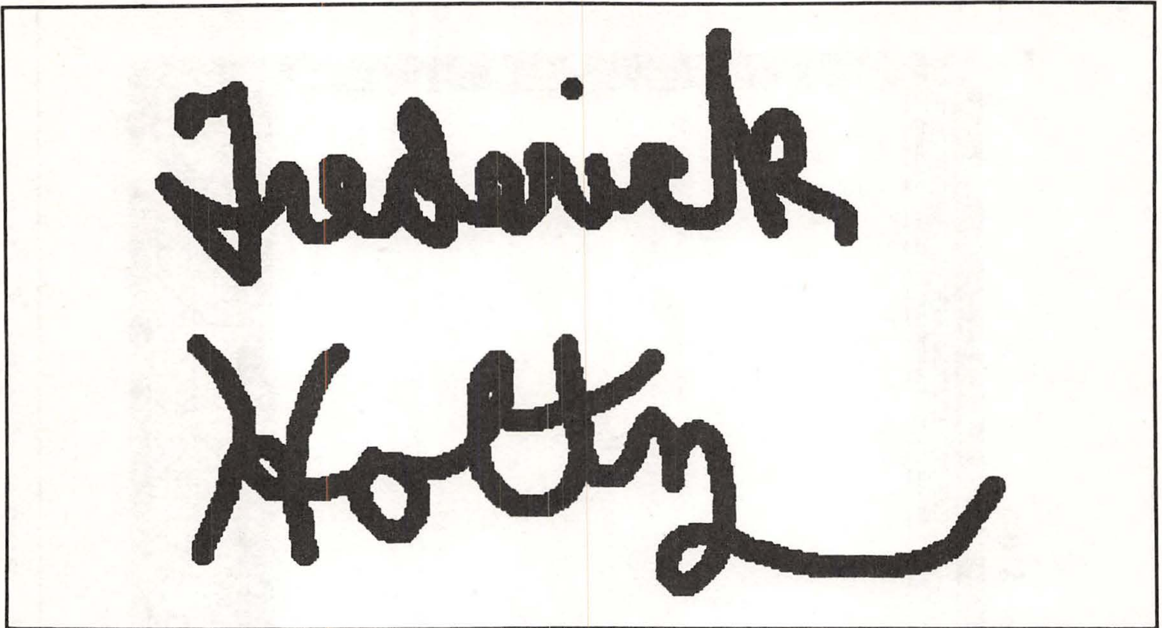


Fig. 2-2. The author's signature. With a little practice, you can easily sign your name using the pencil icon.

The bottom four rows of the tool columns contain icons for rectangles, circles, general image drawing, and a rubber band line effect. These icons produce line drawings that are not filled in with any type of design. Similar icons appear to their right, only these ones are filled. When the filled in ones are used, the drawings produced are filled in with a hatch color selected from the pattern bar at the bottom of the screen. To choose a different pattern, you simply drag the pointer to the bottom of the screen and click the option desired. On the far left end of the bottom bar is a larger box that shows which hatch pattern is currently being used.

These patterns are far more than simple dots and lines. One of them looks for all the world like a steel fence, as shown in Fig. 2-3. Others produce a slate roof pattern as shown in Fig. 2-4. This ability to produce realistic detail and a variety of effects enables the user to produce highly accurate drawings. Indeed, some of the pictures done with this program resemble *digitized* pictures, made with a closed circuit television camera connected to the computer.

While it is easy to draw simple pictures with *MacPaint*, drawing detailed pictures of people or complex objects requires some artistic ability, especially when using shading. Because the high-resolution Macintosh screen produces a large number of dots in a small amount of space (512 × 342), lines do not take on the jagged appearance they could on lower resolution screens. So you can draw perfectly formed circles, ellipses, and other curved objects with ease.

THE GRAPHICS TOOLS

The drawing tools include the following:

- The pencil produces thin lines on the screen just like a pencil on paper. The lines are usually black against a white background, but you

also convert the background to black and write in white. This applies to all the other drawing tools available as well.

- The **paintbrush** can be adjusted to one of 32 shapes or sizes. The paintbrush is used to "paint" on the screen by moving the mouse. It paints in the color or style clicked from the pattern bar at the bottom of the screen.
- The **paint sprayer** (represented by a spray paint can icon) squirts patterns on the screen in the same fashion a spray paint can does. The more you spray the area, the more filled in the area becomes.
- The **eraser** does just what its name implies. It doesn't just obliterate the entire screen (unless you want it to), but instead, it erases any particular lines you tell it to remove. When you click this icon, a small open box appears on the screen. Anything the box passes over while the mouse button is held down is erased, just as if it were the eraser on the end of a pencil.
- The **paint bucket** fills any drawing of any object with the current hatch style in effect. You simply click the paint bucket, drag it to the area to be filled, and click again-and an open area is filled with the chosen hatch style.
- The **hand** allows you to move the entire drawing page to a new location. You simply click the hand, place it on the screen, and move up, down, left, or right. The screen travels in the direction the hand is moving, and any objects on the screen naturally travel with it.
- The **rubber band line** can be started at one point on the screen and will then follow the arrow to another point. As you move the pointer, the line extends back to its original starting point.

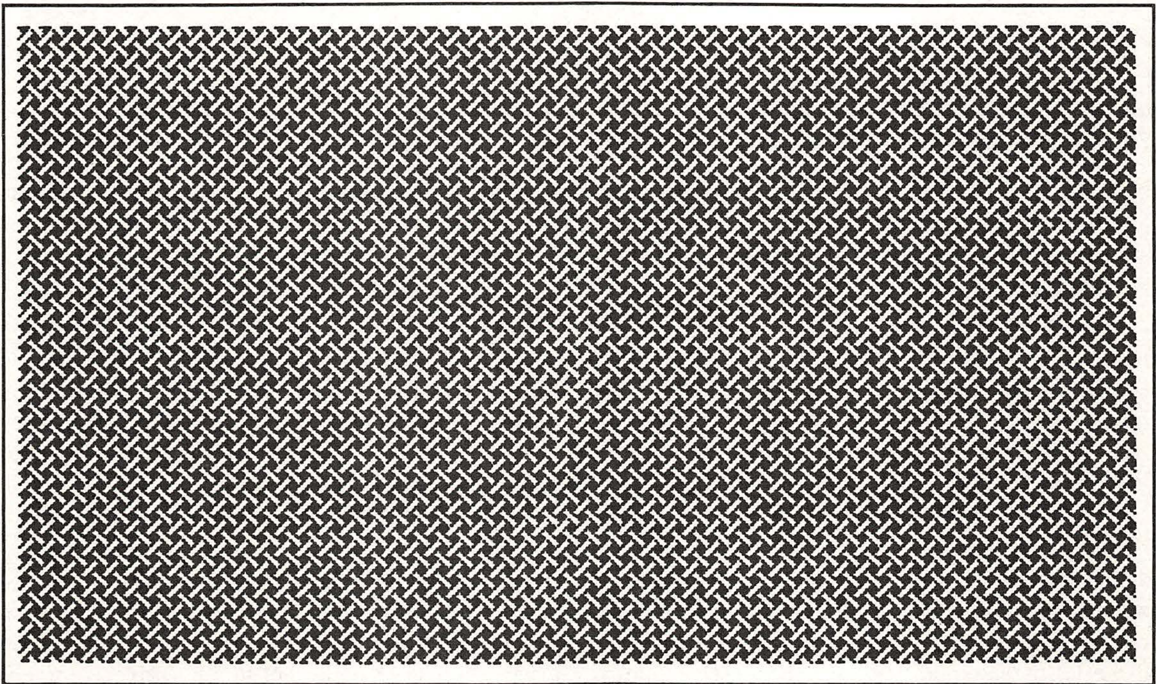


Fig. 2-3. One of the many patterns available in *MacPaint*. This one resembles a steel fence.

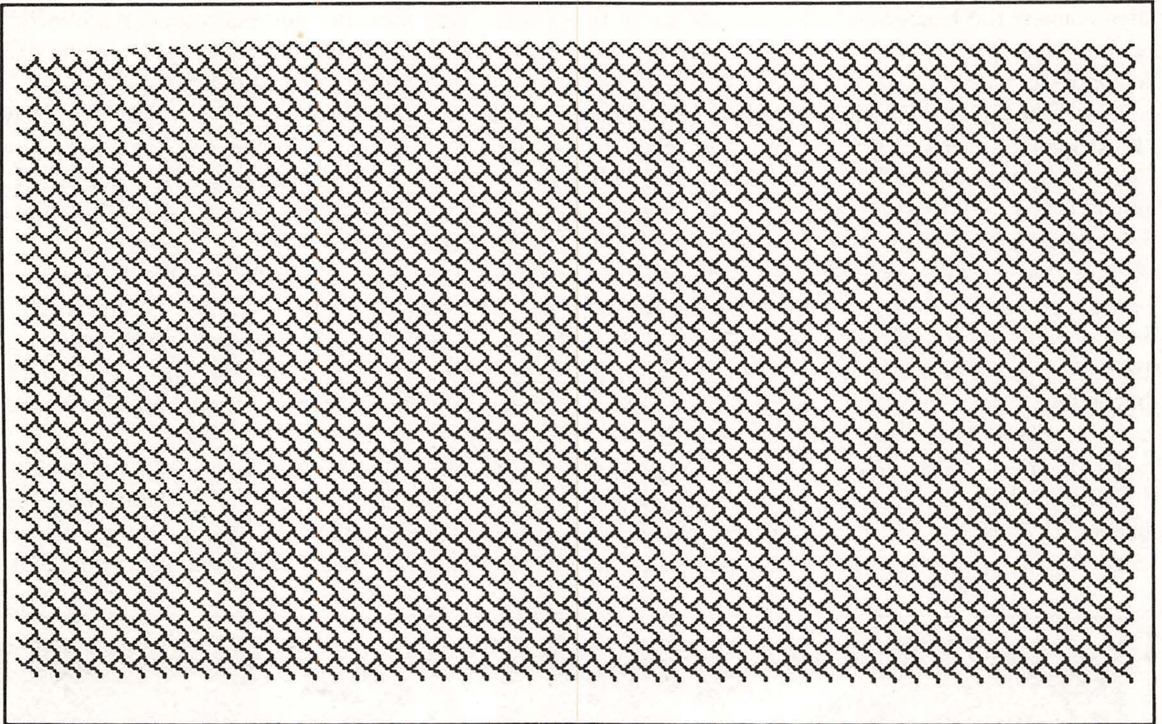


Fig. 2-4. Another pattern in *MacPaint* that resembles a slate roof.

The line is set by clicking the mouse. There is also another rubber band effect that can be used to draw objects that are then automatically filled in with the current hatch pattern.

● The **lariat** is used to cut out a portion of an image and move it elsewhere on the screen. When the lariat icon is clicked, you simply release the button and drag it to the screen. You then press the button again and hold. As you move the mouse, the lariat draws a line. You simply encircle the item or portion of the drawing to be removed and release the button. Move the lariat into the image that is cut away and an arrow will appear. When this happens, simply hold the mouse button down and move the mouse. The portion encircled will be moved in the direction of the pointer. When you have the cutaway portion properly positioned, release the mouse button and click the lariat icon again. The image is frozen.

● The **border icon** represented by a box made of dotted lines is used to form a border around a drawing. When this icon is clicked, a dotted line cursor appears on the screen. When you press the mouse button and move the mouse, a border is drawn in rectangular fashion. The border can be used to encircle an object or a portion of an object to allow it to be moved elsewhere on the screen. In this manner, it works very much like the lariat tool.

● The **rectangle** produces squares, rectangles, or lines on the screen. You have a choice of an open or filled rectangle. As mentioned, the open icon draws open images, while the filled version draws the same shape, but fills it with the current hatch pattern. Simply click the icon once and move

the mouse pointer to the screen. Place the crosshair in the place you want to begin the figure, click and hold, and then move the pointer to another area on the screen. If you pull it to the right of your starting point, the rectangle is formed toward the right of the starting point. If you pull it toward the left, the rectangle is formed to the left of the starting point. Depending on how you move the pointer, the image formed may be a straight line (vertical or horizontal), a perfect square, or any type of rectangle.

- The **round-cornered rectangle** produces a rectangular image with the four corners rounded. This image is shaped very much like a monitor or television screen.

- The **oval** draws circles or ellipses on the screen. It can also be used to draw straight lines by producing an ellipse whose top sector aligns with the bottom sector.

- The **general line maker** works just like the pencil icon, but uses a crosshair cursor to produce its images. The filled version will automatically fill in an object using the current hatch pattern.

- The **general rubber band line maker** that works just like the previous rubber band icon. However, the filled version automatically shades in an image using the current hatch pattern.

- The **keyboard interface** allows you to change the type to any of a variety of different styles and sizes. This icon that represents it is the letter "A." When the "A" is clicked and pulled on to the screen, a raised cursor appears. Place the cursor where letters are to be printed and then click. A flashing cursor then appears and letters may be typed via the keyboard.

In the bottom left corner of the screen, there is a line menu that allows you to choose the width of the lines to be drawn. These apply only to the rectangle, round-cornered rectangle, oval, general line, and rubber band line modes. Lines may be chosen to be very thick or pencil thin. The general line and rubber band line modes that are similar to the pencil and standard rubber lines make use of the line menu, whereas the others do not. A line width is selected by placing the pointer on the desired line (one of five) and clicking once. This is the mode that will be in effect for the icons selected until you click another line setting.

THE MENU BAR

The menu bar at the top of the screen is set up to provide easy access to machine capabilities. It allows you to do many of the things we've been discussing more efficiently. To access the information contained under each heading, simply place the mouse pointer on any of the seven menu bar options. Hold the button down. As long as you hold the button, you will see all the suboptions available to you. To access one of them, simply move your mouse downward until the arrow points to the desired option. Each time an option is passed over by the arrow, it will be highlighted in black. While the proper option is highlighted, simply release the mouse button and that function is executed.

The Apple Logo Menu

The first menu bar selection is designated by the Apple logo (appropriately, an Apple). This is alternately highlighted by a black block. Place the pointer here and press the mouse button. You will see several programs here, such as SCRAPBOOK, ALARM CLOCK, CALCULATOR, and CONTROL PANEL. Slide your mouse pointer down until CALCULATOR is highlighted. When you release the mouse button, the CALCULATOR program is run, which displays a calculator on the screen. To leave the CALCULATOR and use another option, simply place the pointer in the EXIT box at the top left of the calculator display and click once. This will return the screen to the previous selection format.

The FILE Menu

To view the suboptions in the FILE menu, place the pointer on FILE and hold the button down. The suboptions will be displayed as New, Open . . . , Close, Save, Save As . . . , and Quit. New simply opens a new screen. It erases any drawing currently in memory. Slide your pointer to New and release the button. You will be prompted as to whether or not you wish to save the current drawing you've done.

Access the FILE menu again and click the Open suboption. This tells the computer that you wish to start a new drawing. Again, you will be asked if you wish to save the one currently on the screen. A new window will then be displayed to ask you for a file name. All drawings must be stored in a file "folder" much like the kind in an office filing cabinet. Since each one must be stored, it must also have a file label, so that the computer can identify the file. The label is what is referred to as the file *name*.

The Open suboption is used to bring a drawing that was previously stored on disk back to the screen. When you input a file name, the file will automatically be read from disk and loaded into immediate memory. The drawing is then brought to the screen.

The Close suboption is similar to New, in that it is designed to erase the current drawing from memory. However, you are again prompted as to whether or not you wish to save the drawing, because you may have made changes to it after you *loaded* it back onto the screen. If you choose YES, the drawing will be saved with the changes you made. If you choose NO, the program is not resaved, and the original is still stored on disk.

The next suboption is Save. This one is simple. It saves the drawing currently in memory to disk. If it's a new drawing, you will be prompted to input a name for it. If it's one that was originally loaded from disk, since it already has a name, it is resaved under the same name, replacing the old version of the drawing on disk with the new version under the same name.

The next suboption is Save As . . . It works like Save, but gives you the option of changing the file name. For instance, you could load a drawing named PICTURE and save it using Save As . . . under a new name, such as HERPICTURE. However, the original MY PICTURE will still remain on the disk along with the newly named drawing.

The final option is Quit. When you select Quit, it tells the computer to exit and go back to the system menu. If there is a drawing currently in memory (on the screen), you will be given the option of saving it before leaving *MacPaint*.

As you can see, the FILE menu on the menu bar allows easy access to the disk.

The EDIT Menu

The *MacPaint* EDIT menu (Fig. 2-5) gives you the tools to cut, paste, and copy graphics from the screen. You can cut away portions of a graphic image and paste the portions elsewhere on the screen. This may be done using the lariat or border icons to determine which portions are to be cut or copied. The lariat is used to remove something (in conjunction with Cut), whereas the border is used to cut away a portion of the image for later pasting. Since the border may be increased or decreased in size, the image that is cut away may be enlarged or shrunk. You can opt for Copy in the EDIT menu to simply copy a portion of an image for placement elsewhere without its removal from the original image.

A separate portion of the EDIT menu allows you to invert the image, trace its edges, flip it horizontally or vertically, and rotate it by 90 degrees. To do any of these, you select the border icon and encircle the portion to be affected by the EDIT options. If you want to select these options for the entire screen, simply click the border icon

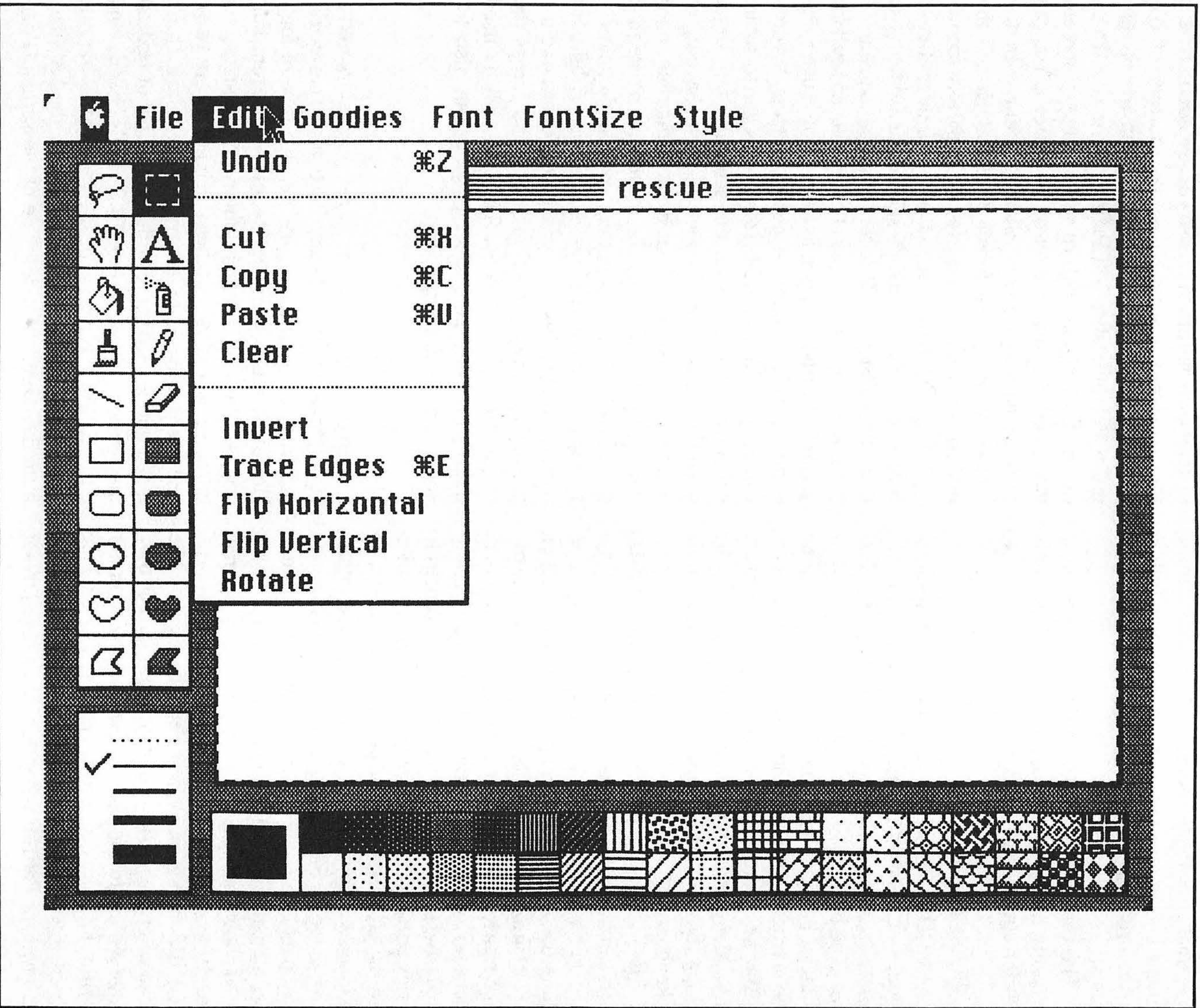


Fig. 2-5. The Edit menu in *MacPaint*.

twice. Each time the border is in effect, you will see a moving border shape appear on the screen, telling you the area that is to be affected by the EDIT options.

The Undo option simply returns the screen to its condition before your last screen change and is great for correcting mistakes.

The GOODIES Menu

The GOODIES menu (Fig, 2-6) is appropriately named, since it provides rapid access to some extremely sophisticated "goodies" available in *MacPaint.* The Introduction suboption calls up the various menu bars and explains what each does. The same is true of the Short Cuts option, which shows the Mac keyboard and how it relates to the image icons. Many of the functions brought about by clicking these icons can also be accomplished via the keyboard.

My favorite GOODIES suboption is Fat Bits, which automatically enlarges a portion of what appears on the screen. Say, for example I wanted to include the Apple logo (an apple with a bite taken out of it) in a drawing of the Macintosh. Let's say the image was to be very small and it was impossible to do any serious drawing, given the size limitation. All I'd have to do is click the border icon to encircle the tiny portion of the image I want to change; then go to the GOODIES menu and click Fat Bits. The tiny image is now enlarged many, many times at the center of the screen, and I can use the pencil icon to easily pencil in any image you want. At the same time, a small block in the upper left corner of the screen shows how the image will appear in normal size. When it's finished, I simply go to the GOODIES menu and click Fat Bits again. The screen will return to its original condition. The only change will be to the portion I was changing under Fat Bits. Fat Bits can also be accessed by clicking the pencil icon twice.

The Edit Pattern suboption can be used to set up your own pattern styles so you are not limited to those currently available at the bottom of the screen. Simply select a pattern from the pattern menu and then click Edit Pattern in the GOODIES menu. The original pattern appears on the screen, along with an enlargement of the pattern. You can remove dots by simply clicking them with the mouse. When you're finished, click OK, and the new pattern will be committed to the pattern menu.

The GOODIES menu also contains Brush Shape and Brush Mirrors suboptions. When you click Brush Shape, 32 different shapes appear on the screen, and you simply click the one desired for your paint brush. They allow the paint brush to be either circular, square, diagonal, horizontal, vertical, or sprayed. Brush Mirrors allows you to set up several paint brushes on the screen at one time, even though only one is seen. You can have up to eight paint brushes working at the same time; each will symmetrically copy the movement of the first paint brush. With this option, you can do kaleidoscopic patterns on the screen in a host of different patterns and styles. You can produce a tic-tac-toe pattern by drawing one horizontal line across the top portion of the screen. This one will be used quite frequently simply for playing, but also for some serious drawing work.

The FONT Menu

In the commercial art world, *font* refers to the style of the letters and numbers that you are working with. Each computer has its own type of font. However, the Macintosh offers nine different font styles. It also offers eight different font sizes, six different print styles (italics and boldface are two), and three alignment controls. There are three menu options that deal with the display of text on the screen. These are FONT, FONTSIZE, and STYLE. An easy way of displaying the various styles and fonts quickly is to select the "A" icon from the menu at the left side of the screen. Posi-

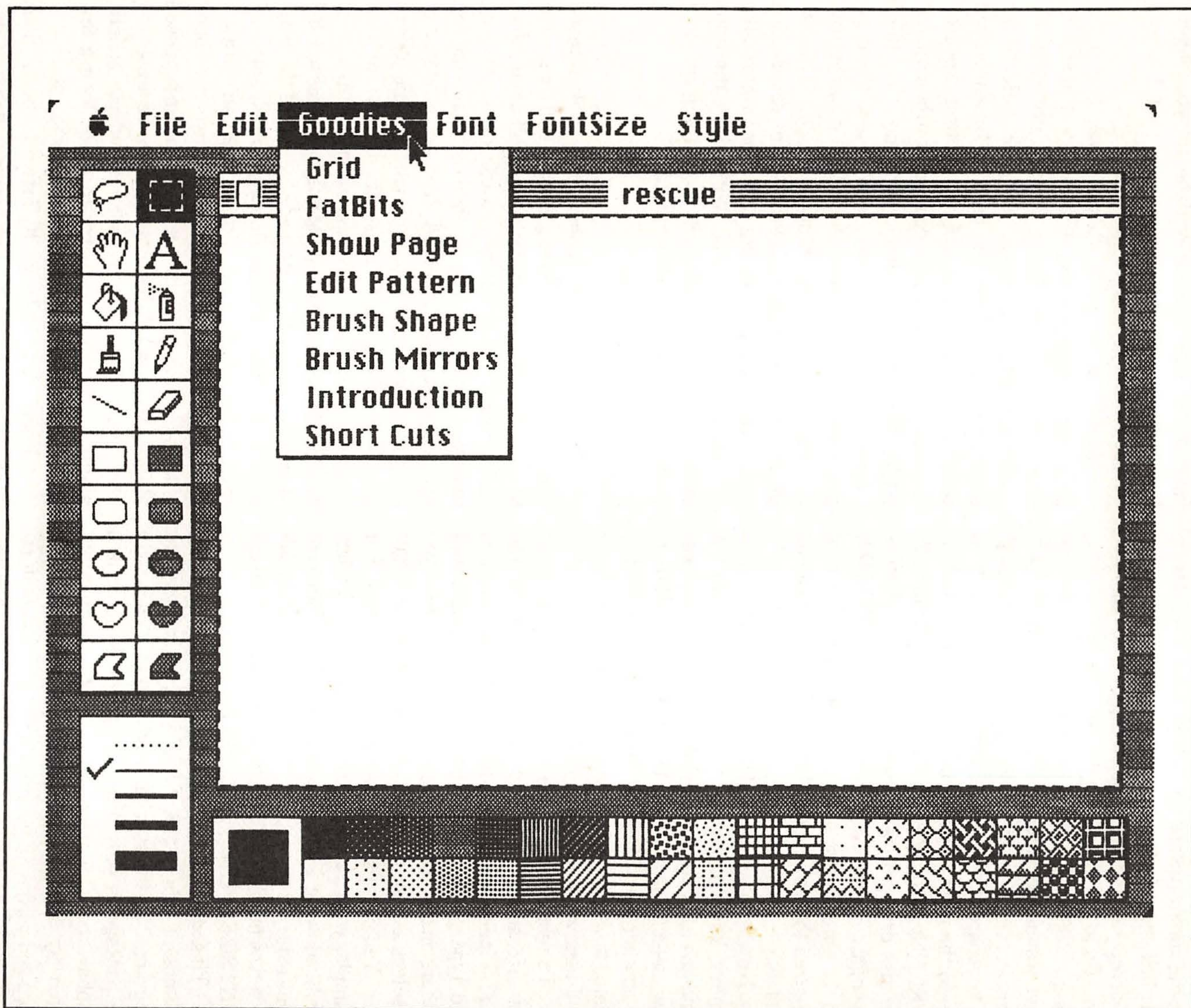


Fig. 2-6. The GOODIES menu, which offers a number of sophisticated graphics functions.

tion the cursor toward the bottom left of the screen and click once. Go to the FONT menu at the top of the screen and select the first style, which is New York. Go to FONTSIZE and select the first choice, 9 point. (Size is measured in points, the way an artist would measure it.) Now, go to the STYLE menu and select Plain. Now type your first name at the keyboard. You will see that it appears in a small size to the right of your cursor. Do not press RETURN, but go to the FONTSIZE menu and click 12. The disk drive whirs and when it stops the letters are slightly larger. Now, select 14 from FONTSIZE and the letters grow still larger. Scan through all of the sizes to get an idea of the wide range of looks available.

Now, go to the FONT menu and select Geneva. You will now see that your letters stay the same size, but they are printed in a different font. Now, select Toronto from the FONT menu. Here is yet another font. Move all the way down the FONT menu, choosing the various options until you have a good idea what each font looks like.

To test the STYLE menu, choose a 36-point font size and a New York font. Then go to the STYLE menu and click Bold. Your display is now in bold type. Now select Italic and an italic script appears on the screen. Further tests will allow you to select an Outline style, Shadow style, or whatever. You can also mix and match the styles. For example, you can display text in Bold Italic Underline Shadow, or Plain Underline Outline. There are thousands of different combinations that can be accessed by mixing and matching the options from the FONTSIZE, FONT, and STYLE menus.

This discussion has only touched upon the millions of things you can do with this excellent program. The instruction booklet that comes with the package is only about 30 pages long. It is a graphics tool you learn to use not by reading, but rather by experimenting. Almost anyone can sit down and within a few minutes feel fairly comfortable with it. It takes only a week or so to become familiar with every option available (maybe longer; I'm still discovering new things that I never guessed were possible).

MacPaint conforms better than any other programs I've seen to the Apple concept of an efficient user interface. Anyone can do professional-looking graphics drawings using this electronic palette. Many novices who have become involved with computers in the recent past, at first expected the computer to be able to do more for them than it could. With *MacPaint,* novices can now find their expectations met. To many neophytes, the computer is something you just kind of sit down in front of, and with a little help, it does its thing. Those of us who know computers a little better realize that this has never been true, but with *MacPaint* and the Macintosh, it almost is true. With a little direction from the user, *MacPaint* will do its thing and do it rapidly, accurately, and beautifully.

Examples

It is not within the scope of this book to teach someone who knows nothing about art all there is to know about drawing on the Macintosh using *MacPaint.* Since theoretically you can draw anything that you could draw with a pencil or brush, one can see the difficulties involved. It is not a matter of teaching someone how to program the computer to do these things, since these are things the user does by hand using the computer and the mouse as palette and paint brush. What will be discussed and shown here are drawings made by two of the least artistic people in the world, my son Robbie and myself. Between the two of us, we can't draw a straight line, but on the Macintosh, we have been able to impress more than a few with our supposed ability. Again, all of these examples were drawn in a short period of time by persons whose artistic abilities are limited. I'll try to pass along a few tricks that made the job a bit easier for me.

Figure 2-7 is not really artwork, but it does provide a graph of all the resident hatch styles

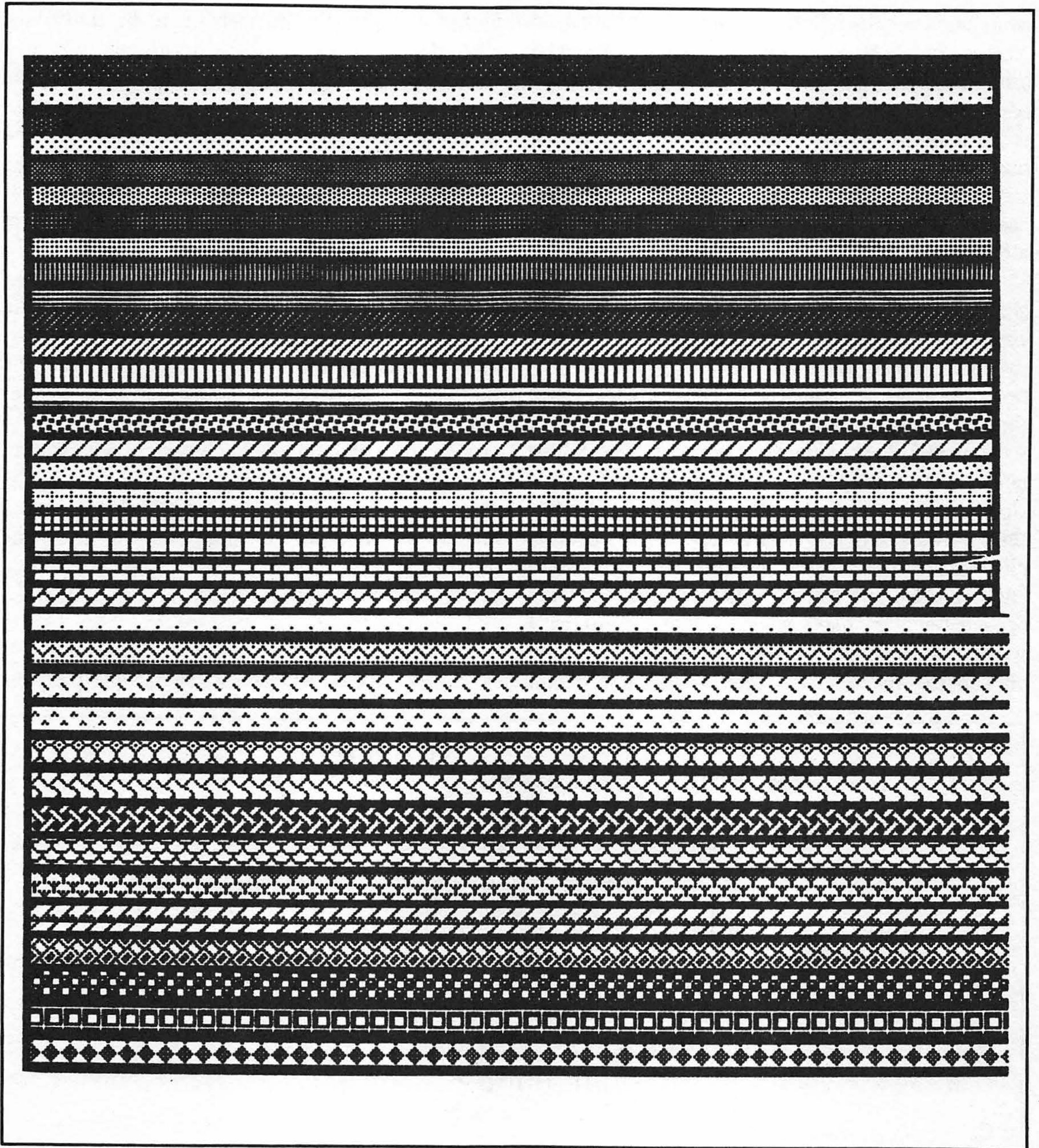


Fig. 2-7. The preprogrammed patterns available in *MacPaint*.

available when *MacPaint* is first initialized. These were produced by using the rectangle icon to make a large square on the screen. Lines were then inserted, and in the spaces, the paint bucket was used to fill in all 38 patterns that are offered, including black and white. Remember, in the GOODIES menu, there is a suboption called Edit Pattern you can use to make your own patterns or alter those already available.

Now on to the artwork. Figure 2-8 shows a pie chart that I made in about two minutes flat. The circle was drawn using the circle icon. I then clicked the rubber band line to outline the various pie sections. All I did was click the mouse at the center of the circle and then move outward toward the rim. As soon as I reached the rim, I released the mouse button and a line appeared from the center to outer rim. Another click breaks the rubber band effect. I then went back to the starting point at the center, clicked and held again, proceeding outward to another segment. I filled in the patterns by using the paint bucket. I simply clicked the pattern that I wanted to use for fill from the pattern menu. I then clicked the paint bucket icon in the left menu, placed it within the pie section to be filled, and clicked again. The only problem with a pie chart like this is that the segments themselves may not be in correct proportion to other segments regarding percentages. For instance, if one segment is to represent 50% of a total, then obviously, the pie section will fill half the circle. No problem here. Nor is it difficult to divide the circle into quarters. However, if one segment is to represent 25% of a total and another is to represent 22%, I doubt the human eye would be able to adequately calculate the exact size for the 22% pie segment. However, most pie charts need not be extremely accurate. So the 22% segment could be drawn slightly smaller than the one representing 25% without any problem arising. Although these pie charts are not as accurate as those that are produced with a complex computer program that takes actual values, in most instances, these are certainly easy to draw, and often the speed with which such a display can be produced is more important than how accurate it is.

PATTERNS, PATTERNS, PATTERNS

It is easy to produce kaleidoscopic patterns

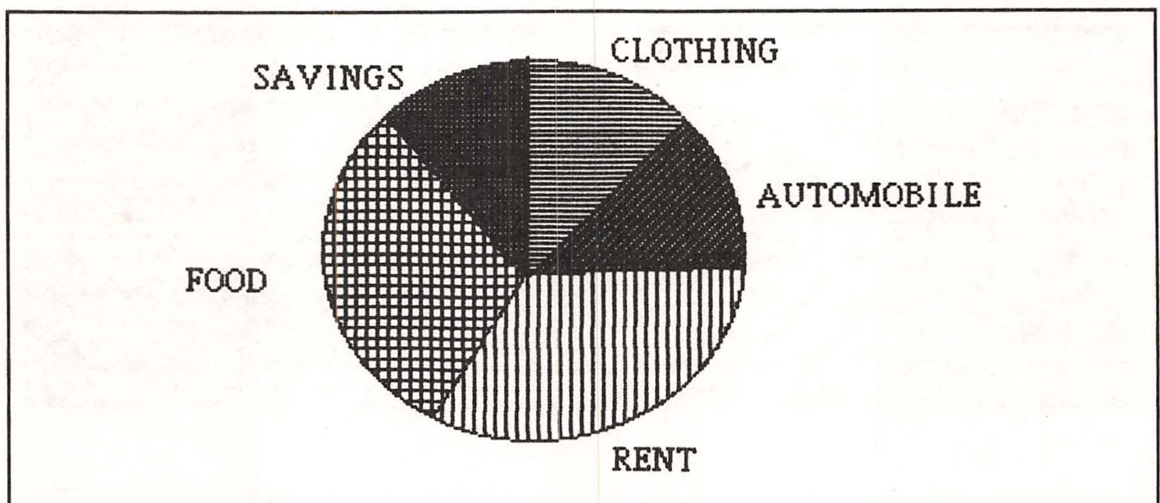


Fig. 2-8. A pie chart, produced in just a few minutes using the *MacPaint* tools.

with a single brushstroke while operating under *MacPaint*. To do this, it is first necessary to access the Brush Mirrors suboption in the GOODIES menu. Figure 2-9 shows a portion of what is displayed on the screen when this option is selected. This is your brush mirror pattern. The lines shown are boldface in this drawing, but will be much thinner on the actual screen. You can set up one of several different mirror patterns by placing the mouse pointer on one or several of the three intersecting lines. Each time you click the mouse, the line you click becomes boldface.

If you click the diagonal line in the upper left corner of this drawing, you set up a two-brush effect. The second brush will draw the same thing at the same time the first brush does but will draw it in a 90-degree angle from the first. This means that if you position your brush at the left center of the screen and draw a line horizontally, another line will also appear vertically and eventually intersect it as you continue traveling to the right. Instead of a single horizontal line, you would end up with two intersecting lines.

Now, access the Brush Mirrors suboption

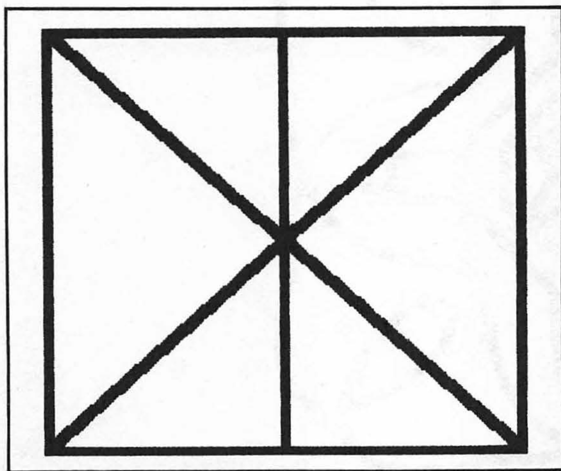


Fig. 2-9. The brush mirror pattern, accessed from the Goodies Menu.

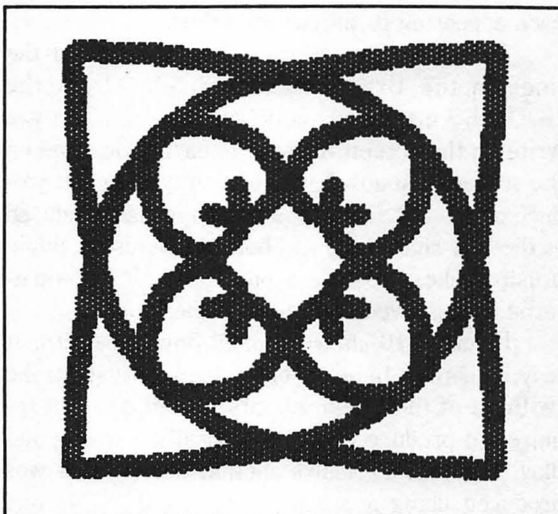


Fig. 2-10. A symmetrical flower-like pattern, produced with a single mouse movement when all brushes are turned on.

again and click off the line you previously clicked on. When you do this, all lines should be in thin format. Then, click the vertical line at the top center of the mirror box. This line will become fat, and you will have set up a mirror brush that is 180 degrees from the first one on the screen. If you now draw a horizontal line across the screen from left to right, the mirror brush draws one from right to left along the mirroring plane.

By turning off the vertical line in the mirror box and clicking on the right diagonal line, your mirror brush is 270 degrees from the first one. When you draw a horizontal line with the first brush (starting on the right) the mirror brush draws one starting from the bottom and moving upward toward the top of the screen. Again, two intersecting lines are produced. You will recall that the first diagonal line clicked caused the mirror brush to begin at the top of the screen and move downward.

By clicking on the horizontal line, you set up a mirror brush that will copy everything you draw on the screen in a mirror image. If you draw a single horizontal line, you will have two horizontal lines,

each appearing to mirror the other.

As a further experiment, click on all of the lines in the Brush Mirrors display from the GOODIES menu and see what happens when you write on the screen. Instead of having one line on the screen, you now have eight. Anything you produce on the left side of the screen is also produced at the top, right side, and bottom. By using single brush strokes, you can produce kaleidoscope patterns in one sweeping move of the mouse.

Figure 2-10 shows a small flower pattern. It may be hard to believe, but only one brush stroke (with all of the brush mirrors turned on) was required to produce this geometrically pleasing display. Figure 2-11 shows another pattern that was produced using a smaller paintbrush. This one required three different strokes, which is evidenced by the fact that the center display is not connected with the one outside it, or the one to the outer perimeter. Figure 2-12 shows a design that was produced by using a medium-sized brush and one of the hatch patterns at the bottom of the screen. The design in Fig. 2-13 was produced with the largest brush available in the Brush Size menu.

And now here's a trick to drawing with *MacPaint*. Draw any design you want on the screen. Now, move your mouse pointer to the border icon at the top right of the graphic tool menu. Click twice. You should see a border appear around the entire screen. Now, hold down the Command key to the left of the space bar and press the E key. Press both keys simultaneously. You will hear the disk drive whir for a second or so and then suddenly, your pattern will begin to change. Figure 2-14 shows the previous image as it appears 20 seconds or so after the Command/C keys have been pressed. As you hold down those keys, the display is duplicated over and over again in a position

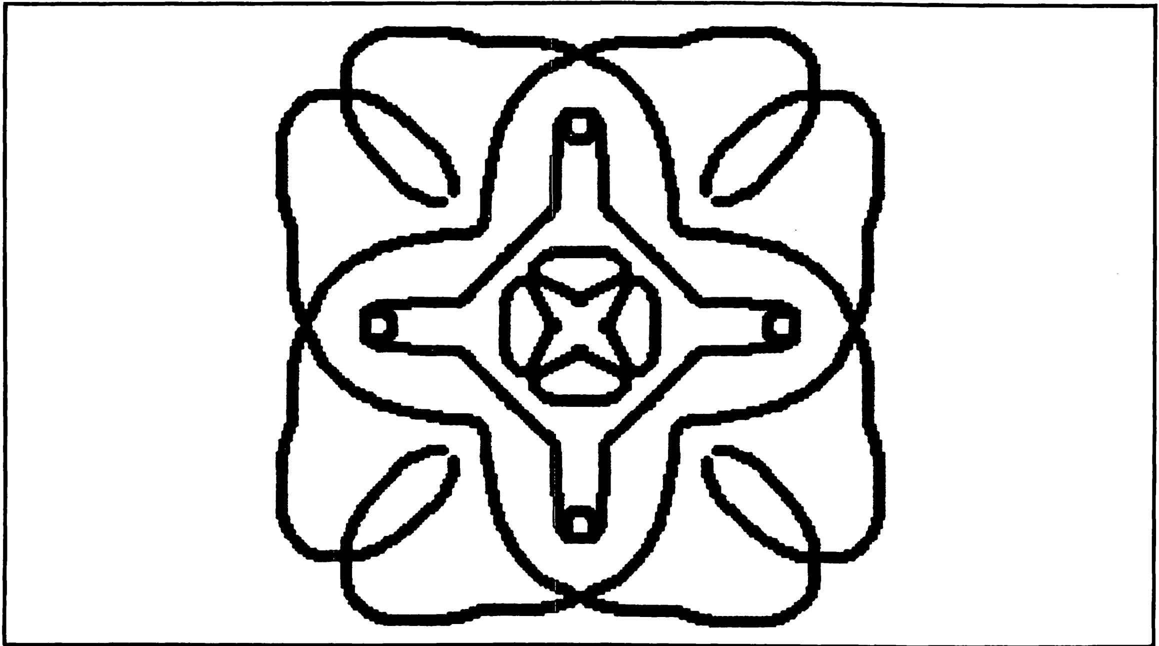


Fig. 2-11. Another artistic pattern, produced with a smaller paintbrush.

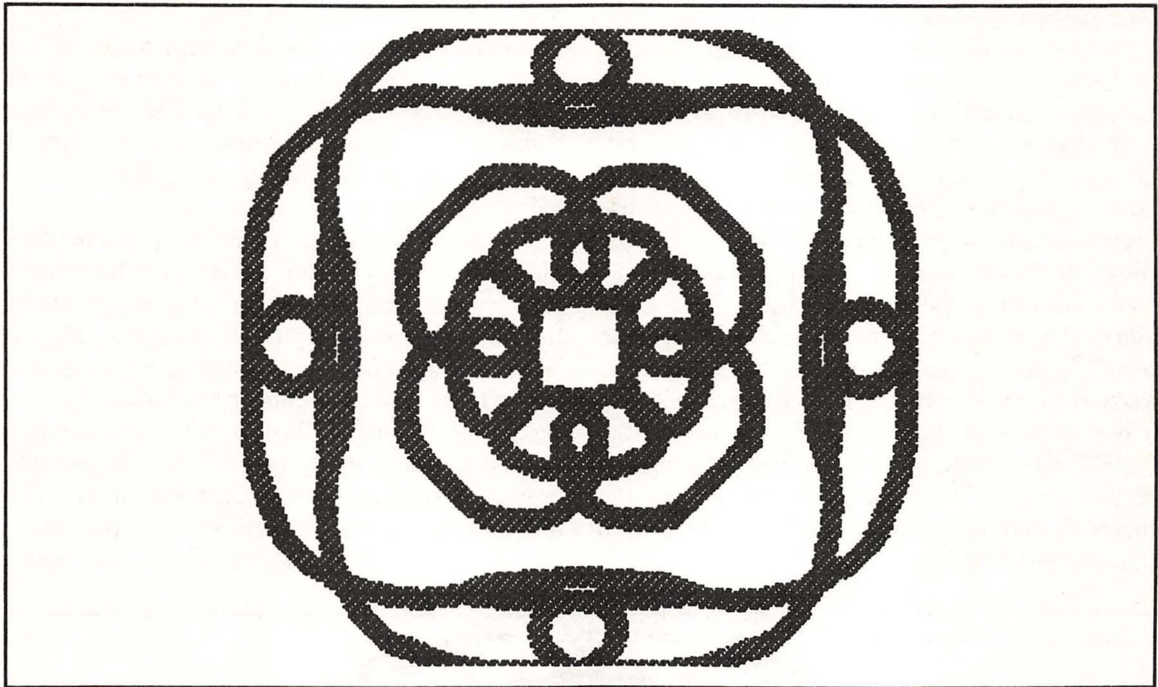


Fig. 2-12. An image, produced using a medium-sized paintbrush and one of the hatch patterns available in *MacPaint*.

slightly off center from the original display's location. Figures 2-15 and 2-16 show other examples of the displays you can make using this gimmick. Figure 2-15 shows the original design, while Fig. 2-16 shows it in modified form. The display will continue to change as long as you hold down the two keys. If you reach a point where you like the design, release the keys and the image will remain stationary. If for some reason you go too far, and pass the design you want, simply pull up the EXIT menu at the top of the screen and select UNDO. This will put the drawing back in its original form.

DRAWING LINES

It is incredibly easy to produce line drawings using the tools available in *MacPaint*. Before starting, here are a few tips. When you're in the pencil mode, it's still difficult to draw a straight line (almost impossible). This mode is used for drawing figures that the other drawing tools cannot produce. With the pencil, you can write your name, for instance. You can use the rectangle tool to draw a straight line every time, however, simply by positioning the crosshair at the point you want your line to begin and moving up, down, left or right. You cannot draw a diagonal line with this tool, but you can draw perfectly straight horizontal and vertical lines. As you move, undoubtedly, you will get slightly out of line and a small box will be formed. But as long as you hold down on the mouse clicker, the box is not permanent. When you get to the end of your line, simply move the mouse in the direction that decreases the size of the box until, finally, you end up with a box whose upper and lower sides align. This is your standard, everyday line.

To draw diagonal lines, use the rubber band

tool to the left of the eraser. You can also draw straight lines with this tool, although it's often easier to use the rectangle.

Another method of drawing vertical or horizontal lines uses the keyboard and the pencil. First, click the pencil icon and move it to the starting position on the screen. Now, release the mouse button and hold down the press SHIFT and E keys. Move the mouse either horizontally or vertically and while in this mode your line will be straight. If you start moving vertically, regardless of where the pencil is on the screen, the line will always be vertical. If you want to switch from a vertical line to a horizontal line, you will have to release the SHIFT and E keys, position the pencil again, and hold down SHIFT and E again. Hold down the mouse button again, and you will now be able to draw a horizontal line.

SOME GRAPHIC RESULTS

Perhaps the first line drawing anyone does on the Macintosh is a picture of the computer itself. My attempt is shown in Fig. 2-17. I'm rather proud of this picture, although there are still a few flaws I would like to correct. Let me tell you the steps I followed to draw it.

I started by using the rectangle icon to produce the face of the computer. I then used the rubber band icon to sketch the connecting lines between the back and front panels of the computer. The top and sides section was also formed with the rectangle icon. I used the rubber band mode again to sketch the indentation beneath the computer face and the lower pedestal portion. The disk slot was produced using a line with a moderately large width followed by one with the widest width possible.

While I still want to improve it, I'm rather

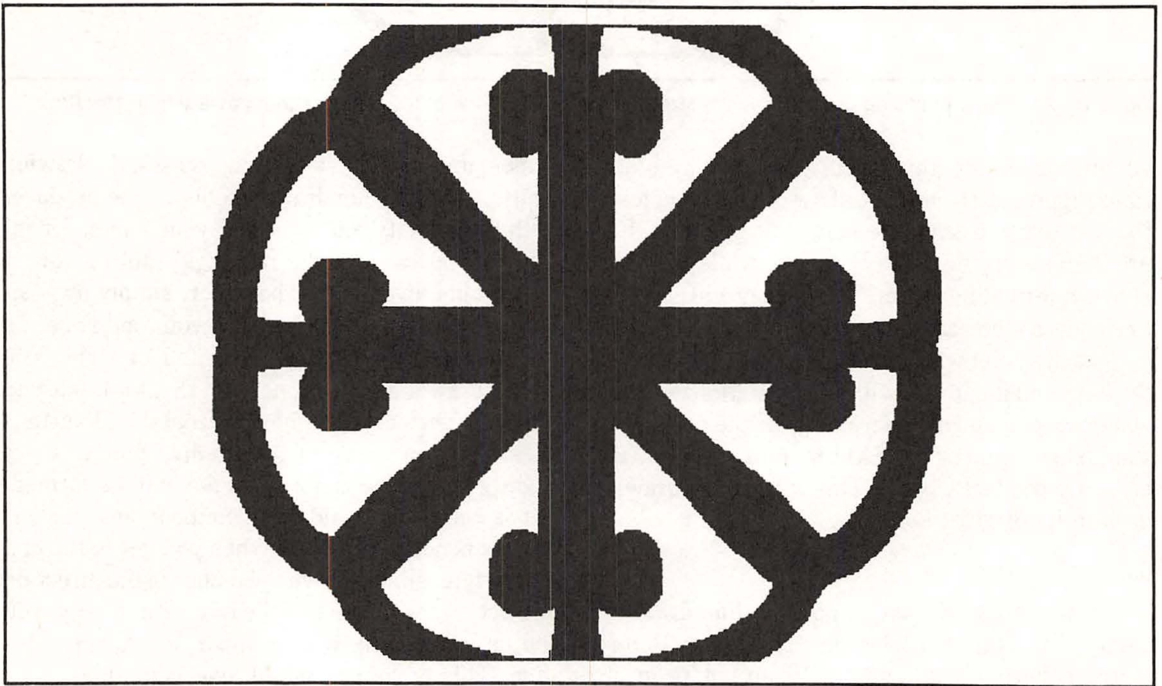


Fig. 2-13. An image produced with a large paintbrush.

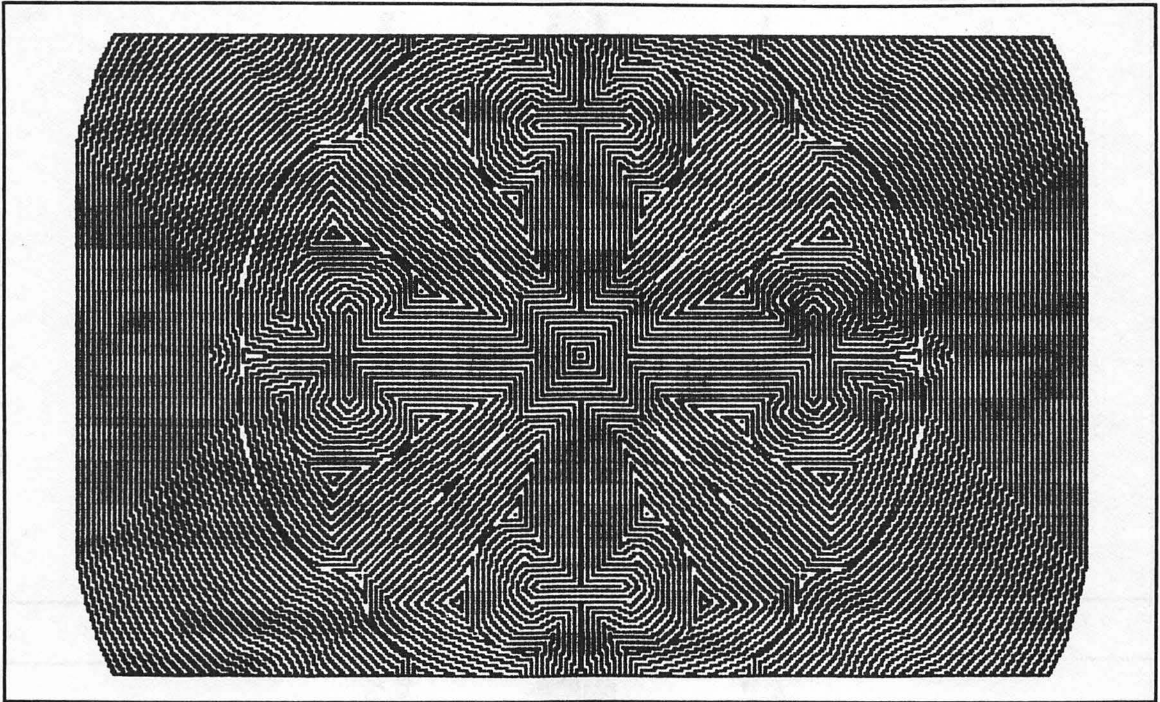


Fig. 2-14. A modified image that uses the Command/C combination from the keyboard.

pleased with the Apple logo in the small square near the bottom left of the computer face. I did this using the Fat Bits suboption in the GOODIES menu. First, I used the rectangle icon to draw the small rectangle the Apple is inside. I then entered the Fat Bits mode, blew this rectangle up to full screen size, and using the pencil icon, I laboriously drew the Apple symbol, building it from the "fat" blocks. I left Fat Bits mode, and the Apple logo appeared in smaller form. I used the same method to draw the keyboard outlet on the computer pedestal.

The screen of the computer is the rounded rectangle filled in. To make the entire drawing more appealing, I filled in the background with the steel fence pattern. All in all, I had to pat myself on the back and say "Not a bad drawing for an amateur artist."

The next two drawings were produced by my son on his second day with the Macintosh. Figure 2-18 shows a portable noise machine, otherwise known as an AM/FM stereo cassette unit. The steel fence effect is used to fill the two circles that represent the speaker enclosures. The rest of the drawing primarily consists of rectangles and straight lines. The designators below the cassette window were done using the Fat Bits mode and by "blocking in" the letters with the pencil. I feel the drawing is pretty good.

Figure 2-19 is a drawing of an old portable television set that resides in my son's bedroom. If you're an engineer or architect, you may notice a few miscalculations of proportions—for instance, the handle is not perfectly centered, but all in all, I would say it's a pretty fair drawing. The RCA trademark at the bottom of the screen is just as it

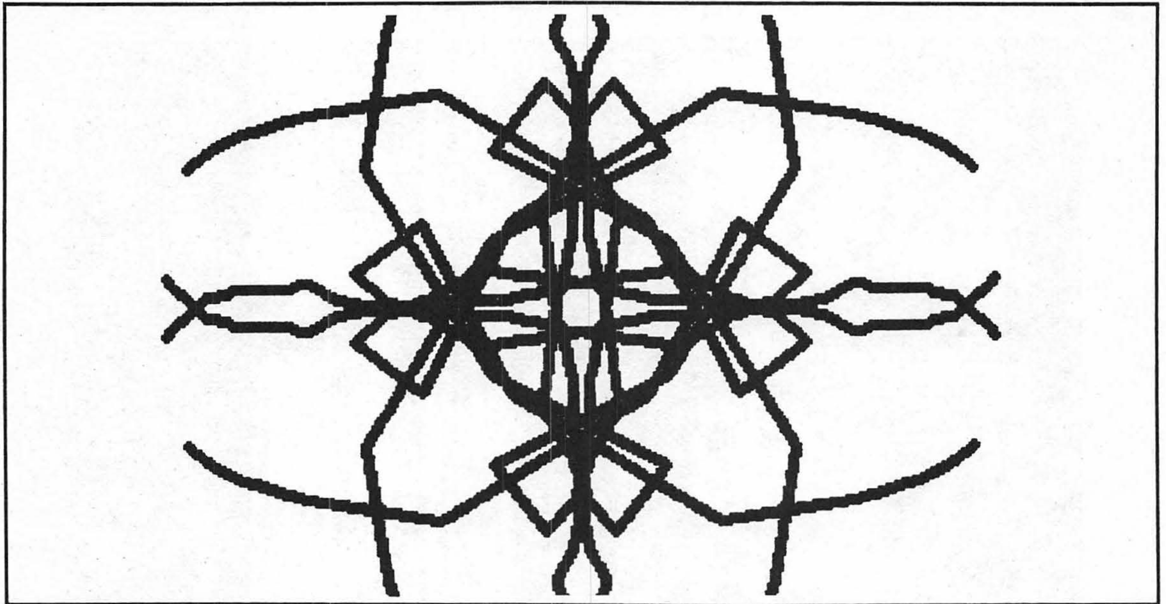


Fig. 2-15. An original image produced by a medium-sized paintbrush.

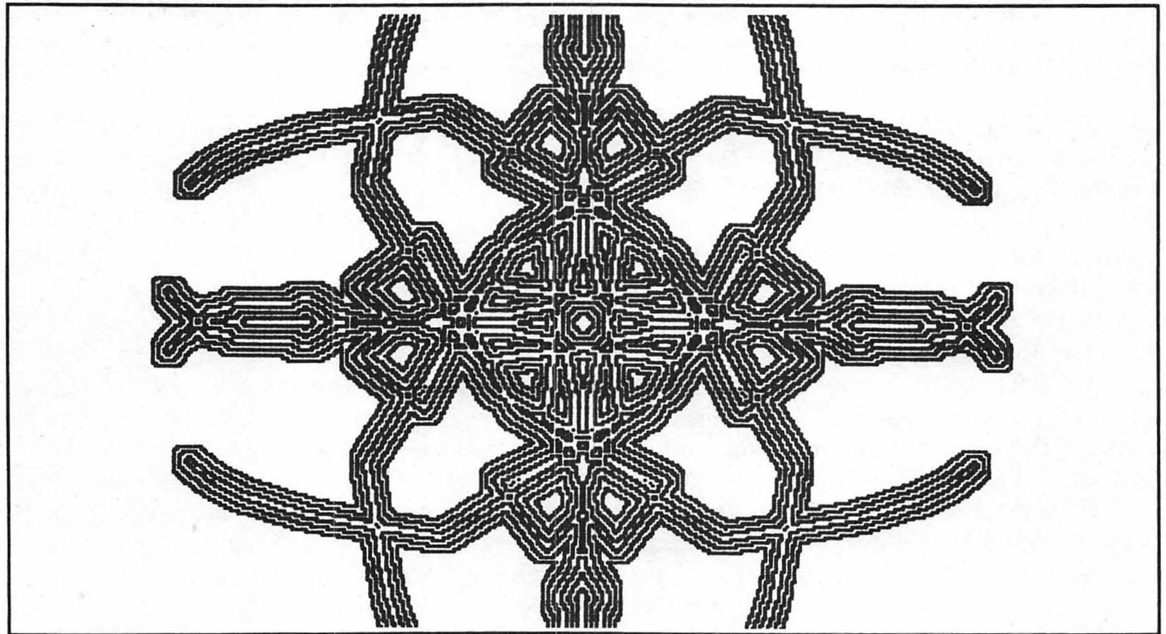


Fig. 2-16. The image from Fig. 2-15 as modified by the Command/C keyboard option.

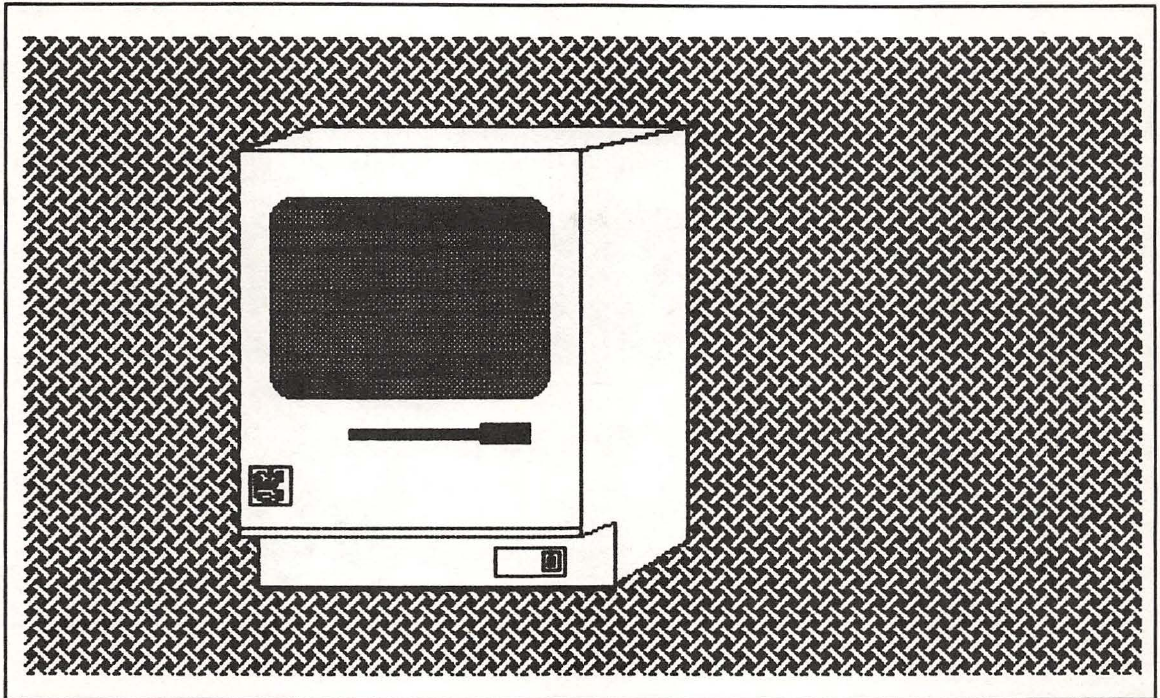


Fig. 2-17. The author's rendition of the Apple Macintosh.

appears on the actual unit. He used Fat Bits to make the trademark and the numbering on the tuner knob.

These drawings could go on forever, and they will as long as Macintosh and *MacPaint* exist. The appendices of this book contain other examples of drawings done with *MacPaint*.

SHRINKING AND ENLARGING

Any image you can draw to the screen can be enlarged or shrunk. Two methods of doing so are explained below.

To use the first method, draw any small object at the center of the screen. When it is complete, click the border icon and move the pointer to a point just above and to the left of your drawing. Press the mouse button and encompass the image in the border box. Now, access the EDIT menu and select Cut. When Cut is activated, the disk drive will whir and the image will disappear from the screen. Now go back to the screen (you're still in border icon mode) and border in an area about twice as large as your original image. There won't be anything on the screen at this time except the border box itself. Release the mouse button, access the EDIT menu, and select Paste. Your image will reappear in large size. Click the border icon one more time to turn off the border feature and then click the eraser twice. This will remove the image from the screen. Click the border box again and border in an area about half the size of the original image. Access the EDIT menu and select Paste again, and your drawing will appear again, but this time in reduced form. The size of the border determines the size of the image. Using this method, any image may be enlarged or shrunk as many times as you want to "paste" it. The only drawback is that some hatch patterns may not

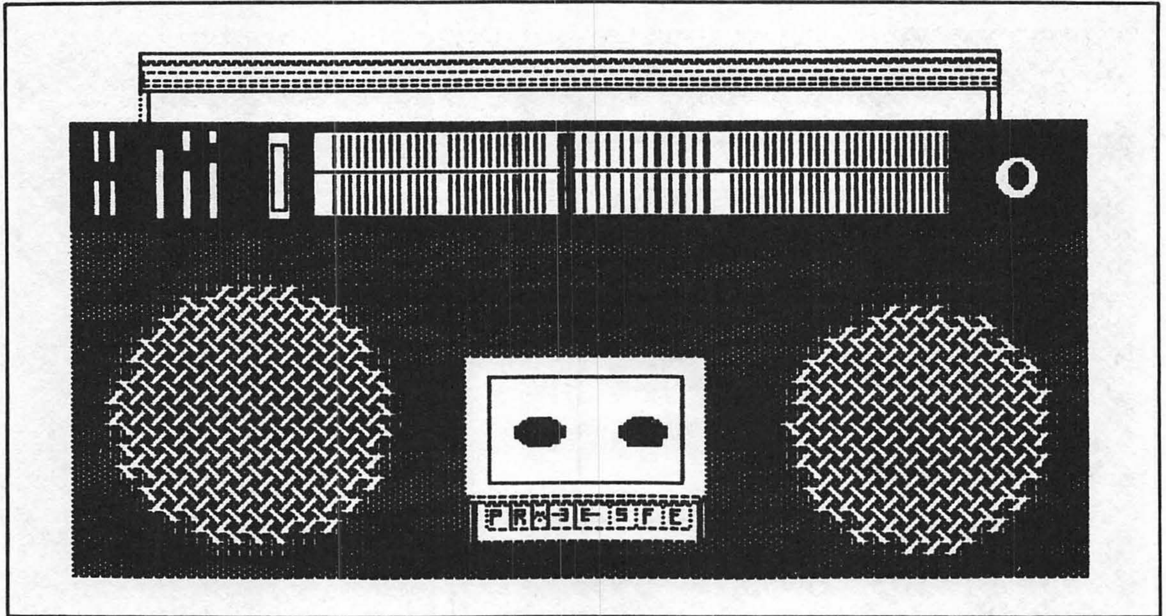


Fig. 2-18. A drawing of a portable stereo, produced by a teenager on his second day with *MacPaint*.

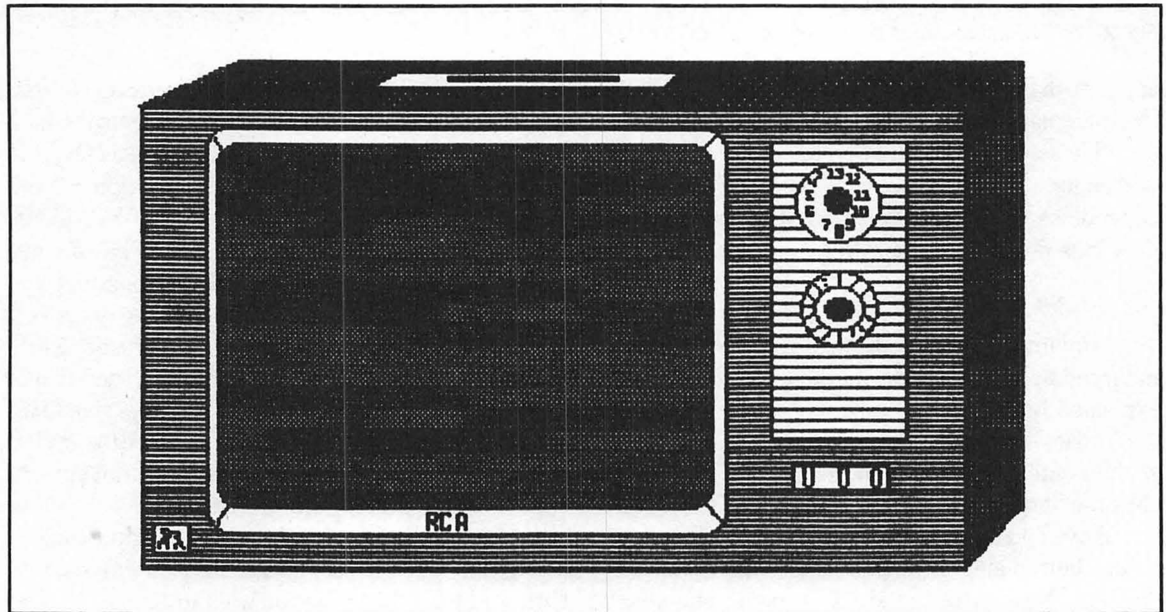


Fig. 2-19. A realistic drawing of a portable television set.

be reproduced accurately when reduced to a very small size.

You can also change the shape or proportions of your image. To do this, erase the screen and click the border icon one more time. This time bring a border on the screen and shape it so it is tall and narrow. Now, click Paste again and your image will be vertically elongated. You might also wish to try a box that is wide and narrow. This will distort your image in a horizontally elongated format. You can also shrink or enlarge parts of an image on the screen simply by bordering in the particular part you want to change.

The second method of enlarging and shrinking is more efficient. I discovered it by accident, and have not yet seen it described in any manuals. Erase your screen and start over again. Draw another small object at the center of the screen. Now, click the border box and border in the image. When you release the mouse button, an arrow should appear as the cursor rather than the cross-hair cursor. If not, move the cursor around a bit (just inside the border) until the standard mouse arrow appears. Now, click the mouse and hold. Move the mouse about the screen. You will see that the image moves with you. This is no big deal and is a standard function of *MacPaint*. Recenter the box and then release the mouse button. Press the SHIFT and COMMAND keys simultaneously. Press the mouse button again. Move the mouse up and down. You will see that the size and shape of the image changes. If your mouse arrow is positioned inside the image, it will move. If your mouse arrow is positioned between the border and the image, pressing the SHIFT and COMMAND keys will allow you to change its overall size. I have found the lower right corner is the best place for enlarging. This way, you can move the mouse down to enlarge the image or, up to a point, to reduce it. Past the reduction limit, the image will enlarge again. This method allows for much more flexibility in expanding or decreasing the size of your image.

USING A FEW MORE TOOLS

Now let's practice cutting out a portion of an image. Start by using the rectangle, draw a medium-sized box at the center of the screen. Now, click the border icon and border in the top half of the box. Place the arrow inside the border and click. By holding down the mouse button and moving the mouse itself, you can move the top half of the box anywhere you want. If you're nimble-fingered, you can even compress the box so that it looks like a standard rectangle. This same method can be used to make circles and ellipses.

You can do the same thing using the lariat icon. Here, all you do is click the lariat and then circle in the area you want to move. Release the mouse button and position the lariat along the perimeter. At the right position, you will see a flashing mouse pointer. When this occurs, click and hold. By moving the mouse, you can now move the part of the image you cut out with the lariat circle to any location on the screen. The lariat is flexible, because it can move any portion of a drawing that you can outline, while the border icon can only move segments that fit within the box-like border.

If you simply wish to copy a portion of an image rather than remove it from the drawing, border in the portion you wish to copy and click Copy in the EDIT menu. This portion of the image is now committed to the computer's memory. To place it elsewhere in shrunken, normal, or enlarged form, use the border icon again. Then click Paste in the EDIT menu. The image will appear in the border.

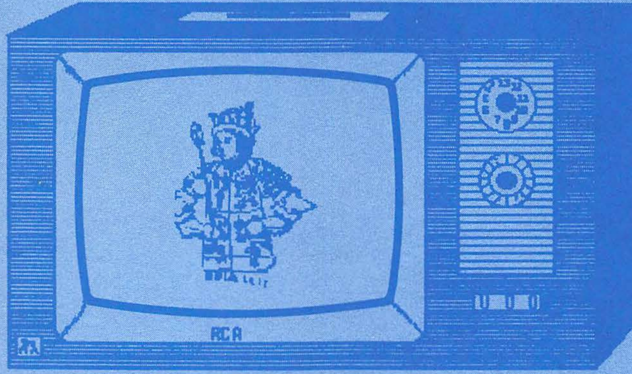
Undoubtedly, there are hundreds of other tips and tricks that apply to *MacPaint*. I would guestimate that even the author is not fully aware of all its capabilities. Constant practice, which can be quite fun, will make the Macintosh computer with *MacPaint* more valuable to you than the most expensive drafting set you have ever owned.

SUMMARY

To say that *MacPaint* is impressive would be a gross understatement. But it is really the Macintosh that is impressive because most of these drawing functions are in the Macintosh ROMs and *MacPaint* merely accesses them.

Anyone can draw with *MacPaint,* from young children to unskilled adults. It is one of the few computer programs on the market that can be useful to nearly everyone. *MacPaint* has helped ensure the Macintosh will be around for a long time.

Chapter 3



Using *MacWrite*

MacWrite is a word processing program that uses both the keyboard and the mouse—and uses them with approximately equal frequency. According to Apple, *MacWrite* introduces radical changes in word processing. You create documents that will print out exactly the way they appear on the screen. Rather than having commands embedded in the text, you actually see what you've really got. You use the mouse to select text and delete, copy, or move it. You can make your documents interesting with many fonts and styles, and you can control margins or line spacing with a single click. You can add *MacPaint* drawings to documents created with *MacWrite* or vice versa. With Macintosh, words and pictures go together naturally.

MacWrite is an interesting word processor. To rate it fairly, I would have to say it is one of the easiest word processors to use, but this ease comes at a cost, because the package is not suited to the production of large documents involving complex operations. They can be written with this package, but complexities may develop.

Just as with *MacPaint, MacWrite's* user/computer interface is efficient. To bring up *MacWrite,* simply insert the disk in the drive and turn the computer on. From the systems menu, click the *MacWrite* icon twice and you will see a screen display like the one shown in Fig. 3-1. As is always the case, you will see the menu bar at the top of the screen that is like the one used in *MacPaint.* The menu bar contains the titles of menus from which you choose various command options. To access these menus, you simply click the desired selection and move the cursor to the suboption of your choice.

The *MacWrite* document window is where all text is written. It contains a title bar, a close box in the upper left corner, a size box in the lower right corner, and a scroll bar in the right margin. You will also see a ruler similar to that found on most typewriters. This lets you know where on the screen your tabs and margins are set. A blinking vertical

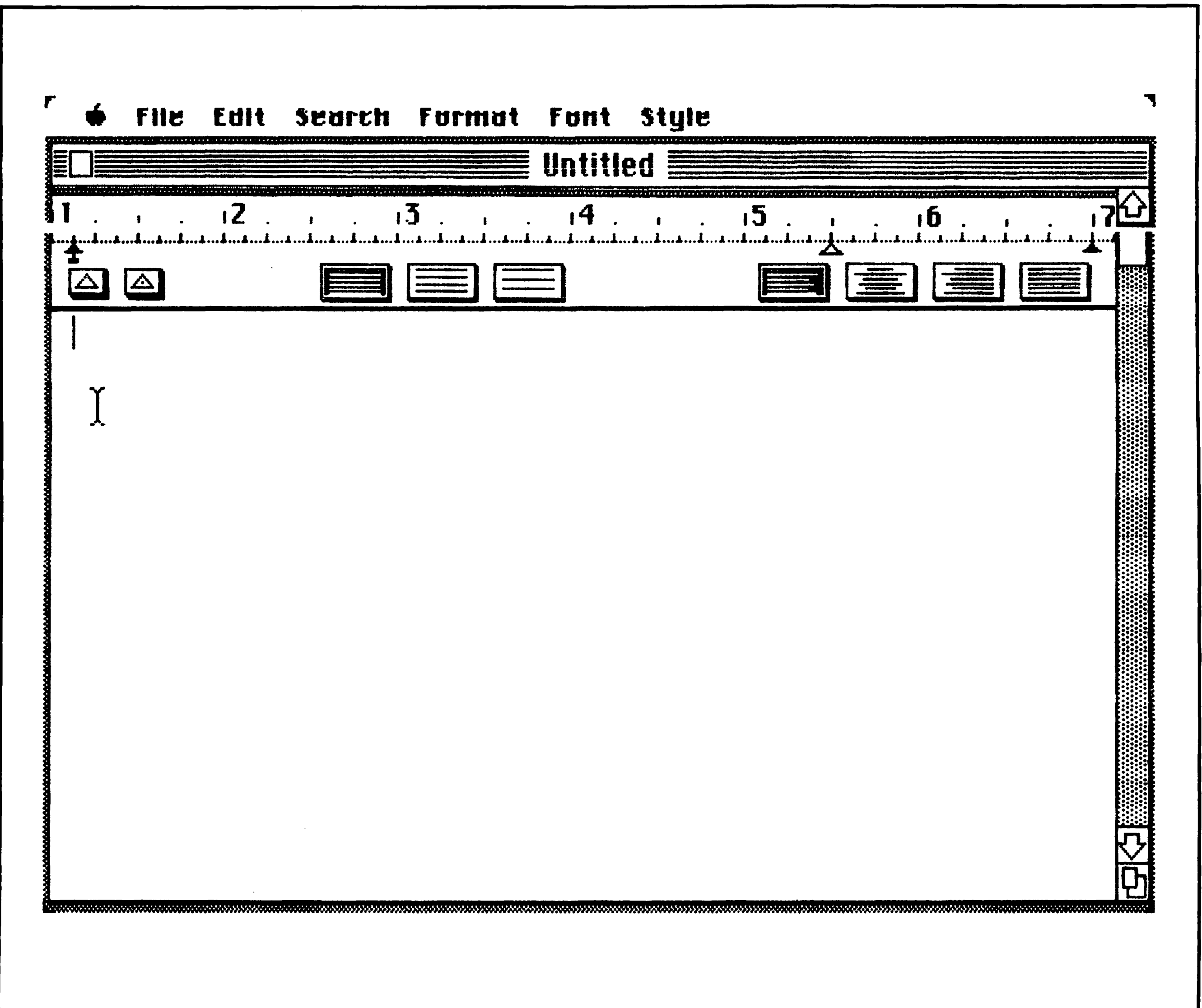


Fig. 3-1. The *MacWrite* window. Note that the menu bar is at the top of the screen.

bar (cursor) marks the location where your characters will be displayed when typed on the keyboard. This bar marks the exact location and will be replaced with a character when typed. The ruler may be removed from the screen or reinserted at other locations. On this ruler, there are adjustable icons for margins, tabs, line spacing, and text alignment.

Apple provides an excellent instruction manual that takes you through *MacWrite* step-by-step, so this discussion will simply be an overview of the package.

The best way to begin learning *MacWrite* is to simply do a little typing. Type the following line:

My name is (your name).

Notice that each time you press a key, it appears in place of the flashing cursor on the screen. If you make a mistake, simply use the backspace key. This will wipe out the letters in reverse order, one letter at a time. You can hold down the backspace key to erase more than one character. When you have entered your line, press the RETURN key and you will see that the flashing cursor jumps down to the left side of the next row (Fig. 3-2).

Unlike a typewriter, *MacWrite* offers a "wraparound" screen. This means that when the text you are typing reaches the right margin, you don't have to press RETURN to get to the next row. The word is automatically sent to the next line. Use the RETURN key only to end a paragraph. To test the wraparound feature, hold down any key that represents a letter or number and watch what happens. The same letter is typed over and over as long as the key is held down. Each time the right margin is reached, the computer automatically executes the equivalent of a carriage return.

Let's leave this document now to clear the screen for another. This one will illustrate other feaures of *MacWrite*. For now, do this by clicking the Exit box in the title margin near the top left of the screen. When you do, the Macintosh will beep twice, and you will be asked whether or not you wish to save the document. For now, click No. The disk will whir and the document window will become blank. Now, click and move the File selection in the Menu Bar. Click New to start a new document, and the computer will provide a cleared document window. Type the paragraph shown in Fig. 3-3. Again, do not press RETURN until you reach the end of the paragraph.

SPACING AND ALIGNMENT

Now, place the arrow on the 1½ space box, located on the ruler just to the right beneath the number 3. This one is the second box in the first three-box series. Click this box. You will now see the line spacing change to the equivalent of 1½ spaces between rows (Fig. 3-4). Now move to the double-space box and click your lines to double-spaced format (Fig. 3-5). If you want to return to the original format, click the first box, and your document will once again be single-spaced.

Now, move on to the alignment boxes at the right side of the ruler. The lines in these boxes indicate what is done to the lines on the screen. For instance, the first box brings about left margin alignment. This means that the letters on the left margin will be aligned, as they already are in Fig. 3-5.

The next box is for center alignment. Click this once. Now you will see that your text centered (Fig. 3-6). Click the first box in this group, and your text will revert back to left margin alignment. Proceed to the third box, click, and the right margin is aligned (Fig. 3-7). When you click the fourth box, both the left and right margins are evenly aligned (Fig. 3-8) just as the text in this book is. In word processing circles, aligned margins are referred to as *justified*.

The spacing boxes and the alignment boxes work independently of each other, so you can align your text using the right group of boxes and then set up the line spacing using the left group.

Right now, let's put the text back in its original

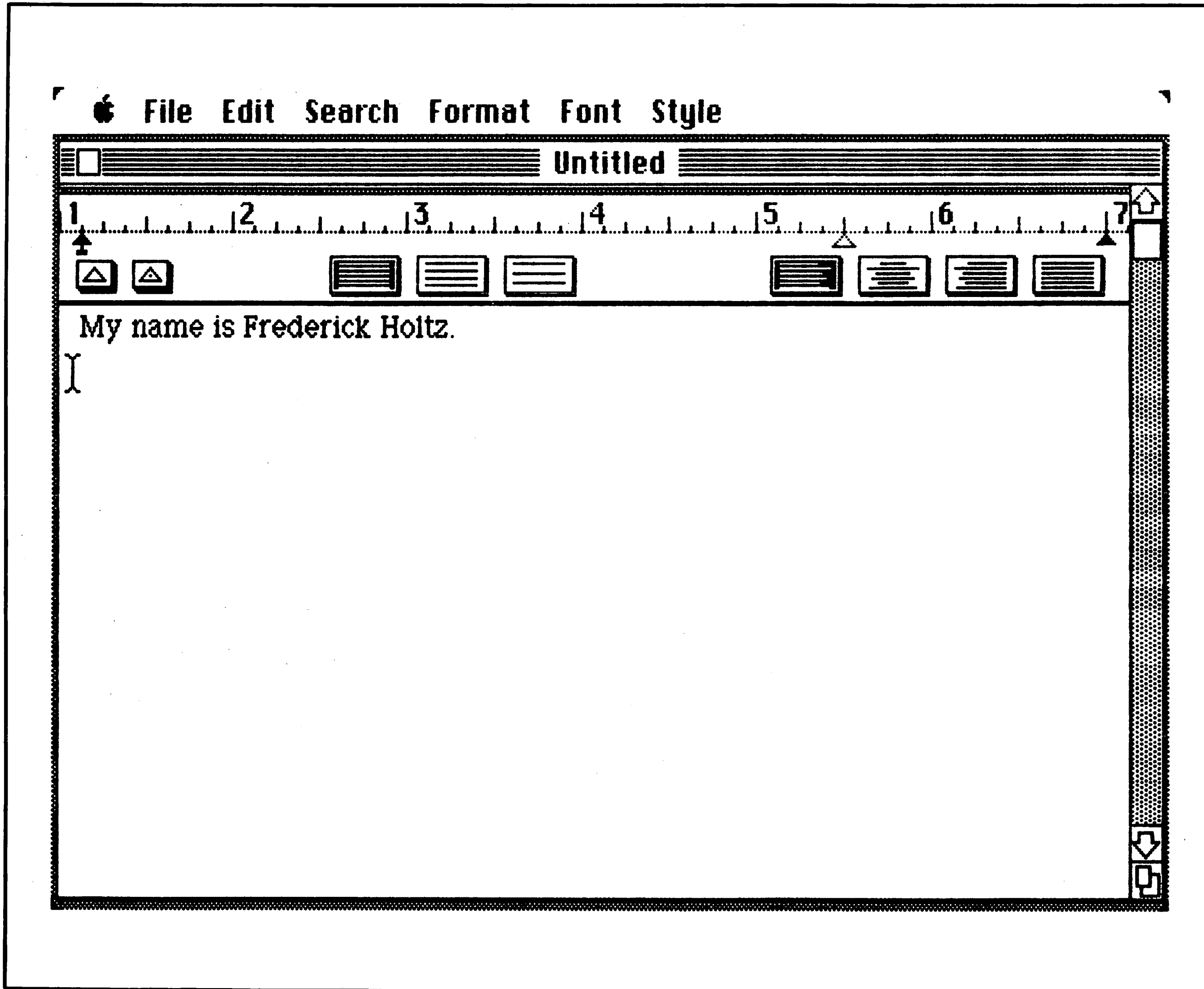


Fig. 3-2. Text, displayed by typing it on the keyboard. The cursor to the lower left of the text can be mouse-controlled to set type at various locations.

This is a test for the Macintosh word processing program called MacWrite. This program is extremely easy to use and makes word processing a joy rather than a chore. Remember, do not press the return key, as MacWrite will automatically "wrap" the screen when the right hand margin is released. Type this example exactly as you see it here. If you make a mistake, simply backspace to the error and correct it. Now, let's purposely misspell a word such as typwriter. The letter "e" has been left out of this word to enable us to test the correcting features found in MacWrite. You may now press the ‹Return› key.

Fig. 3-3. Sample paragraph for testing *MacWrite*. Type this paragraph exactly as it appears. Do not press RETURN until the last sentence is completed.

form by clicking the first alignment box and then the first spacing box.

THE SEARCH SELECTION

You will remember that the word "typwriter" was purposely placed in our document. This is obviously a misspelling of the word "typewriter." Let's correct this word. Move your pointer to the Search selection in the menu bar at the top of the screen. This selection contains two options, Find . . . and Change . . . (Fig. 3-9). Drag the pointer to Change . . . and release the button. A Change window will appear on the screen with two blocks to fill in (Fig. 3-10). The top row is for typing in the misspelled word exactly as it appears in the text, so type the word "typwriter" there and press RETURN. Now, move your pointer to the beginning of the Change To row and click once. Type "typewriter" and press RETURN. Move the mouse pointer to the lower box labeled Change. Click the mouse. You will see that the word is now spelled correctly in the text of the document. We can now exit the Change window by placing the mouse pointer in the exit window at the left of the title bar. The word has been corrected (Fig. 3-11). To remove the black box, click any blank spot on the screen.

For this next maneuver, let's assume that you remember typing the word "enable" somewhere in your document and that after it was typed, you decided to change it to "allow". Assume also that your document is tremendously long and you don't have the time to laboriously search through the lines in order to find it. *MacWrite* allows you to do this very easily. Again, pull out the search menu, this time selecting Find . . .. The Find window will appear on the screen, and you are prompted by Find What to type in the word you're looking for (Fig. 3-12). Type the word "enable." Press RETURN and the word is found in the document and is displayed in white against black block (Fig. 3-13). Type that is white on a black background is referred to as reverse type.

Close out the find window by clicking its exit box, access Search again, and click Change. Here, you will see the Change window with the word "enable" already typed in its row. If you have followed all of these instructions, you will also see the word "typewriter" in the Change To row. Using your mouse pointer, click the area following the "r"

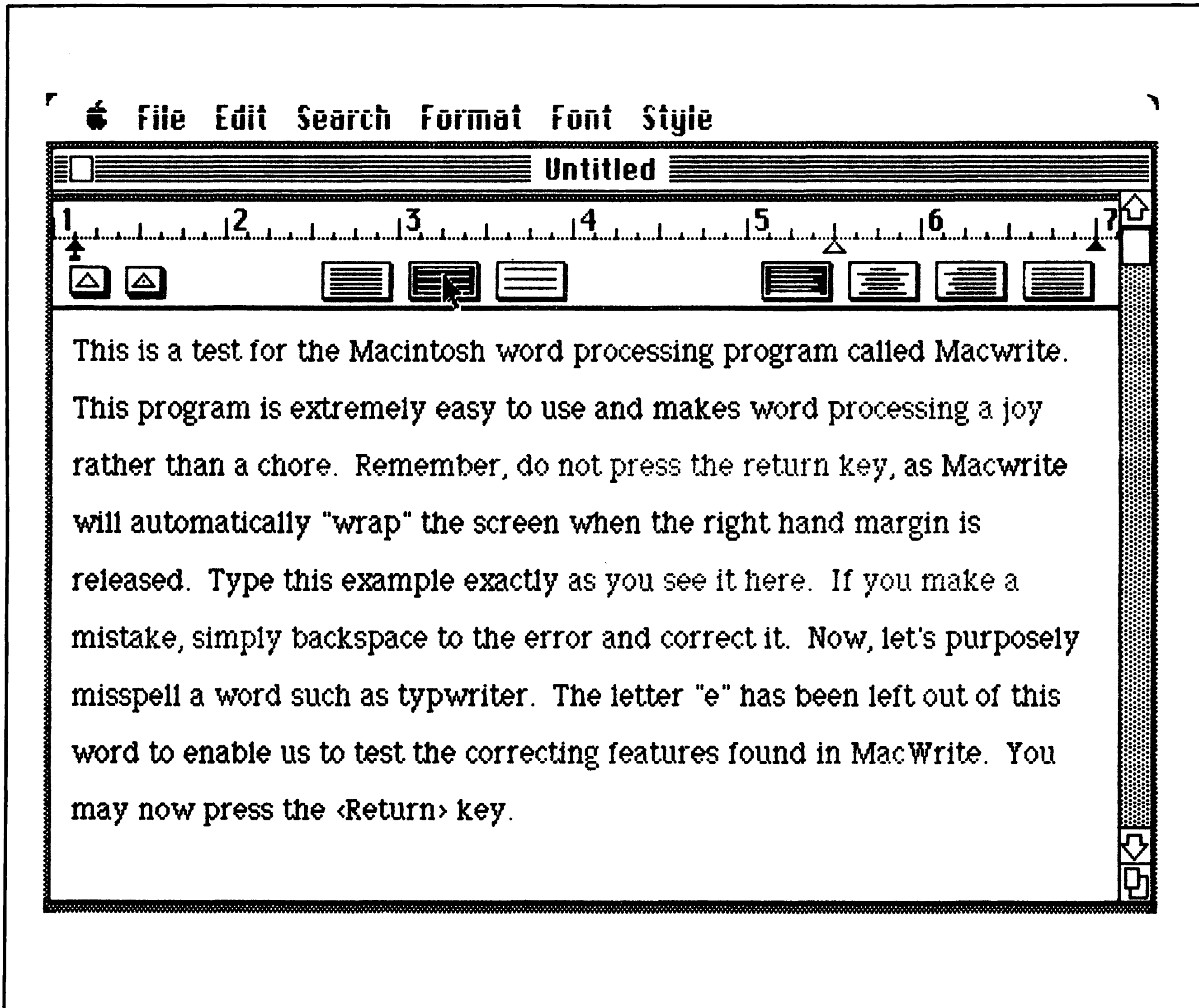


Fig. 3-4. The sample paragraph, displayed with 1½ line spacing.

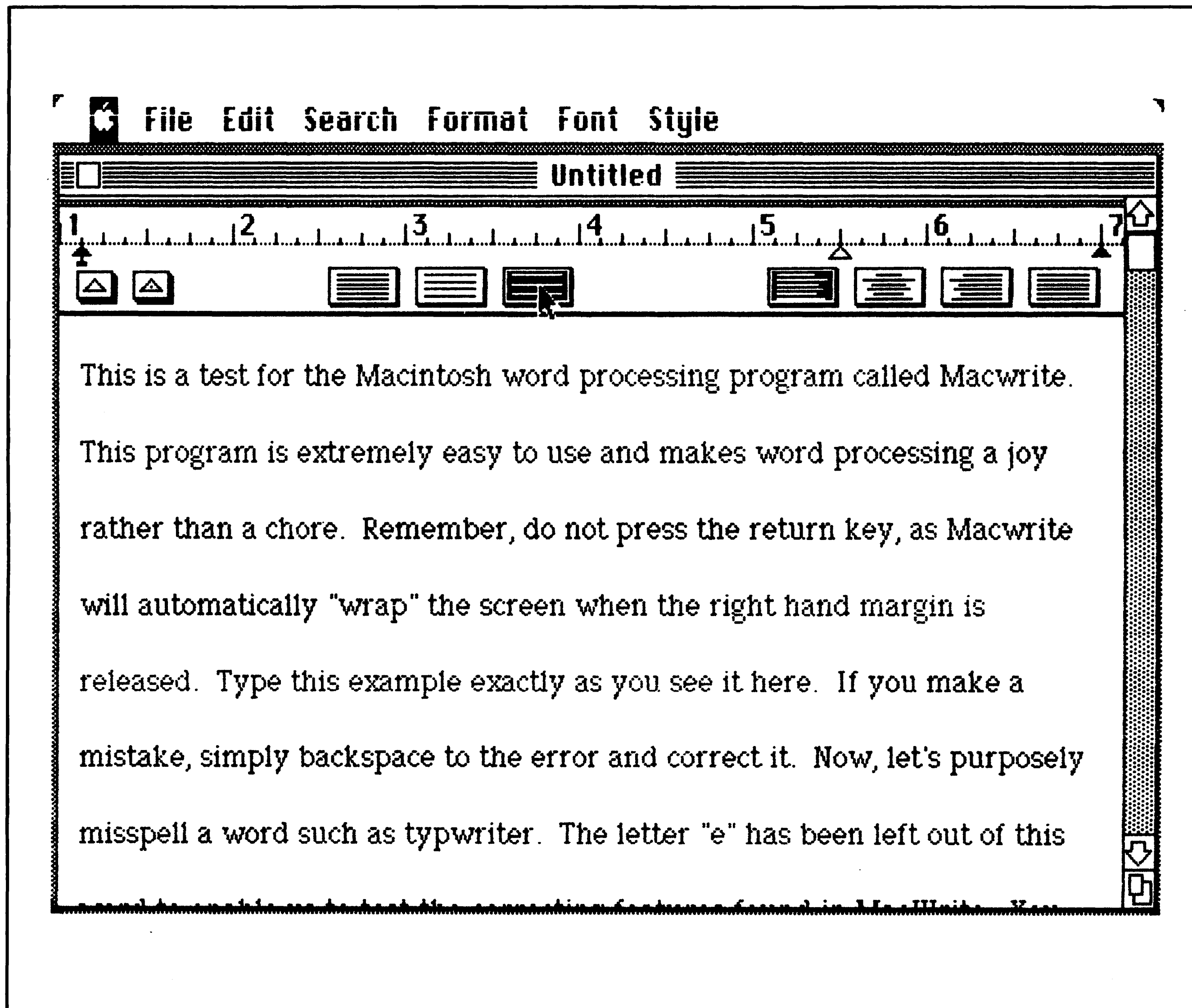


Fig. 3-5. The same sample paragraph, double-spaced.

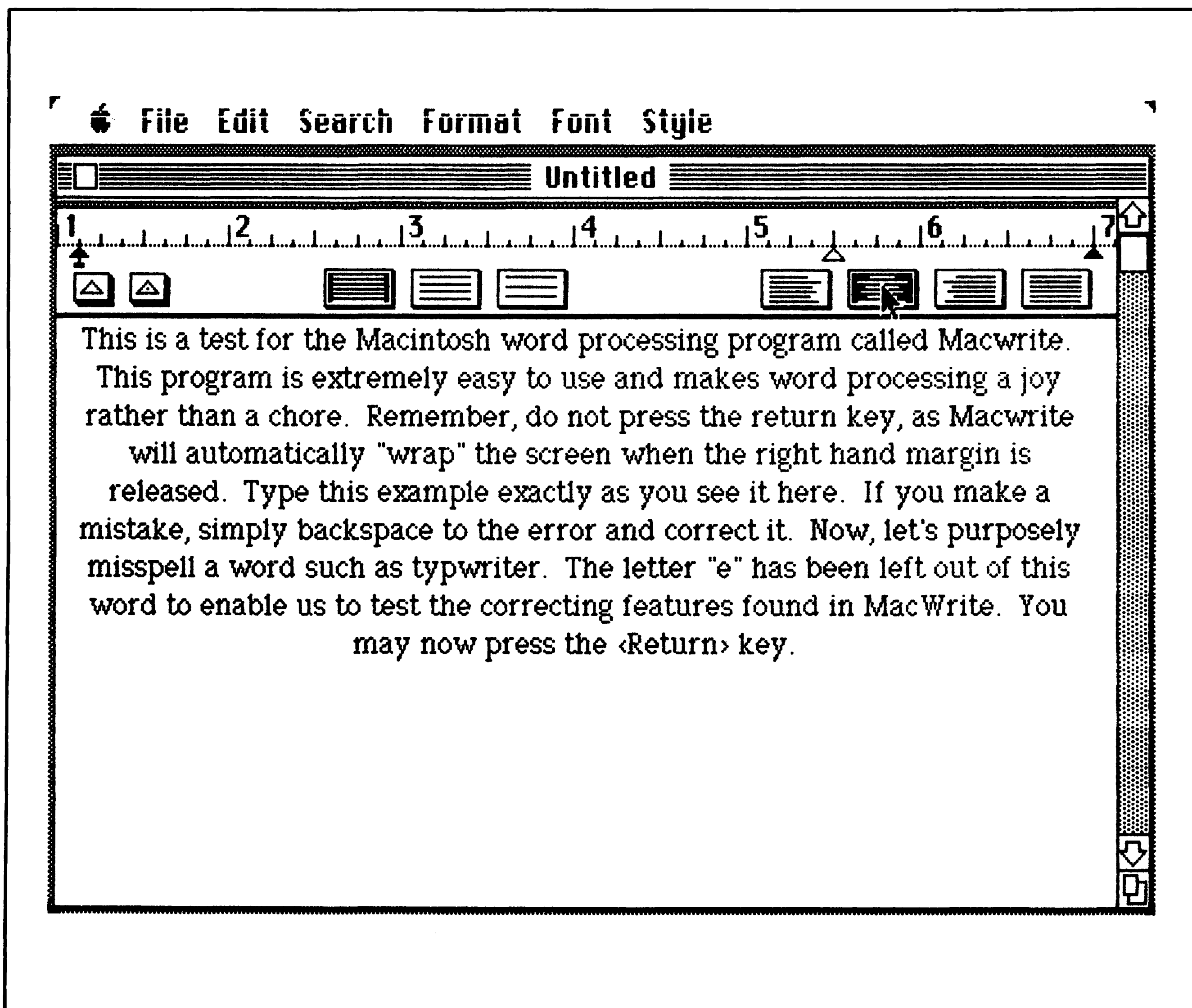


Fig. 3-6. The sample paragraph, set with center alignment.

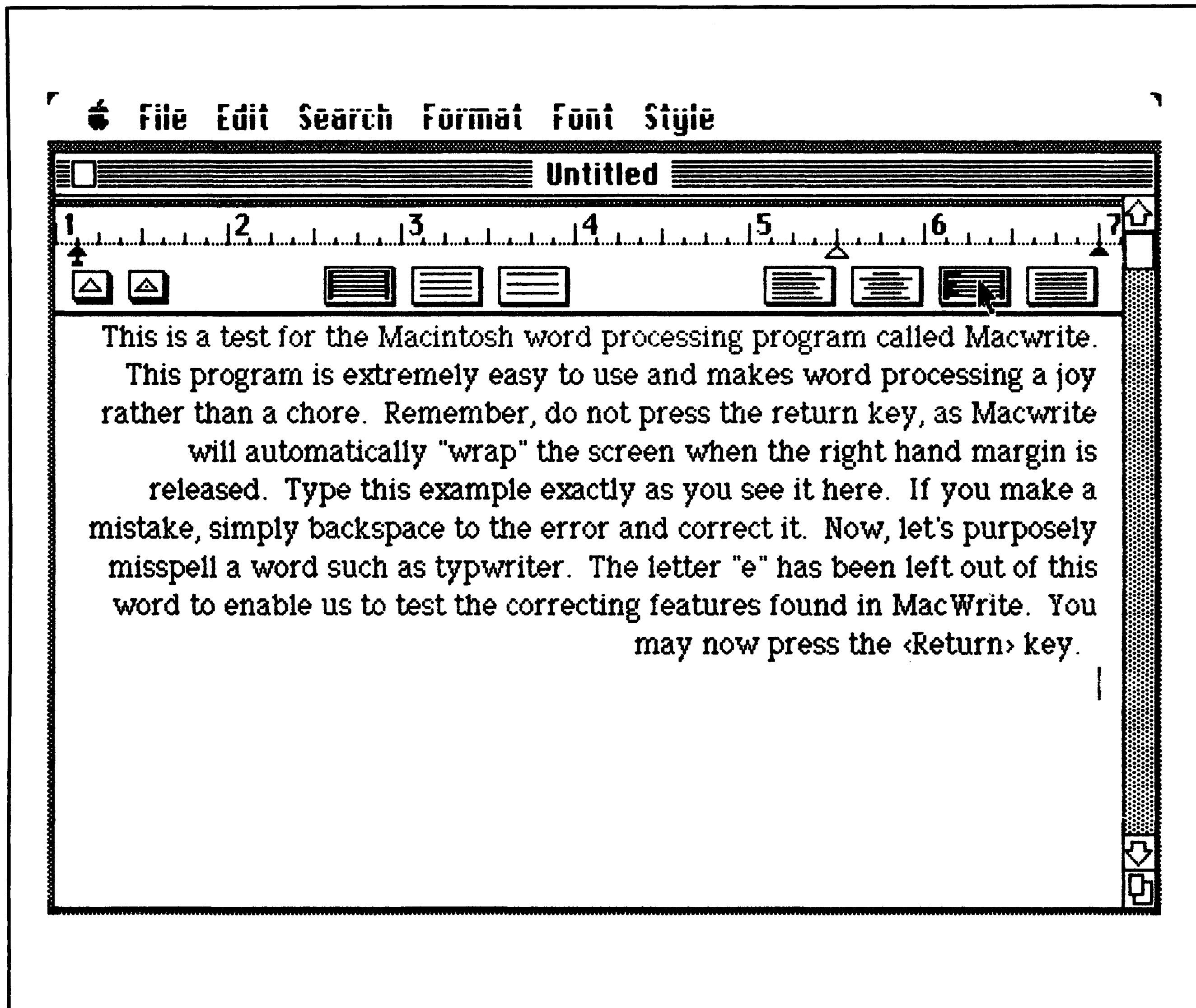


Fig. 3-7. The sample paragraph right justified.

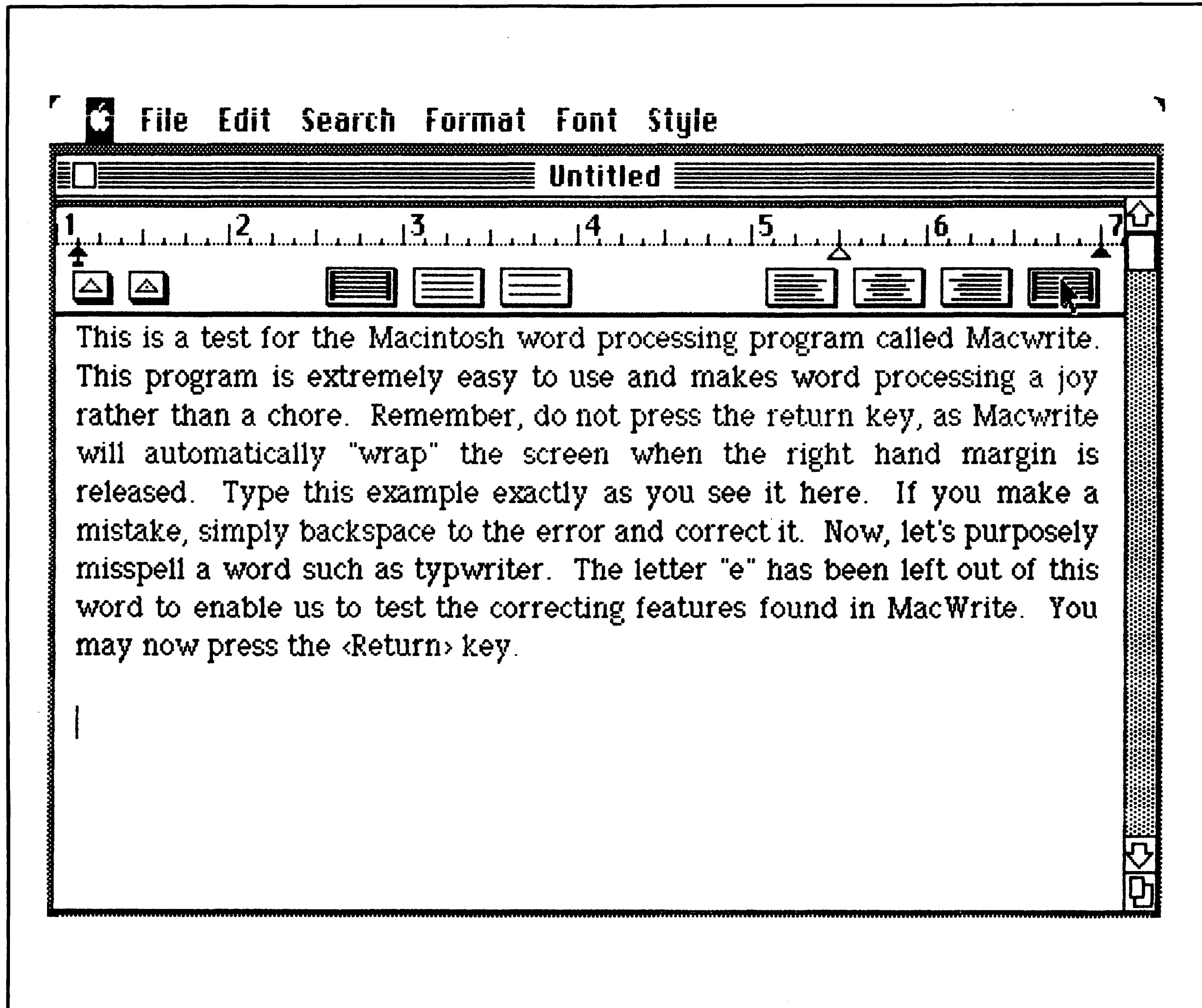


Fig. 3-8. The sample paragraph both left and right justified.

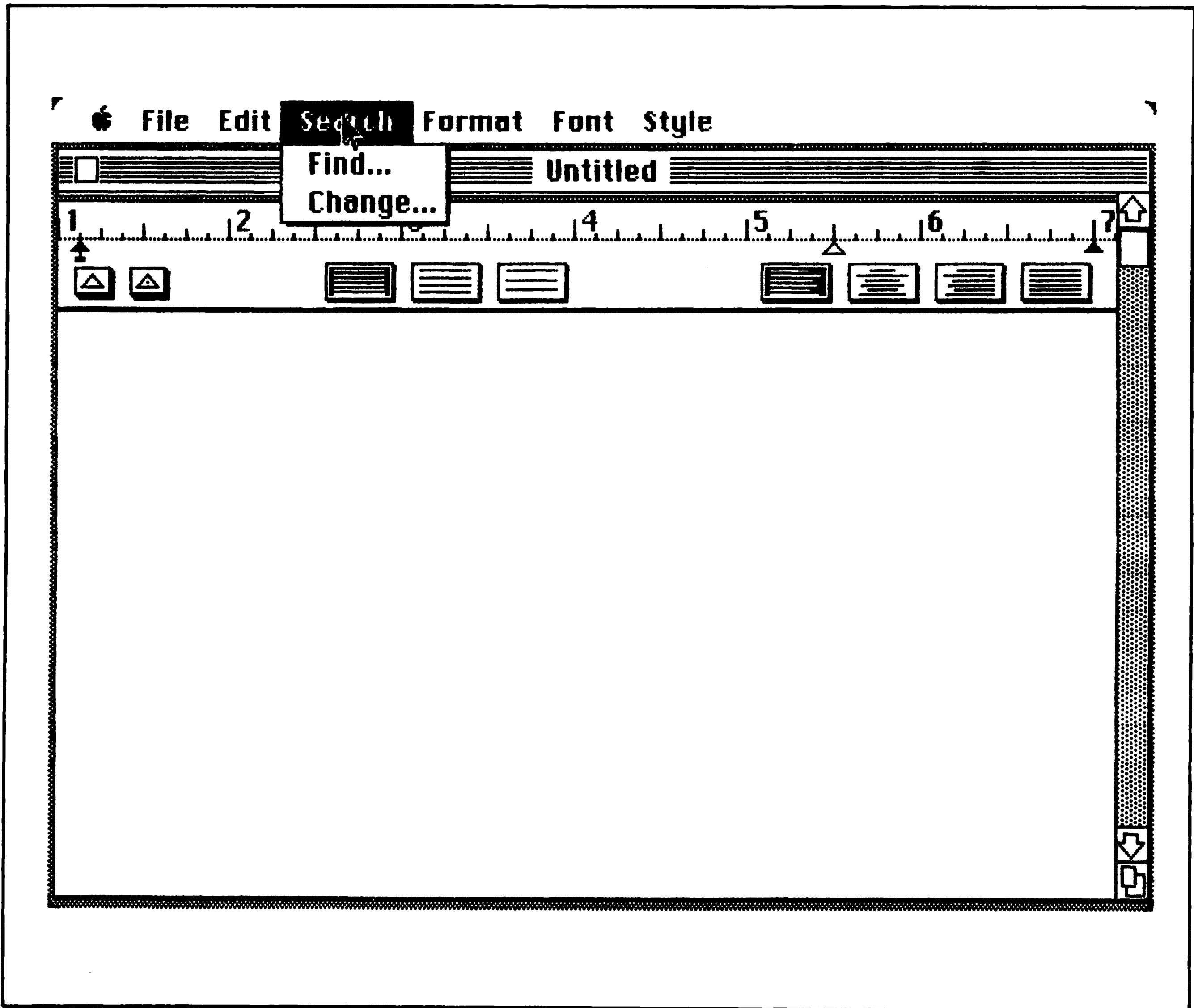


Fig. 3-9. The Search Menu, used to locate and/or change words or sentences in the text.

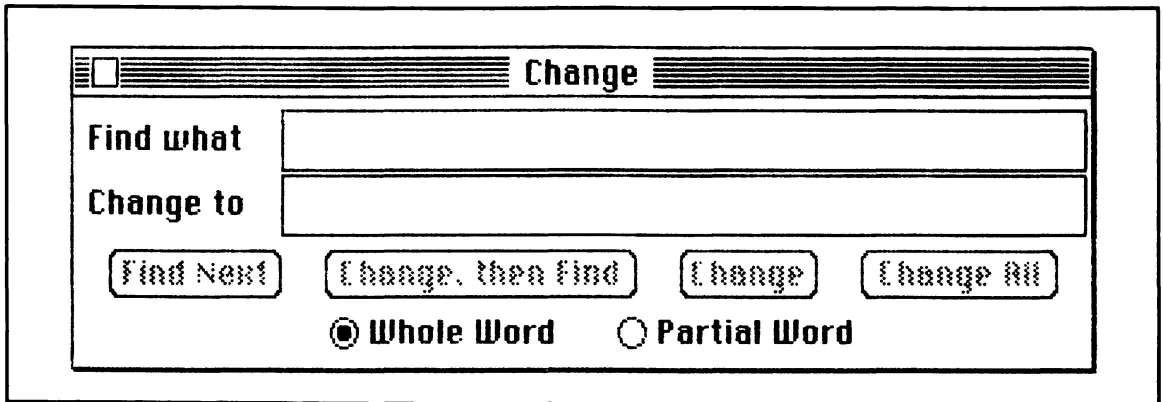


Fig. 3-10. The Change window.

in "typewriter" to place your cursor there on the screen. Use the backspace key to wipe out "typewriter" and then type "allow." Press RETURN. Click the Change option near the bottom of the window and exit the Change window. Click any blank spot on the document screen and the word "allow" will no longer be in reverse type.

The Find option may be used first to see if certain words appear in the text. You may then move on to Change. The latter is normally used more than the former, since the Change option includes the Find capability as well. Pull out the Search selection again and click Find. Click the Find What row and type in the word "the." Press RETURN. Move the pointer to the Find Next selection and click. Click again and keep clicking until you receive a window telling you "the was not found." The number of times you clicked the mouse is the number of times the word "the" was used in the document. You may have noticed that each time you click Find Next, the word "the" in the document was enclosed in a black box. By looking at the document, you can see just where the words you are looking for are located. Of course "the" does not often change meanings, but some words may, and you may want to change some occurrences without changing others. That is what this trick allows.

It is simple to move words, sentences, or even whole paragraphs to other locations in the document. To practice this function, let's interchange the first and second sentences in the text. Use the mouse to place the text cursor just before the word "This" in the second sentence of the sample paragraph shown in Fig. 3-13. Click the mouse. The cursor appears where the pointer was placed. Click the mouse again and hold. Begin moving the pointer toward the right side of the screen. You will see that the letters are in reverse. Don't go all the way to the end of the line, but back toward the left until the reverse disappears. The easiest way to move an entire line of text is to click the mouse at the beginning of the line and move down one line. When you do this, you will see that the entire line is in reverse. Move the next line up to one space past the period following "chore." Release the mouse button. Move the pointer up to the EDIT menu (Fig. 3-14) and click Cut. The sentence in reverse is removed from the text. Move the pointer to a point just before the letter "T" in the first word of this document and click once. Pull up the EDIT menu again and select Paste. The sentences have now been interchanged. The result is shown in Fig. 3-15. Before you do anything else, move the pointer to a blank spot on the screen and click. The reverse

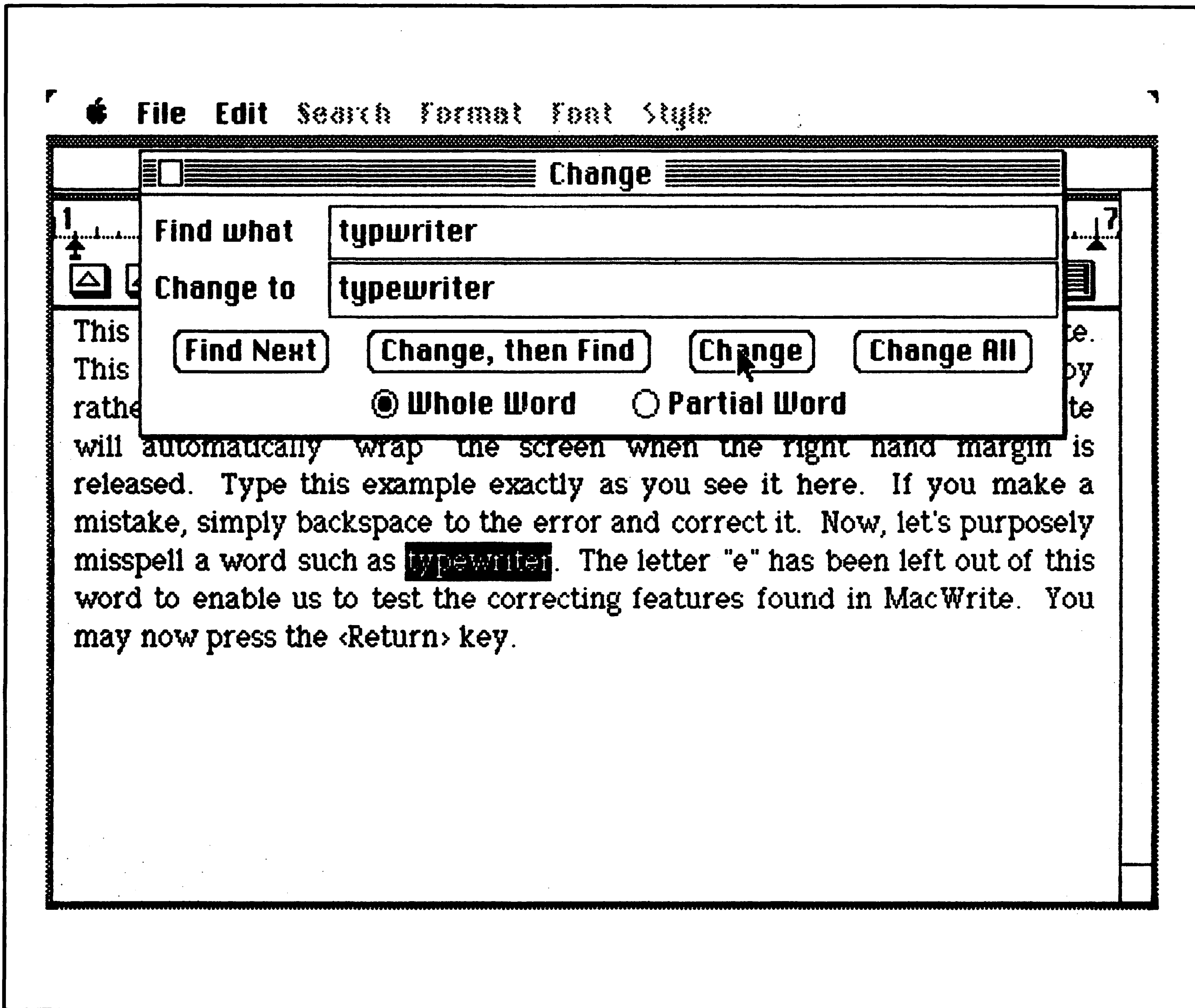


Fig. 3-11. Correcting the misspelled word "typwriter" with *MacWrite*.

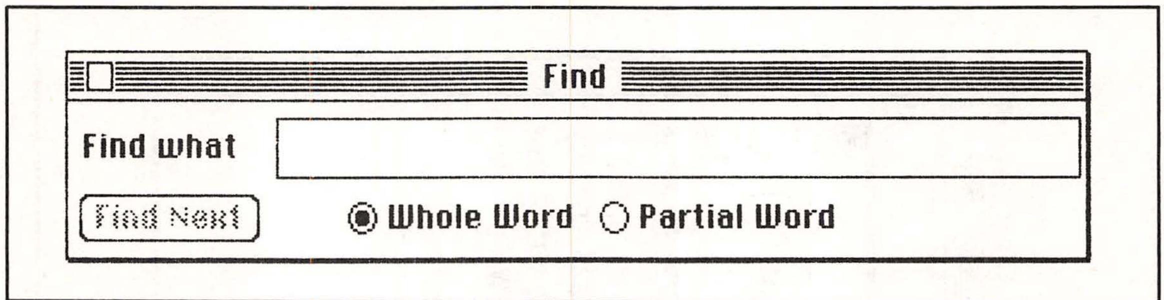


Fig. 3-12. The Find window.

type is now normal, and the change is complete. Using this method, you can move words, sentences, and whole paragraphs any where in your document.

INDENTING AND TABBING

On the surface of the ruler you will see several triangles that lie just below the marked line. The triangle at the far left seems to be fitted with a base, but it is actually two symbols combined. With the same document displayed on the screen, click and hold the black triangle just below the number 1 on the ruler. Move the mouse to the right and you will see that the icon with the base travels with the pointer. Move it to just beneath the number 2 on the ruler and release. Now the first sentence in your paragraph is indented to the point marked by the symbol, because it tells the computer to do an automatic indentation at the beginning of each paragraph. When the end of a paragraph is signaled (by pressing RETURN), the text cursor is moved down one line and indented to the marker point. When *MacWrite* is first entered, the indent marker is at the far left side of the ruler, so there is no indentation until you set one.

The alignment selections on the ruler have no effect on the paragraph indentation. For instance, if you click Left/Right Justify, all lines in the document will be justified at the left and right margins, except for those lines that are indented. These will be right justified only. Beneath the number 1 on the ruler is another pointer or symbol that is represented by a filled-in triangle. There is also another one at the far right of the screen. These are your margin pointers, and they work just like the margin controls on a typewriter. Click the left margin pointer and move it to 1.5 on the ruler. When you release the button, you will see that the entire left-hand margin of your document is aligned to that point. Click and move the right margin pointer and you will see the right margin shift as well. It is usually best to set these margins before you begin typing, as moving the right margin toward the left after the document is complete can sometimes preclude complete right justification. You will have noticed by now that in many ways, *MacWrite* allows the Macintosh to be used and set up just like a high-quality typewriter.

Also on the ruler scale are two triangular pointers that are not filled in. The one on the left contains a small dot, while the other is completely open. Using the mouse, place your text cursor near the bottom of the screen at a blank area and click. Press the TAB key and you will see that the cursor travels to the points on the screen that lie beneath these open triangles. These are your tab sets, and while only two are shown on the ruler scale, you will see two boxes toward the left of the ruler that contain these symbols. Using your mouse pointer, you can click these symbols and then move them to

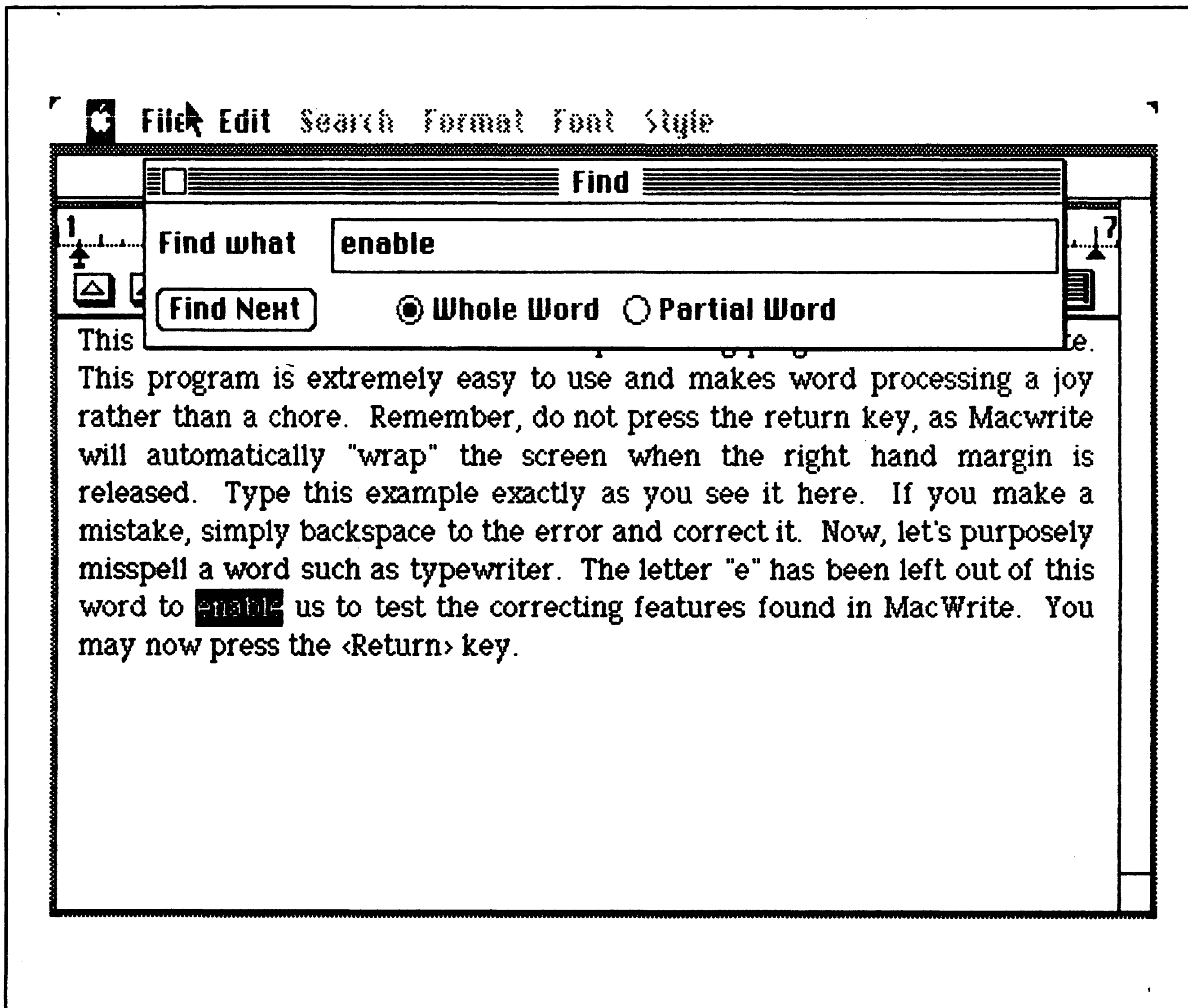


Fig. 3-13. The word "enable" is located by typing it in the Find window.

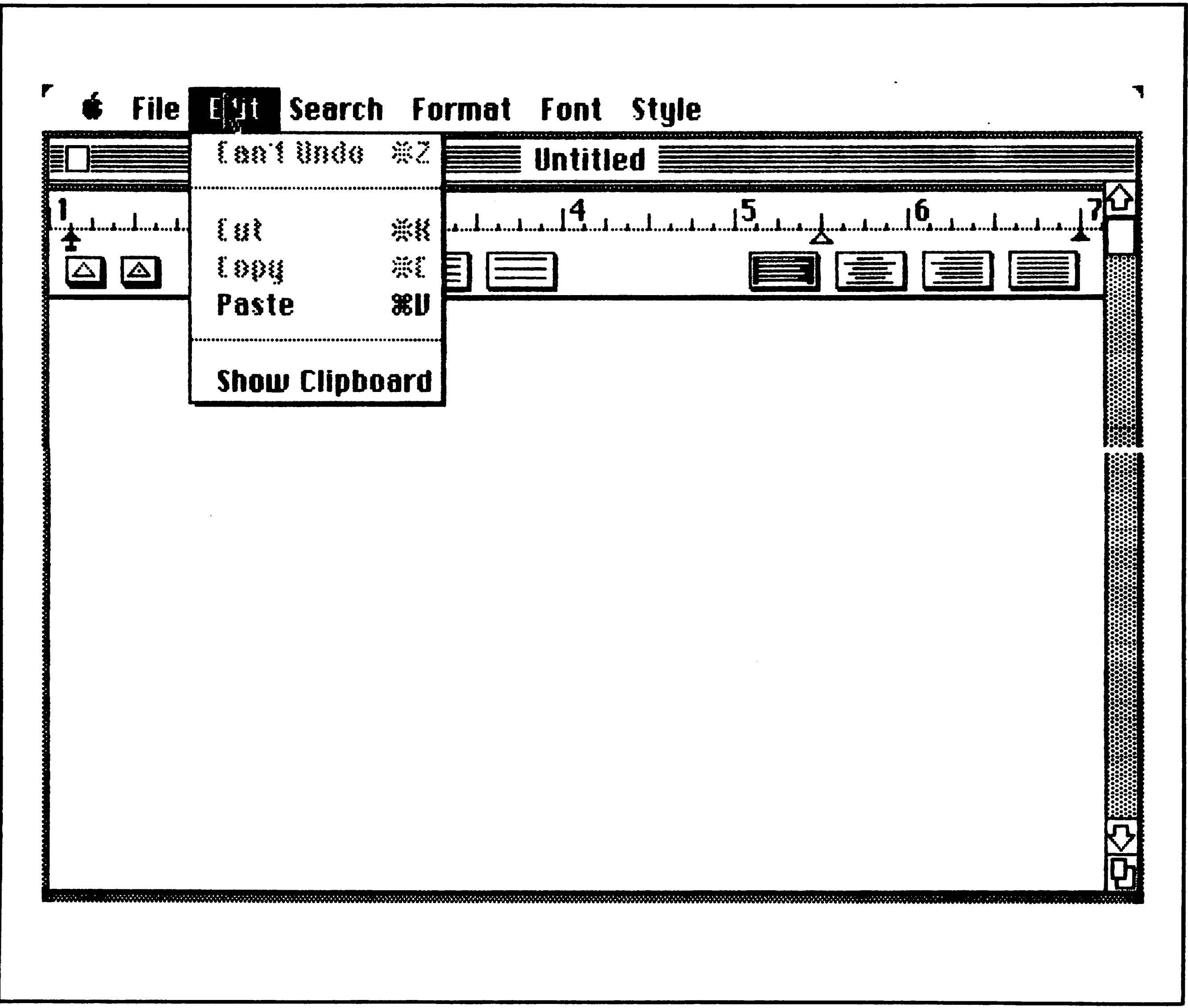


Fig. 3-14. The EDIT menu.

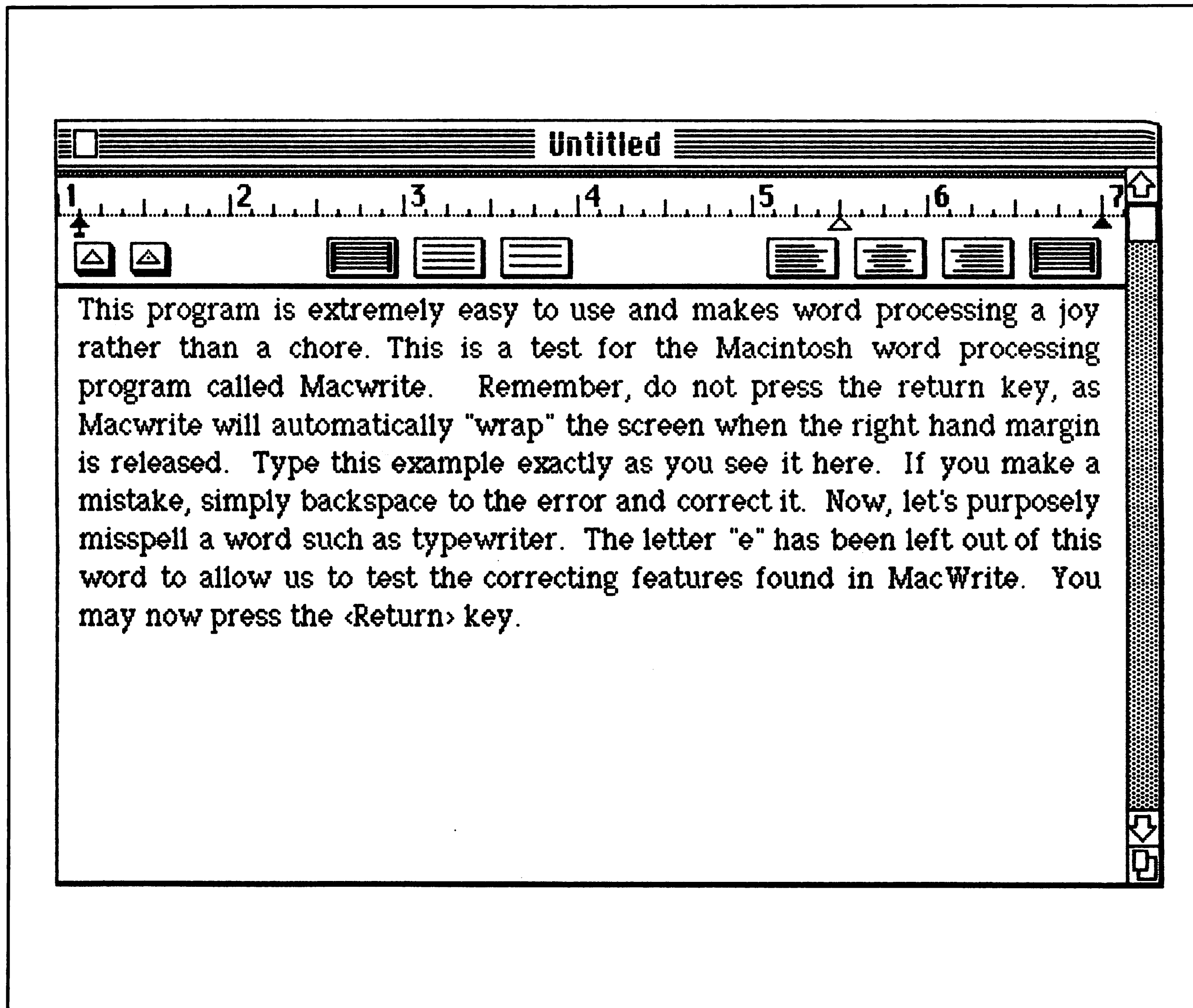


Fig. 3-15. The document, after the second sentence in the original paragraph has been moved to the first place position.

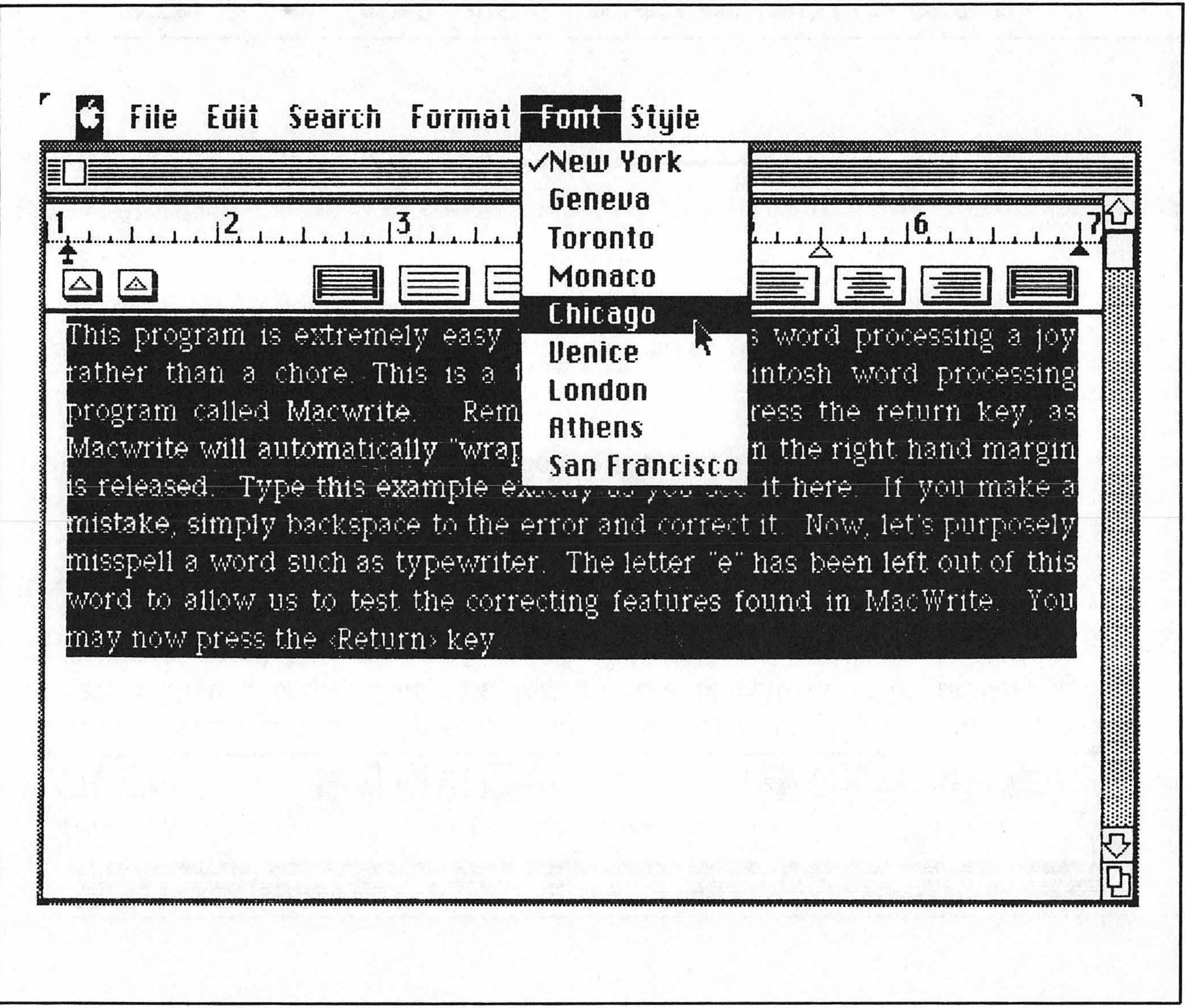


Fig. 3-16. Selecting a Chicago font from the FONT menu.

different points on the ruler. You can have as many tabs as you want.

By now you've probably noticed that some of the tabs are open and some contain a dot. This dot represents a decimal point and, appropriately enough, the marker is called the *decimal tab marker.* It sets up columns of numbers so that the decimal point is aligned with the tab. The open marker is called the *regular tab marker* and performs the same function as the tab key on a typewriter—it indents text or lines of text columns and is used for all text operations. Simply remove the decimal marker from its *tab well* at the left of the screen or from its location on the ruler and place it in the position the decimals are to be lined up in. When you no longer need this marker, click and remove it from the ruler scale. You can put it back in a tab well at the left of the screen, but this is not necessary. All you really need to do is drag the tab marker away from the ruler and release the mouse button.

CHANGING THE TYPEFACE

When *MacWrite* is first entered, the "default" *font* (the one that's there if you don't select another) is New York. The default *style* is Plain Text and the size is 12 Point. Until now, we've been writing in these default modes. Suppose you would like to make a change after the document is complete. This is easily accomplished. First, place the text pointer at the beginning of the first sentence. Click and hold the mouse button, moving downward until the entire document is in reverse. Release the mouse button and pull out the Font selection in the menu bar. For this test, select Chicago (Fig. 3-16). Release the mouse button and all of the text in reverse is converted to the Chicago font. Pull out the Font selection again and click London. Again, the text is changed, as shown in Fig. 3-17. Pull up Font one more time and click New York to return to the original font.

While your document is still in reverse, pull out the Style menu selection (Fig. 3-18) and click Bold. All text is now boldface. Pull out the STYLE menu again and click 18 Point. The document now appears in boldface, but instead of 12 Point, it's blown up to 18 Point (Fig. 3-19). Select some of the other options in the STYLE menu and the FONT menu. You can easily see that many different typefaces and font styles are available under *MacWrite.*

This program is extremely easy to use and makes word processing a joy rather than a chore. This is a test for the Macintosh word processing program called Macwrite. Remember, do not press the return key, as Macwrite will automatically "wrap" the screen when the right hand margin is released. Type this example exactly as you see it here. If you make a mistake, simply backspace to the error and correct it. Now, let's purposely misspell a word such as typewriter. The letter "e" has been left out of this word to allow us to test the correcting features found in MacWrite. You may now press the □Return□ key.

Fig. 3-17. The text change resulting from selecting the London font.

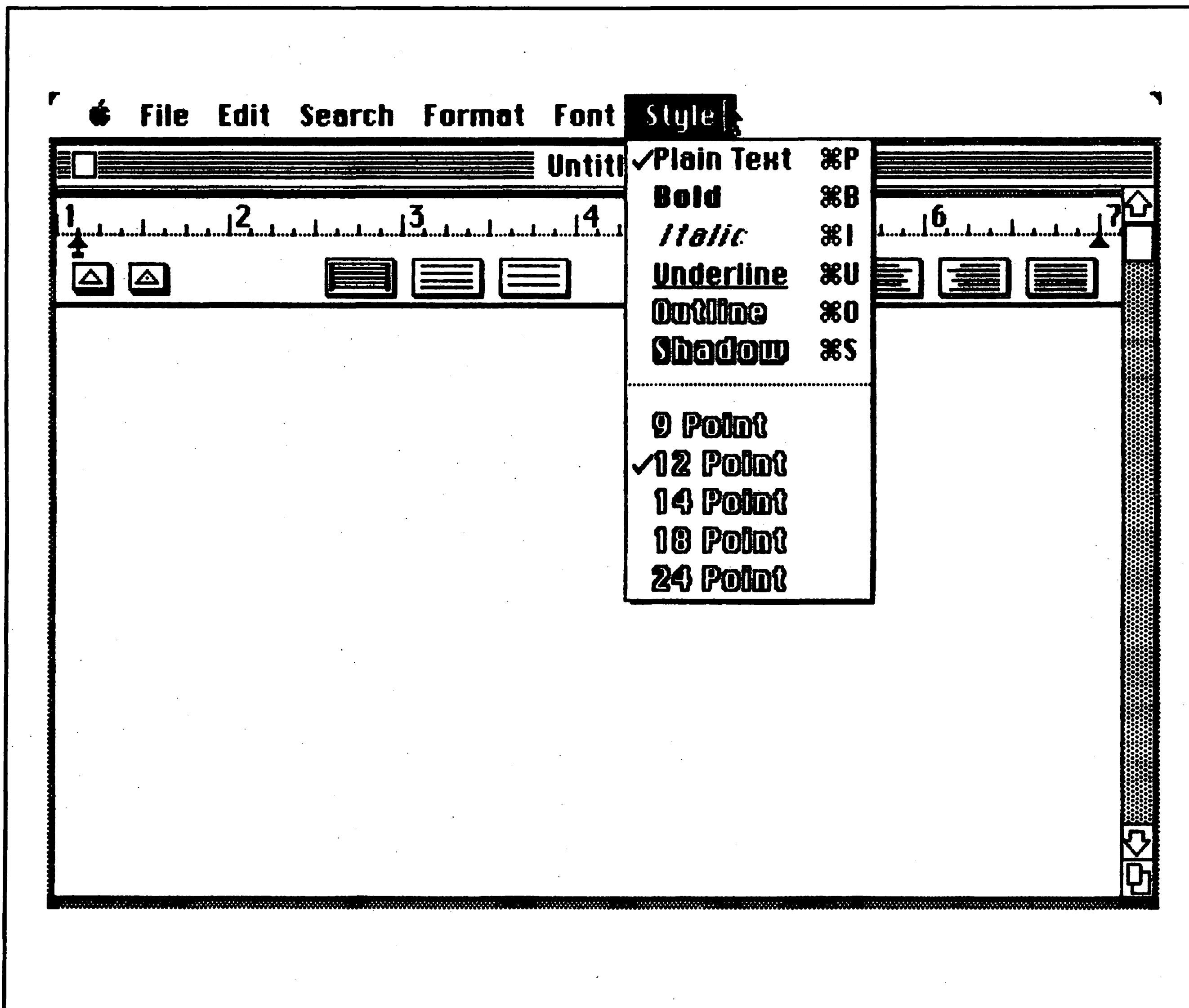


Fig. 3-18. The STYLE menu allows you to not only change text style but type size as well.

This program is extremely easy to use and makes word processing a joy rather than a chore. This is a test for the Macintosh word processing program called Macwrite. Remember, do not press the return key, as Macwrite will automatically "wrap" the screen when the right hand margin is released. Type this example exactly as you see it here. If you make a mistake, simply backspace to the error and correct it. Now, let's purposely misspell a word such as typewriter. The letter "e" has been left out of this word to allow us to test the correcting features found in MacWrite. You may now press the <Return> key.

Fig. 3-19. Text that has been printed in 18 Point type.

When you finish experimenting, return the typeface to the default mode (New York, Plain Text, 12 Point), and click the screen.

PRINTING YOUR DOCUMENT

Assuming that the document has been completely edited and set up with the typeface desired, you can now print it on the Apple ImageWriter printer. Simply select File in the menu bar (Fig. 3-20) and click Print. Make sure your printer is on-line and that you have paper wound onto the platen. If the printer is operated without paper, the print head and the platen can be damaged.

As soon as Print is clicked, the disk drive will whir and you will be presented with an option window that allows you to choose the quality of print, the page range, the number of copies, and the paper feed mode (Fig. 3-21). For the first test, select Draft by placing the mouse pointer in the circle

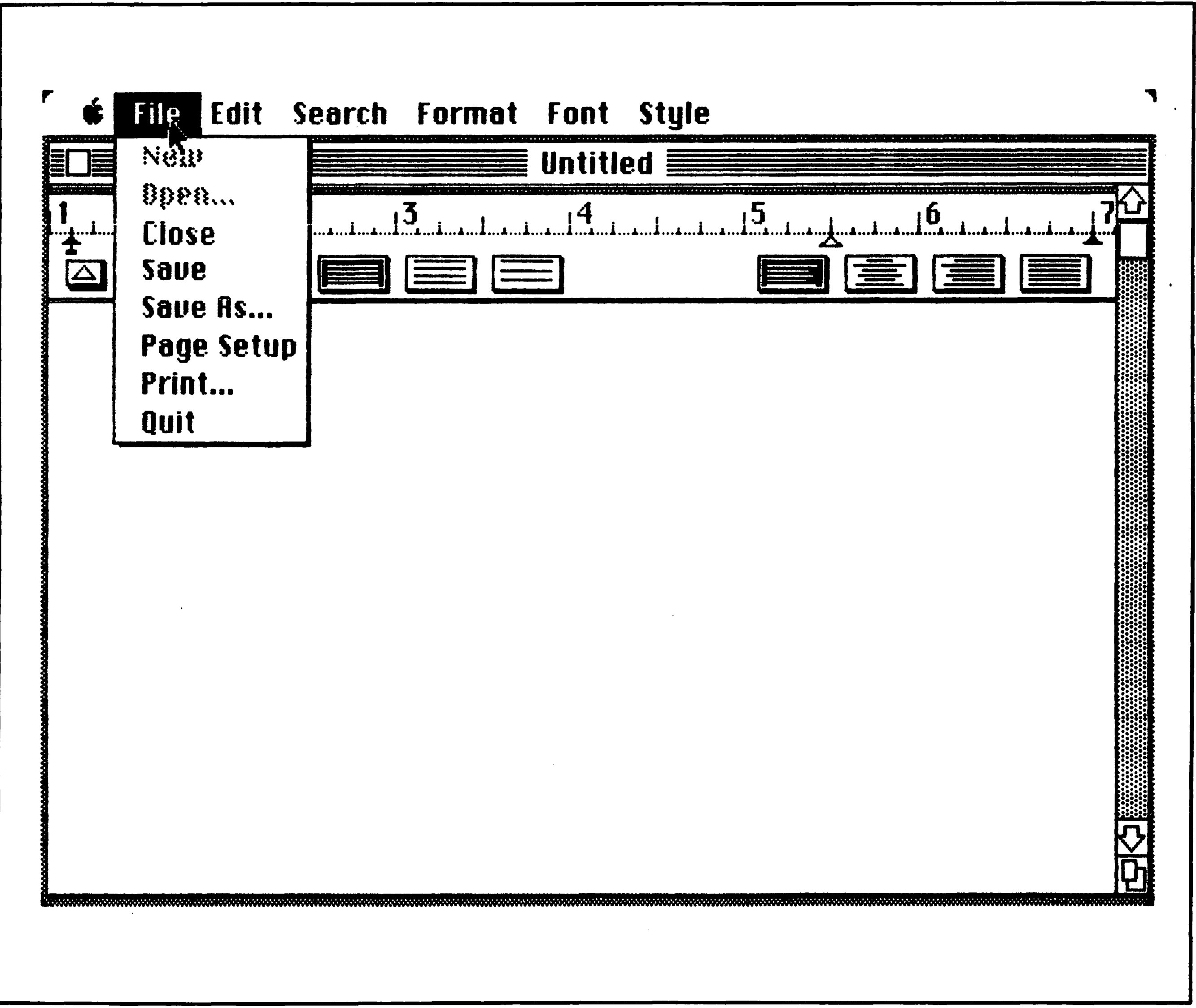


Fig. 3-20. The *MacWrite* FILE menu.

Quality: (●) High ( ) Standard ( ) Draft [OK]

Page Range: (●) All ( ) From: [ ] To: [ ]

Copies: [1]

Paper Feed: ( ) Continuous (●) Cut Sheet [Cancel]

Fig. 3-21. The Print option window allows you to choose High, Standard, or Draft Quality type. Controlled Printing is also offered, and you can specify the number of copies desired.

preceding Draft and clicking once. Forget about the other options for now. Move the pointer to the box in the upper right corner of the screen marked OK and click once. You will receive a prompt telling you that printing is about to begin. Figure 3-22 shows the results of a Draft print.

When the printer stops, your document is still displayed on the screen, so let's print it again using a better quality of print. Pull out the File menu and click Print. This time, click the Standard mode and click OK. Figure 3-23 shows the same document in standard Print mode. Notice that there is a tremendous difference in the quality of the two documents. Draft copies are used only for hard copy editing. The standard print is quite good and would be suitable for informal letters or office memos.

Let's do one more printout. Access the File menu and click Print again. This time, select High Quality. You will see a prompt telling you that your document is being saved to disk, as was the case with the Standard mode. Shortly thereafter, the printer will begin printing. Figure 3-24 shows the results. The Draft mode does not do any processing of the document, whereas the other two modes commit the information to disk, so proportional spacing, alignment, and other features are as they appear on the screen. The High Quality mode should always be used for important communications because the Apple ImageWriter, although a dot-matrix printer, is capable of displaying letter

This program is extremely easy to use and makes word processing a joy
rather than a chore. This is a test for the Macintosh word processing
program called MacWrite. Remember, do not press the return key, as
MacWrite will automatically "wrap" the screen when the right hand margin
is released. Type this example exactly as you see it here. If you make a
mistake, simply backspace to the error and correct it. Now, let's purposely
misspell a word such as typewriter. The letter "e" has been left out of this
word to allow us to test the correcting features found in MacWrite. You
may now press the <Return> key.

Fig. 3-22. Draft Quality hard copy.

This program is extremely easy to use and makes word processing a joy rather than a chore. This is a test for the Macintosh word processing program called MacWrite. Remember, do not press the return key, as MacWrite will automatically "wrap" the screen when the right hand margin is released. Type this example exactly as you see it here. If you make a mistake, simply backspace to the error and correct it. Now, let's purposely misspell a word such as typewriter. The letter "e" has been left out of this word to allow us to test the correcting features found in MacWrite. You may now press the <Return> key.

Fig. 3-23. A printout in Standard Quality.

quality text in the High Quality mode. The quality is not as good as that of a daisy wheel printer, but it's faster and cheaper. The fastest print mode is Draft, followed by Standard, and then High Quality. In High Quality mode, each portion of a line is printed several times to darken the text. This fills in many of the holes, and from a short distance, makes the text look almost letter quality, but it takes much longer than the other two, and uses much more of the ribbon than the other two modes. If you do all your printing in High Quality mode, your ribbon will not last as long as it would using the other two modes.

SAVING YOUR DOCUMENT

To save a document to disk, select the appropriate suboption in the FILE menu. This menu will also allow you to close out an old document, open a new one, set up a page in many different styles, or exit *MacWrite* altogether.

This program is extremely easy to use and makes word processing a joy rather than a chore. This is a test for the Macintosh word processing program called MacWrite. Remember, do not press the return key, as MacWrite will automatically "wrap" the screen when the right hand margin is released. Type this example exactly as you see it here. If you make a mistake, simply backspace to the error and correct it. Now, let's purposely misspell a word such as typewriter. The letter "e" has been left out of this word to allow us to test the correcting features found in MacWrite. You may now press the <Return> key.

Fig. 3-24. A printout of High Quality print mode.

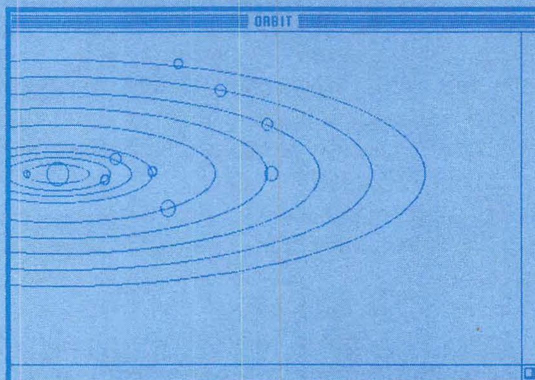
SUMMARY

To say that *MacWrite* is versatile is an understatement. It does what it was designed to do in an extremely efficient manner and allows the user to interact comfortably with the machine. Then again, this is the whole story behind Macintosh. The computer is friendly not just because of the way it was designed, but also because of the way its software accesses its built-in features and interacts with the user.

With *MacWrite,* you can turn out professional-looking memos, letters, and documents within an hour, while most other word processors require that you spend hours, days—even weeks—learning to use the software!

The ImageWriter printer, while not letter quality, does an excellent job of producing highly readable text that borders on letter quality. You can also save printing time and ribbon wear by choosing one of the two lower-quality print modes. Like *MacPaint, MacWrite* is easy to learn. I hope the discussion in this chapter has allowed you to progress even faster.

Chapter 4



Using The Macintosh Finder

The Apple Macintosh has its disk operating system (DOS) written into ROM rather than requiring you to load it from a disk file. The DOS, which allows your computer to access disk files and perform the task of instructing the computer, is a program just as *MacWrite* and *MacPaint* are programs.

One part of the DOS is called *Finder*. Finder is your computer's built-in application program. It directs traffic between you and the disk. Apple describes the Finder as a central hallway in the Macintosh house, that manages moves from one room (application) to another and organizes the documents you create with those applications. Like a front door—its the way you enter the Macintosh house—the link between you and your disks.

The Finder is in charge of organizing files on the disk. The Finder opens, closes, discards, copies, moves, and renames those files (usually referred to as *documents*).

You use the Finder every time you insert a disk in the drive and turn the computer on. Do this now with your *MacWrite/MacPaint* disk, and you should see a display similar to the one shown in Fig. 4-1. The Names of the documents are shown in the display window. In the figure, the main window indicates the documents contained on this disk. Other information is also provided in this window. In the top margin, the Finder tells me that I have six items (documents) on the disk, 352K of memory already used, and 48K still available.

If you have many documents on your disk, the window may not be large enough to display icons for all of them; but as with most Macintosh windows, it can be enlarged either vertically, horizontally, or in both directions by using the Size box in the lower right corner of the window. Figure 4-2 shows how the window is expanded using the size box.

THE FINDER MENU BAR

The Finder menu bar is at the top of the screen. There are five menus that can be selected from the menu bar. These are the APPLE, FILE, EDIT, VIEW, and SPECIAL. Let's take a look at each of them.

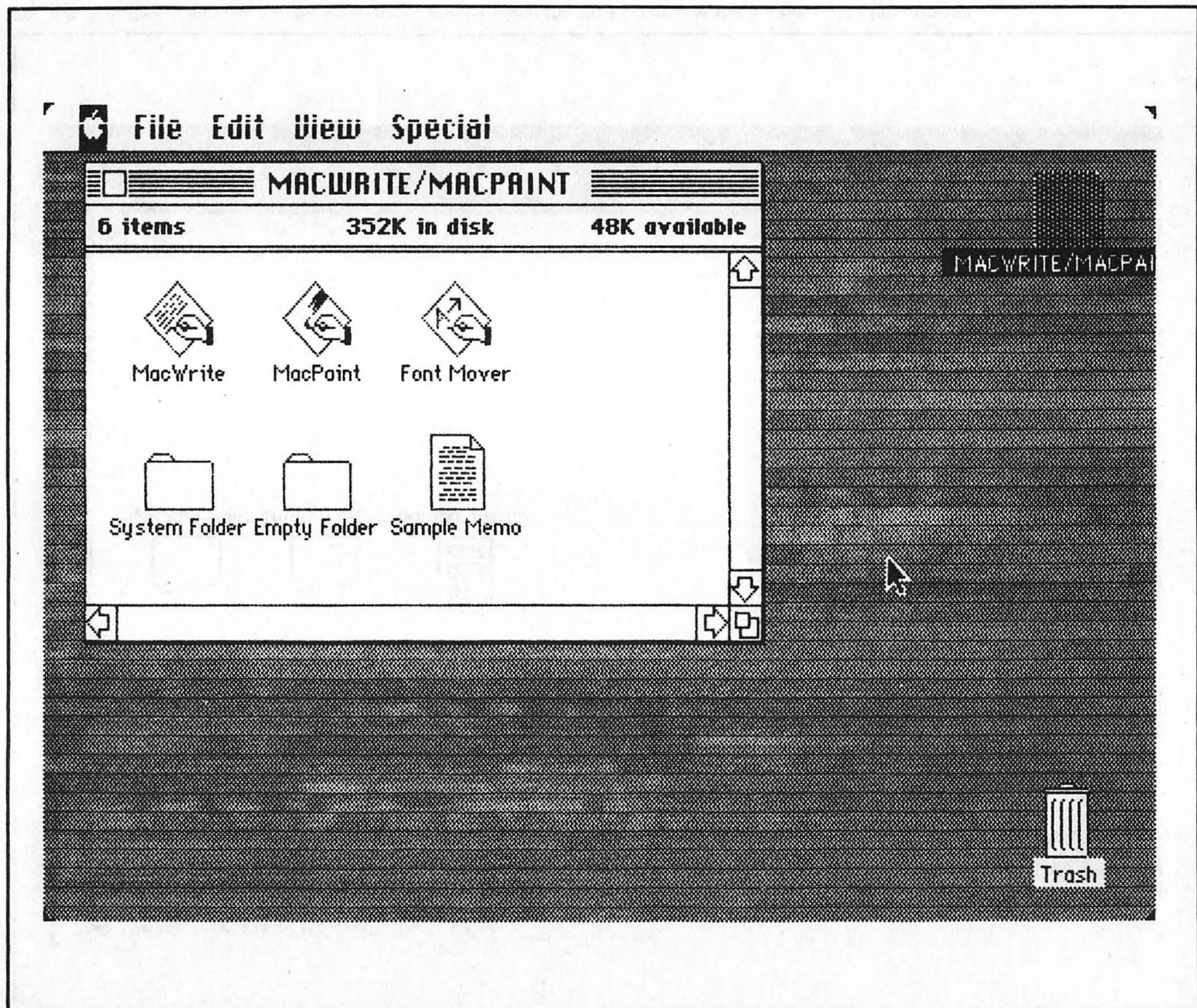


Fig. 4-1. The Macintosh Finder is entered whenever a disk is inserted and the computer is turned on.

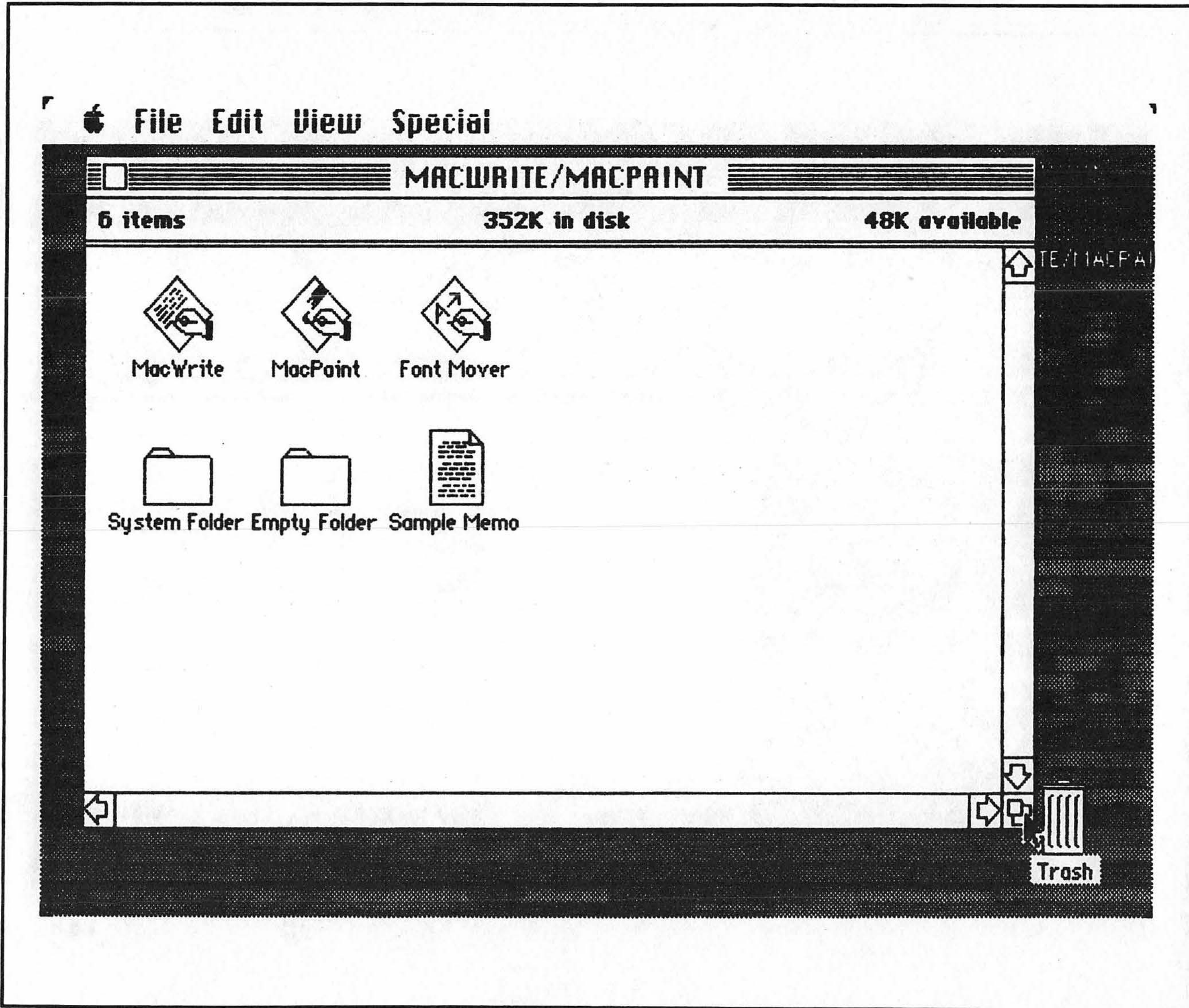


Fig. 4-2. The Finder window can be enlarged or shrunk using the Size window at the bottom right of the screen.

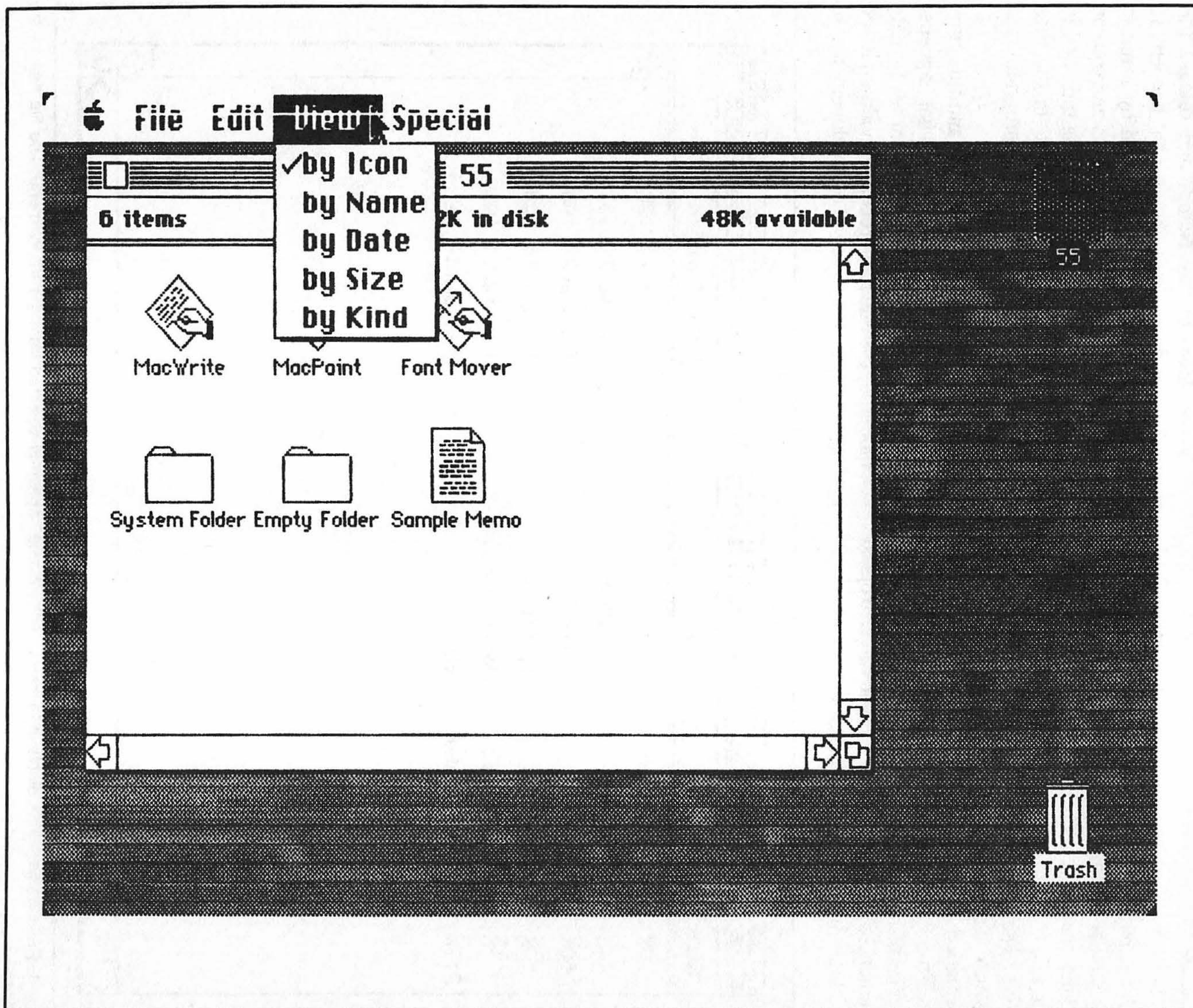


Fig. 4-3. The VIEW menu allows you to display disk information in five different ways. The window is currently displaying items by icon.

The VIEW Menu

Let's look at the sub option labeled VIEW. Click and pull out this option now. Your screen will look similar to that shown in Fig. 4-3. Here the working copy of *MacPaint/MacWrite* has been renamed "55", to fit in my personal disk cataloging system. Don't be confused by the renaming; the information contained on this disk is the same as the information on the original *MacPaint/MacWrite* disk. From this drawing, you can see that there's a checkmark to the left of "by Icon." The window documents are listed by icon, but you can also opt to have them listed by name, date, size, or kind.

Move down to "by Name" and release the mouse button. Figure 4-4 shows what happens to the screen. All documents are listed by name in alphabetical order. The icons are gone, and you can see the last date of each program was modified in the far right column. To the far left, in the Size column, is the disk memory each program consumes.

Pull the VIEW menu out again and this time select "by Date." Figure 4-5 shows what happens. The programs are now listed in the order by date last modified, the one most recently modified listed first. Here, we can see that all of the documents on

MACWRITE/MACPAINT

Size	Name	Kind	Last Modified
0K	Empty Folder	folder	Tue, Jan 24, 1984
13K	Font Mover	application	Tue, Jan 24, 1984
60K	MacPaint	application	Tue, Jan 24, 1984
53K	MacWrite	application	Tue, Jan 24, 1984
3K	Sample Memo	MacWrite document	Tue, Jan 24, 1984
217K	System Folder	folder	Wed, Jan 18, 1984

Fig. 4-4. Previous file items displayed when the "by Name" option is selected from the VIEW menu. The file names are displayed in alphabetical order.

MACWRITE/MACPAINT

Size	Name	Kind	Last Modified
53K	MacWrite	application	Tue, Jan 24, 1984
13K	Font Mover	application	Tue, Jan 24, 1984
60K	MacPaint	application	Tue, Jan 24, 1984
3K	Sample Memo	MacWrite document	Tue, Jan 24, 1984
0K	Empty Folder	folder	Tue, Jan 24, 1984
217K	System Folder	folder	Wed, Jan 18, 1984

Fig. 4-5. Disk contents displayed by Size (the amount of memory each requires). The files are listed from largest to the smallest.

this disk were last revised on Tuesday, January 24, 1984, except the System Folder, which was modified Wednesday, January 18, 1984. The date each program is saved or modified is automatically supplied by the Macintosh based on its internal calendar.

Pull the VIEW menu out again and select "by Size." Figure 4-6 shows the way the documents are now listed—according to the amount of memory each consumes on disk, the largest document listed first.

The last *View* option allows you to display documents by "kind", that is, by category. Folders are listed first, followed by applications, and then documents, as shown in Figure 4-7. To return to the icon display again, simply select "by Icon" from the VIEW menu.

The APPLE Menu

Now let's have some fun. The Finder makes the APPLE menu available in every application program. The Macintosh APPLE menu is identified by the flashing Apple symbol in the upper left corner of the menu bar. Figure 4-8 shows the selections from this menu. If you select Scrapbook, you will see something similar to what is shown in

MACWRITE/MACPAINT

Size	Name	Kind	Last Modified
217K	System Folder	folder	Wed, Jan 18, 1984
60K	MacPaint	application	Tue, Jan 24, 1984
53K	MacWrite	application	Tue, Jan 24, 1984
13K	Font Mover	application	Tue, Jan 24, 1984
3K	Sample Memo	MacWrite document	Tue, Jan 24, 1984
0K	Empty Folder	folder	Tue, Jan 24, 1984

Fig. 4-6. Disk contents displayed by size (the amount of memory each requires). The files are listed from largest to the smallest.

Fig. 4-9, which shows the Scrapbook window. The Scrapbook presently contains nothing, as indicated by the display in its center. The Scrapbook is used to store text selections as well as pictures that you may want to transfer from *MacWrite* or *MacPaint,* or even from Microsoft BASIC! To place an item in the Scrapbook, you cut or copy the item using the EDIT menu. You may then use the EDIT menu to cut or copy something from the Scrapbook and paste it in another application program. To get rid of the Scrapbook window, simply click the Exit box in the upper left corner.

From the APPLE menu, you can also select Alarm Clock. This will display a clock on the screen just like the one shown in Fig. 4-10. Notice that there is an Exit box to the left of this window and an image to the right that looks something like a musical note. If you click the icon to the right, your display will look like that shown in Fig. 4-11. Now you can change the time the alarm is set for, or reset the time or date. To reset the time, click the icon on the bottom left (the wall clock face). In the row above you will see the current time display keeping pace with the original display in the upper row. Your cursor will convert to a crosshair to allow you to reset the clock. Place the crosshair on the hour in the center row. Click once and that portion will be shown in reverse. Now type in the new hour on the

keyboard. To change the minutes, click that part of the display and repeat the process. To change from AM to PM, click AM and another icon will appear just to the right of this designation. Click this icon to switch to PM. When you're finished, hit RETURN. You can also use the AM/PM icon to adjust the hours or minutes portion of the time display. The upper arrow moves the time ahead, the lower arrow moves it back.

Change the clock date by clicking the center icon in the bottom row. The date will be shown in the center row. Then use the same methods to alter the date as you used to alter the time.

To set the alarm, click the alarm clock icon in the bottom right row and change the alarm time. Move to the icon in the left center row and click once. This activates the sound mechanism. When the clock reaches the alarm time, a beeper will sound. Figure 4-12 shows all three set modes.

The Note Pad does what its name implies. It displays a Note Pad window on the screen, as shown in Fig. 4-13. Use this Note Pad for jotting down notes to yourself. When you wish to turn the page, simply place the mouse pointer on the dog-eared flap at the lower left of the window and click once. The page will flip. If you wish to go back a page, place the pointer on the page portion that lies below and to the left of the dog-eared flap and click.

MACWRITE/MACPAINT

Size	Name	Kind	Last Modified
217K	System Folder	folder	Wed, Jan 18, 1984
0K	Empty Folder	folder	Tue, Jan 24, 1984
60K	MacPaint	application	Tue, Jan 24, 1984
53K	MacWrite	application	Tue, Jan 24, 1984
13K	Font Mover	application	Tue, Jan 24, 1984
3K	Sample Memo	MacWrite document	Tue, Jan 24, 1984

Fig. 4-7. The "by Kind" display from the VIEW menu sorts files by type.

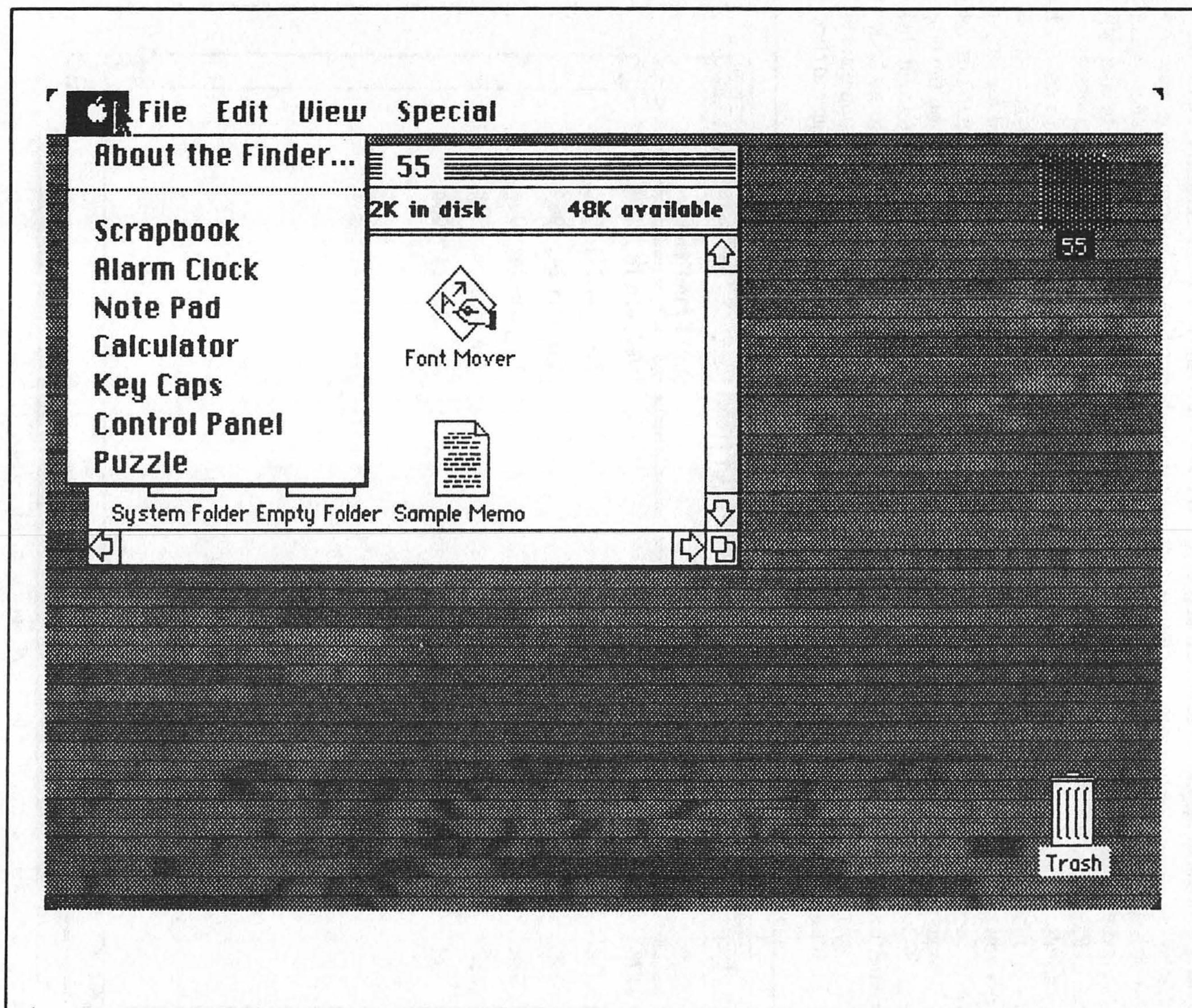


Fig. 4-8. The APPLE menu.

Fig. 4-9. The Scrapbook window is used to hold items that may be moved later.

The previous page is then displayed. Any notes entered to the Note Pad are saved in memory and can be referred to later.

The next APPLE menu sub option is my favorite. It's the Calculator, shown in Fig. 4-14. It's used just like the real thing except that instead of punching the keys with your fingers, you direct the mouse pointer to the appropriate key and click. With this calculator, you can add, subtract, multiply, and divide. Since it behaves just like any electronic calculator, you don't have to exert any effort to learn how to use it.

Another favorite of mine is the Control Panel, shown in Fig. 4-15. As its name implies, it controls something. It looks like a toy, but it's not; it actually does control many of the Macintosh's responses, and it saves the settings for future use. This one's going to take some explanation.

First, there is the speaker volume control on the far left. If you place your crosshair cursor on the slide bar, press and move up or down, the speaker volume is increased or decreased. Each time you release the button, you will hear the sound at its new volume setting.

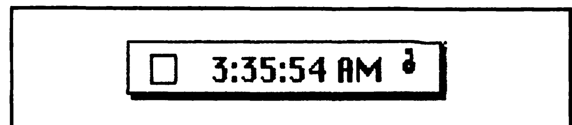


Fig. 4-10. The initial Alarm Clock window.

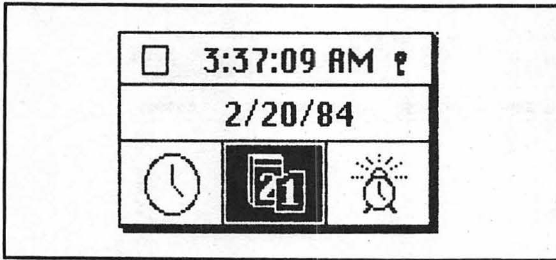


Fig. 4-11. The Alarm Clock Set window.

The center area contains a keyboard icon and to the right of it, two control rows. The top row determines the rate at which a key will repeat. You've probably noticed that when you hold down a key on the keyboard, that key is repeated over and

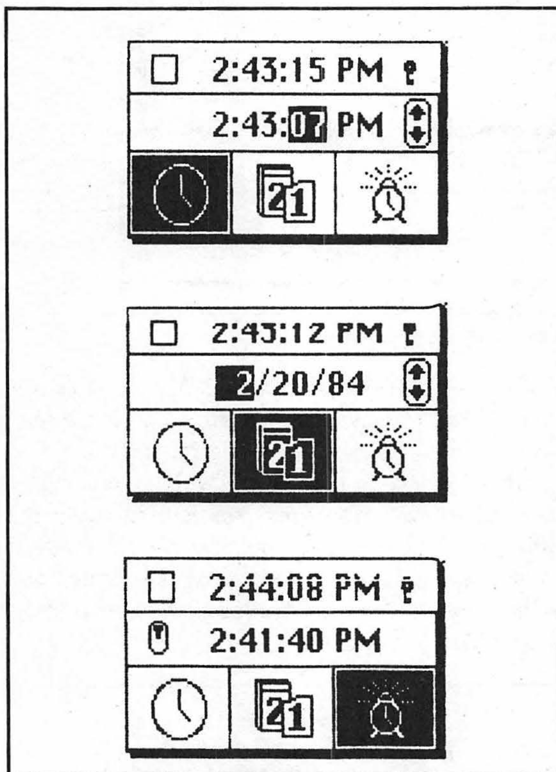


Fig. 4-12. Resetting time, date, and alarm setting on the Alarm Clock.

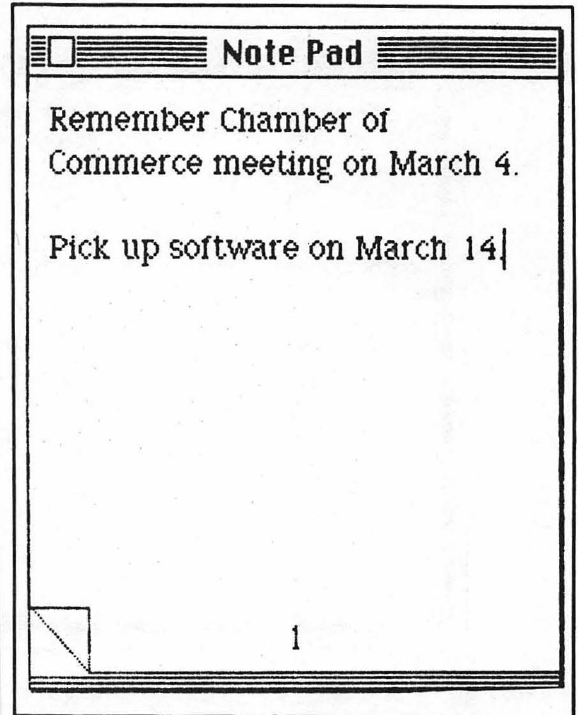


Fig. 4-13. The Note Pad is used to jot down reminders. The page is flipped by clicking the dog-ear with the mouse.

over on the screen. You can control the rate of repeat from a low factor of 0 to a high of 4. You may have noticed the tortoise and hare images to the left and right of the numbers. These indicate speed directions.

The bottom row sets the keyboard touch response. If you choose a low number, you will have to press a key for a longer period of time to get the letter than if you select a high number. To my knowledge, no other computer gives you this direct control over keyboard touch response.

To the lower left there is a box that contains a picture of the mouse and two option boxes marked 1 and 0. The 1 setting makes the mouse pointer move farther when you speed up mouse movement. When you want to jump from one part of the screen to another, you don't have to move the mouse as far as

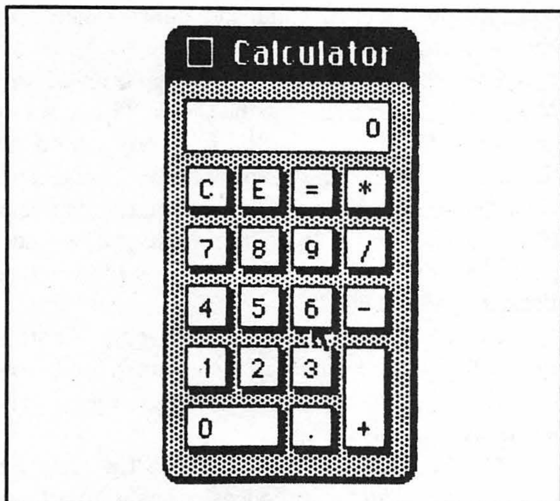


Fig. 4-14. The Macintosh Calculator works the same way as a Handheld one.

you normally would if this option is selected. If you select 0, the mouse tracking speed remains constant.

In the center bottom window are two boxes that can alter the desk top pattern. At present, your desk top is a medium gray, which is the default setting. By placing the crosshair on any of the dots in the left box, you can make them disappear by clicking. This adds more white space to your desk top, and it becomes lighter in color. To actually see the desk top, move to the right box and click the shaded area once. You will see how it appears in the background around your Control Panel. If you don't want to modify the current pattern, but would like to completely change the pattern, click the white portion of the right box and different options will become available. If you want to see each one on the whole screen, click the shaded area of the right box once more.

To the right of this box is the Mouse Button Double Click Speed Control. Clicking the leftmost square will allow you to double click fairly slowly. The middle box sets the mouse for a more moderate speed, while the right box sets it at a rapid rate. You

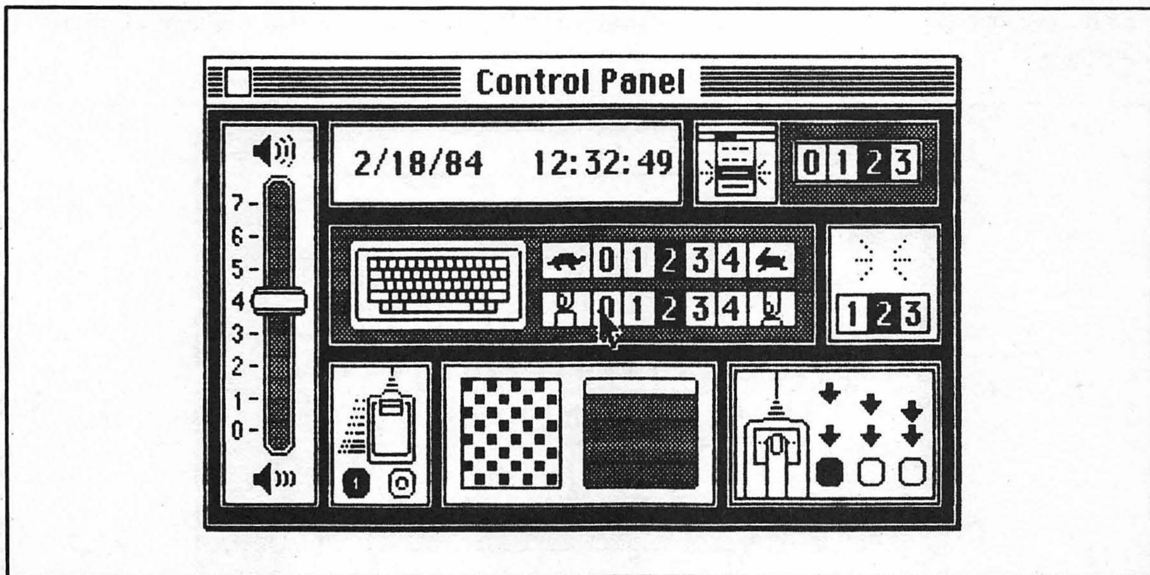


Fig. 4-15. The Macintosh Control Panel allows you to set speaker volume, keyboard sensitivity, mouse travel, and a number of other functions.

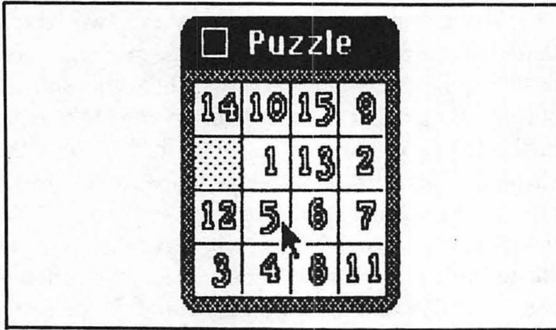


Fig. 4-16. The Macintosh Puzzle is a game that can keep you entertained for hours.

can set this speed control to suit your individual pace.

The center right box is the Rate of Insertion Point Blinking Control. Clicking the 1 will bring about the slowest blinking speed for this cursor, which lets you know where text will be displayed on the screen. The middle setting, 2, is the standard default setting, and 3 will double the standard speed.

In the upper right window is the Command Blinking Control. Here, you click 1, 2, or 3 to control how much a command blinks when you choose it from a menu.

Last on our round-the-screen tour of the Macintosh Control Panel is the Clock. This is set in just as the Alarm Clock is. Place the crosshair over the date or time section you wish to change and click the arrows that appear between the date and time to go forward or backward. Once you've made your changes, click any other point on the Control Panel to set the clock.

Now, you can probably see that the Control Panel allows you to do more than play. With it, you actually set the computer to respond to your personal style of interacting.

Another sub option in the APPLE menu is called Puzzle, and it produces a game board, as shown in Fig. 4-16. This is a graphic display of a popular handheld game containing 15 numbered squares and 16 placeholders. The numbers are jumbled and you are to put them in order again by clicking the squares adjacent to the one vacant square. As you click one square, it will slide into the single vacant space, just as in the mechanical version of the game.

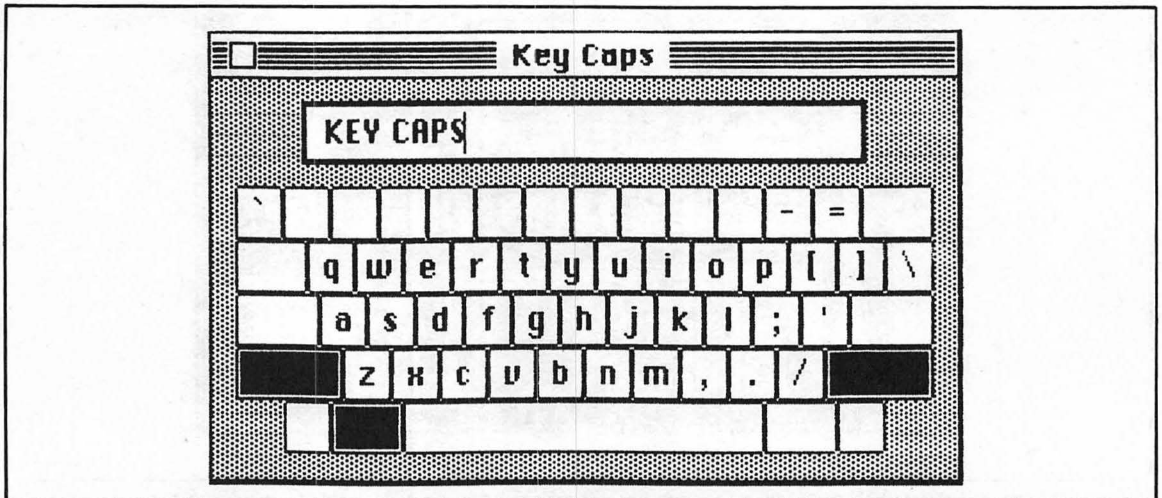


Fig. 4-17. The Key Caps display helps you learn the keyboard.

The Key Caps selection from the APPLE menu is shown in Fig. 4-17. It shows you how each key on the board changes when a different one of the Global Keys—the Option, Shift, or Caps Lock keys—is pressed. On the keyboard, if you press the Option key, all of the characters that are affected are displayed on the Key Caps representation. This allows you to become more familiar with the keyboard without having to try everything on a hit or miss basis. You may either type letters via the keyboard or press the mouse arrow to do the same thing.

The APPLE menu is a combination of things. First, it gives you access to programs that allow for personal tailoring of the keyboard, mouse, and desk surface. Second, it provides several learning aids to help beginners master the system quickly. Third, there are a few fun items that also help the user get more from the highly proficient and capable machine.

The FILE Menu

When you open the FILE menu, you will be given eight sub options, as shown in Fig. 4-18. Each of these has a simple but distinct purpose and will quickly help you manage files and disk. Open is used just as it is in *MacPaint* and *MacWrite*. It starts the application. To do this, you can click the icon of the document that you want to open, click the FILE menu, and select Open. But all you really have to do is click the document icon twice to produce the same result a little more quickly.

To select Duplicate, again, click the document first. The document you click is duplicated on the same disk. Duplicating a folder makes a copy of it and everything in it, even if it's currently on the desk top.

If you select Get Info a window opens that displays information about the document. This includes the kind of file, the size, and the date it was created, much like the displays in VIEW menu, only for one document at a time.

The Put Back selection places the selected documents back in its original location. Close closes the active window and returns you to whatever mode you were in prior to opening that window. You can also select Close All, which closes all windows to their icons. All of the desk accessories disappear, and only the disk and trash can icons are displayed.

Print prints the current document to the screen, and Eject tells the machine to eject the disk. Upon first using the Macintosh, many experienced computer operators are quite surprised by the fact that you can't lift out a disk door and remove the disk by hand. Once it's inserted, it's there to stay until you tell the Macintosh to eject it. If for some reason your application should crash (machine locks up), you can force the machine to eject the disk by turning the computer off and on again while holding the mouse button down. This is nice to have just in case.

The EDIT Menu

The EDIT menu works with the Finder just as it does in other applications. The selections are shown in Fig. 4-19. The Finder's EDIT menu allows you to edit text or pictures in desk accessories, text in an information window, and the names of icons. The Undo option will be used to reverse your last text editing action in any desk accessory. The Cut option removes a selection and places it in the Clipboard. You can then select the Show Clipboard option to display the current contents of the clipboard. The Copy option places a copy of the selection in the Clipboard, replacing any of its previous contents. Past puts a copy of the contents of the Clipboard at an insertion point specified by the mouse pointer.

The Clear option removes whatever you previously selected (by clicking the icon) without placing it in the Clipboard. Select All selects all of the icons in the active window just as if you clicked them individually.

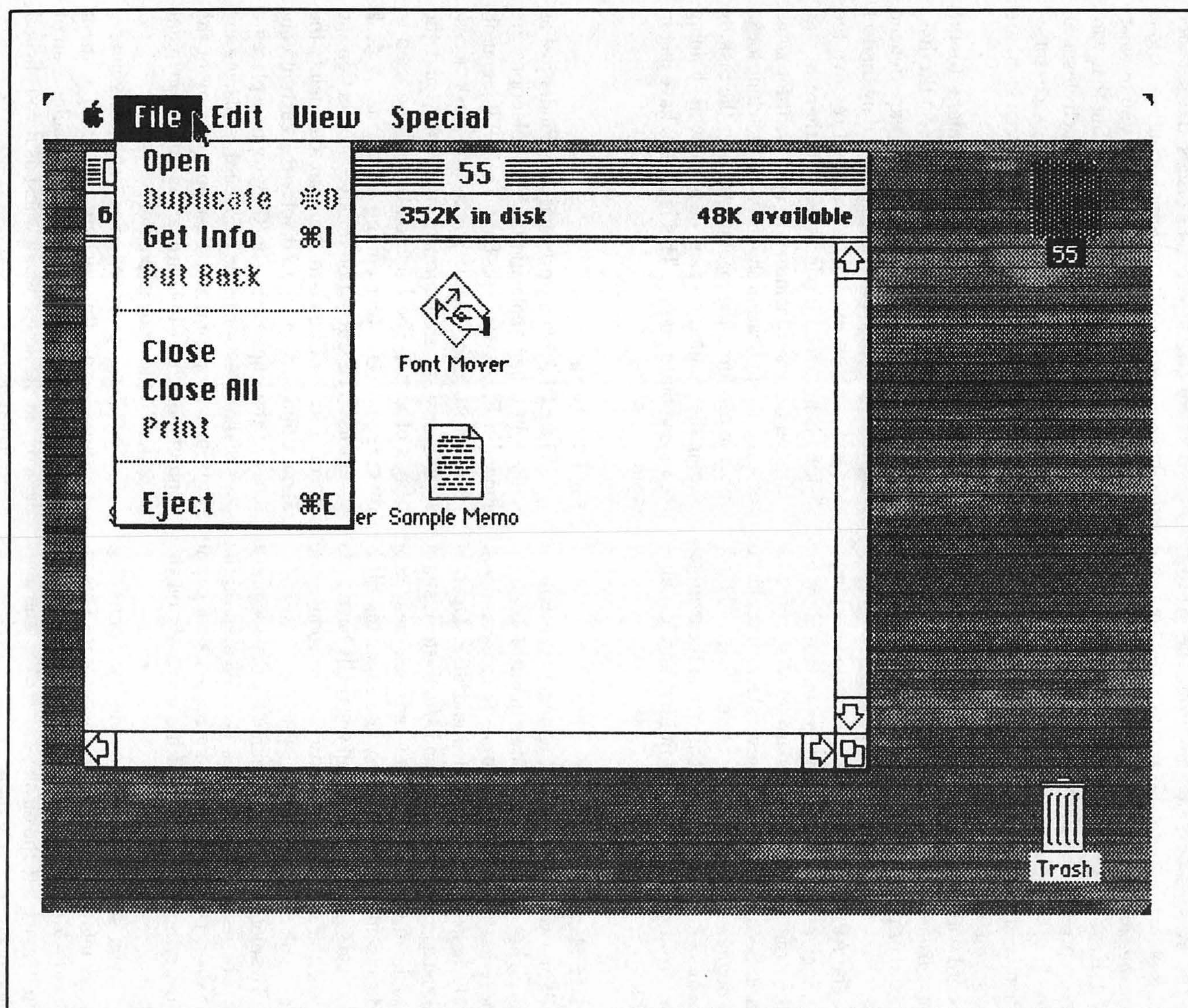


Fig. 4-18. The Finder FILE menu.

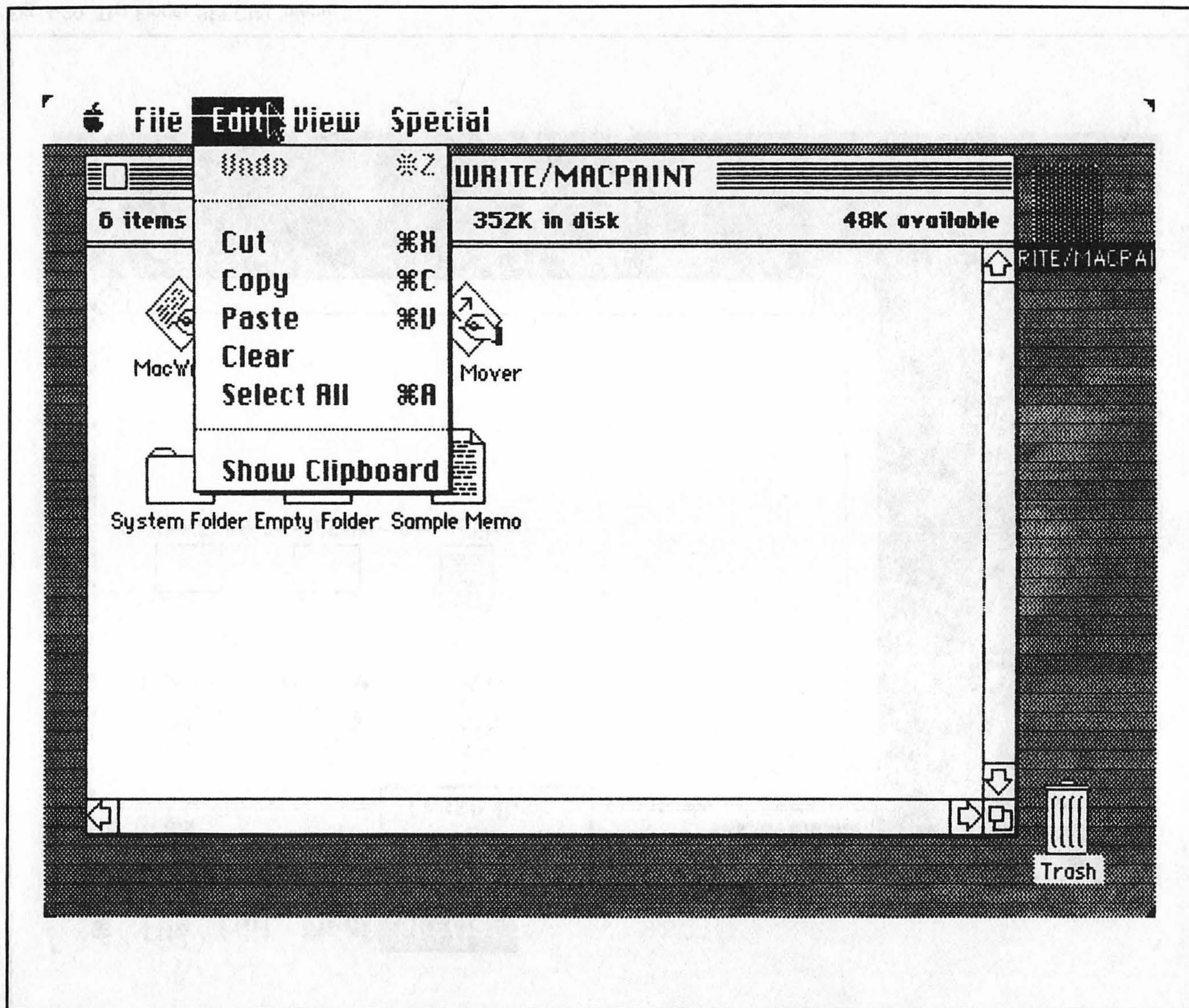


Fig. 4-19. The Finder EDIT menu.

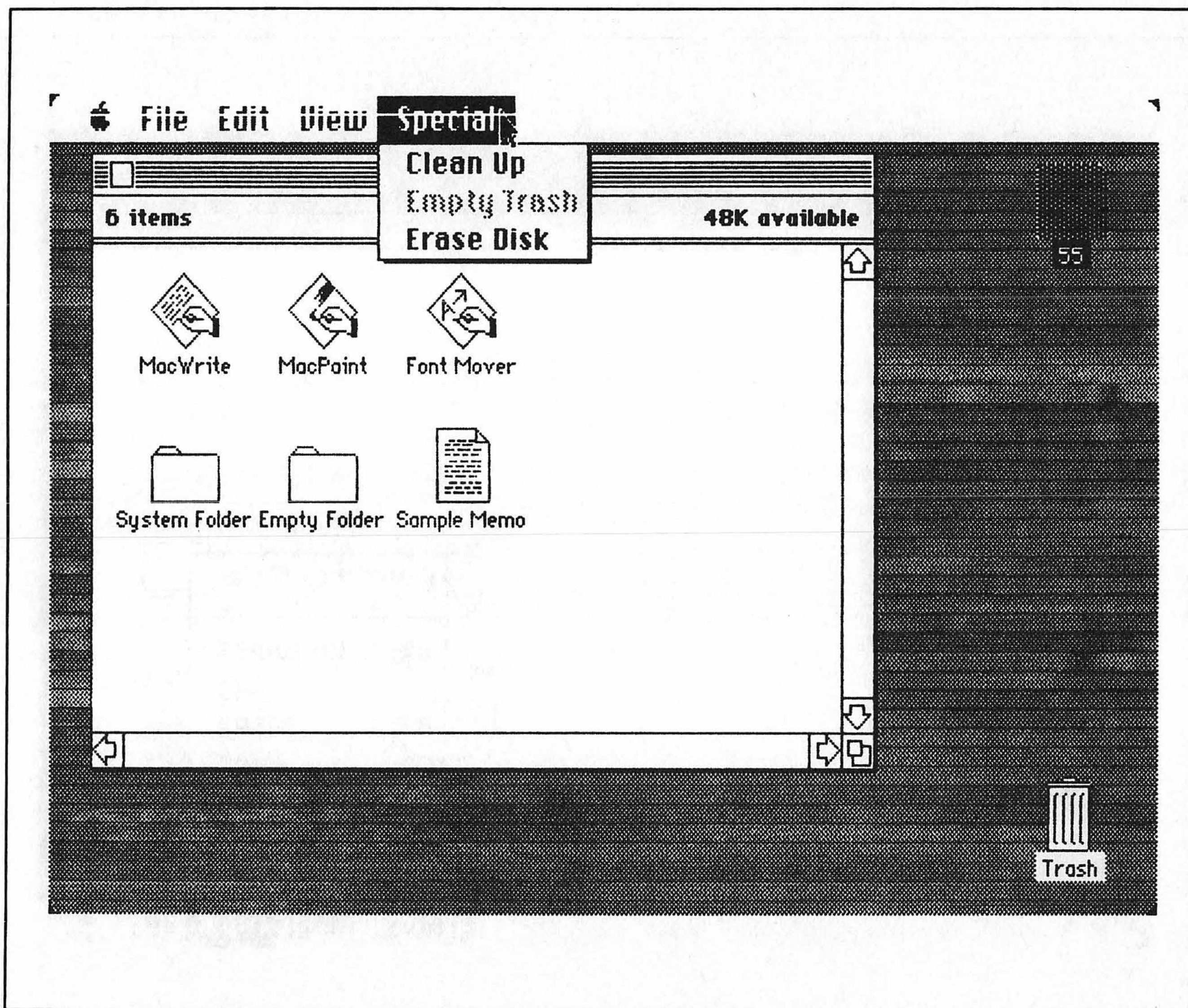


Fig. 4-20. The Finder SPECIAL menu.

The SPECIAL Menu

The SPECIAL menu contains only three options as shown in Fig. 4-20. Stay away from the third one unless you wish to erase your entire disk. When you select Clean Up, all of the icons in the window are arranged in neat rows and columns, like a maid who comes in at the end of your work day and straightens out your desk. The Empty Trash option is used to permanently erase any documents of folders previously thrown in the trash can. The trash can is automatically emptied when a new application is started or when the disk is ejected, but this selection allows you to do it at times other than these.

FILE HANDLING

Let's learn more about the file handling capabilities of the Finder operating system. First, make sure that you're displaying the file contents by icon (VIEW menu). To the right of this window you will see a disk icon with the name of the disk printed below it. The contents in the window are those of the disk that is filled in. If another disk had been loaded first, most of its information would still be in memory. You can access that information by clicking its icon. You will then be told to insert that disk as the current one is ejected. The files on the disk are displayed in various ways. In the examples shown, you may have noticed that some look like folders, and indeed, this is what they're called. Folders are receptacles that allow you to arrange your applications and documents on the disk. When you open a folder, its contents are displayed in directory windows.

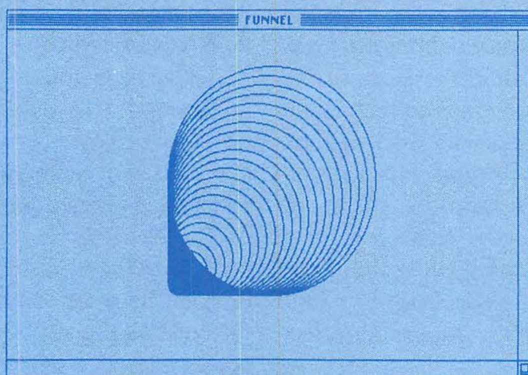
The desk top is the window in which these folders appear. Apple tells you that if you're concentrating on only a few documents and applications at a time, you can move all of them to the desk top and work on them there. You can remove those you don't wish to work on from the desk top. To move an item from the desk top to another point, place the mouse pointer on top and to the left of it and pull to the right and down. The item will be enclosed in a dotted border. When you release the mouse button, the item and its name are in reverse. Place the mouse arrow on the reversed filename and pull to another point on or off the desk top. Again, a border will appear, but when you release the mouse arrow, the document will disappear from its original location and reappear at the new location.

The trash can at the bottom right of the screen is used for discarding documents and folders. You simply drag the document or folder to the trash can. You can still retrieve these documents if you change your mind before you start a new application, eject the disk, or try to add more documents to that disk. You can also go to the SPECIAL menu and instruct the computer to empty the trash. This effectively erases the documents or folders forever.

SUMMARY

The Macintosh Finder takes a little getting used to especially if you are accustomed to manipulating disk files on other computers. With the Macintosh, everything is done with the mouse, and you hardly ever have to use the keyboard. The desk top feature allows you to arrange your files in order of importance and in an arrangement that makes for easy access. Once you become accustomed to using the Finder, it's quite a chore to switch back to the old style of other personal computers.

Chapter 5



Manipulating the Microsoft BASIC Screen

At the time of its official release, there was no high-level language generally available for the Macintosh. However, within a month, Microsoft announced it was developing a BASIC interpreter specifically for the Macintosh, and Apple then announced that MacBASIC would be forthcoming in 1984.

Microsoft BASIC is used in this chapter to show how to manipulate the screen; it can be categorized as the standard BASIC of all American-made microcomputers. IBM, Radio Shack, Hyperion, Compaq, Epson, and other popular microcomputers have been using it for years. While all dialects of BASIC have many similarities, there can be a real problem in switching from one dialect to another and then back again. Microsoft BASIC generally offers more features than most other dialects. Since it is a standard, those of us who are accustomed to programming in Microsoft BASIC had some idea of what the interpreter for Macintosh BASIC would be like. Certainly, there were a few surprises, but the language generally conforms to Microsoft BASICs for other computers.

The first time I used Microsoft BASIC, I was quickly able to sit down and program the computer, just as I would the others I have programmed that use Microsoft BASIC. Some programmers felt that Apple might offer a BASIC interpreter that would closely conform to their own dialect of BASIC used with the Apple II/III series of computers. While there are people who swear by Applesoft BASIC, there are at least an equal number who swear at it. In comparing the two dialects, Applesoft does not offer as much as Microsoft BASIC and is considered by many to be rather cumbersome. This especially applies to the editing functions in Applesoft BASIC, which are practically nil. Microsoft BASIC has traditionally included a large number of powerful editing functions that, reduced to one statement, help avoid having to retype an entire program line because of a simple spelling error. With Microsoft BASIC, you can quickly correct a misspelled word

in a line, insert or delete information, and turn on automatic line numbering. Microsoft BASIC also offers automatic line *re*numbering, which is invaluable. Additionally, BASIC programmers number their lines in increments of 10. The first line is 10, the second 20, and so on. If it's necessary to go back and insert lines between two that already exist, you can insert nine lines (assuming an increment of 10) between each. However, if you fill up all nine places, some major restructuring is necessary if you want to insert all the lines in an existing program in increments of 10, so if you filled up all nine available lines, you would simply RENUM the whole program, automatically numbering all lines in increments of 10.

Rumor has it that MACBASIC is extremely powerful and will require no line numbers whatsoever. Could this be the wave of the future?

This chapter will delve further into the actual uses of the various user functions available in the Macintosh version of Microsoft BASIC. In this chapter, we will discuss editing, structuring, and especially use of the on-screen windows. The windows are rather unique and are not currently available on any other computer that uses Microsoft BASIC.

Microsoft BASIC is brought up on the computer by simply inserting the 3½-inch disk and turning the computer on. You will immediately hear a beep, and shortly thereafter "WELCOME TO MACINTOSH" will appear on the screen. The menu bar will show FILE, EDIT, VIEW, and SPECIAL which we will discuss in detail later.

Also, a disk icon will appear in the upper right corner of the screen, and you will see the Macintosh trashcan in the lower right of the screen. Shortly, another window will appear entitled MS-BASIC (Fig. 5-1). In this window, the various files on the disk will be displayed.

To actually enter the programming mode of Microsoft BASIC, click the block in the upper left corner of the window twice. After a few seconds, you'll be the programming mode.

FOUR WINDOWS

The Microsoft BASIC screen (Fig. 5-2) has four windows that can be thought of as separate areas on the screen devoted to displaying different types of information. The large window on the screen (in the background in Fig. 5-2), is the ***output window***. You cannot interact directly with this window. It is reserved to display what happens when the program is running. The output window is titled with the name of the program that is running if it has been saved. If it hasn't been saved, the word "UNTITLED" appears at the top of the output window, as it does in Fig. 5-2, to indicate that the program in memory has not yet been saved to disk or given a name usually when you are in the process of writing a new program.

The lower window is called the *command window*. This one usually appears at the bottom of the screen. It is titled, appropriately enough, COMMAND. The command window is where the line you are working on will appear. Program lines being edited appear in this window. As soon as you press RETURN, the information contained in the command window is committed to the output window, the command window is cleared, and you may insert another program line. The command window is always active unless a program is being executed or a list window is being displayed on the screen. The command window may be moved to any point on the screen, and its height may grow to twice what it is, when it first appears.

A third window on the BASIC screen is called the *list window*. You can type LIST via the keyboard and press RETURN or access LIST from the menu bar. When you do, the list window appears on top the output window and the entire list of program lines are displayed here. When you run the pro-

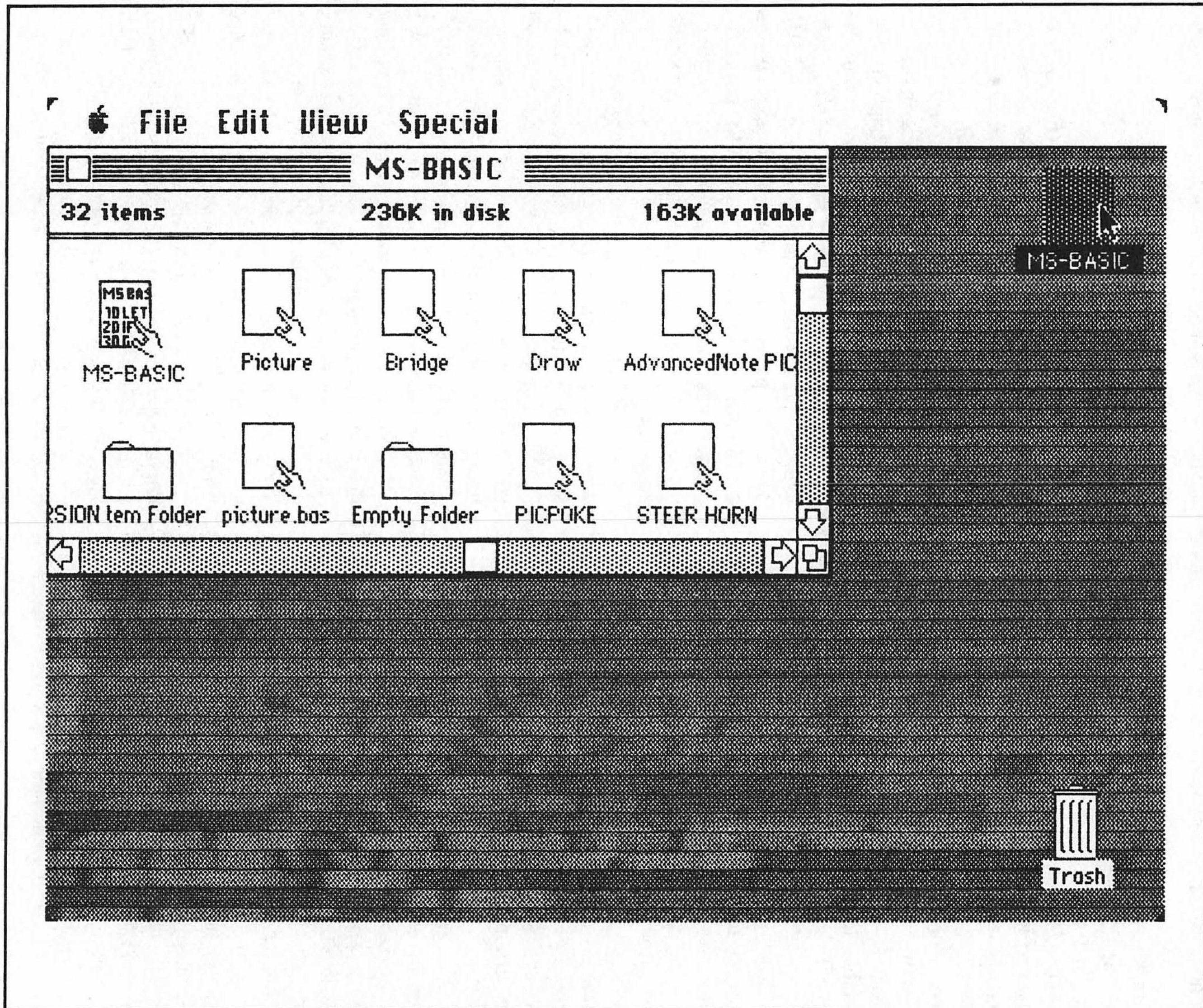


Fig. 5-1. The Macintosh screen upon accessing an MS-BASIC disk.

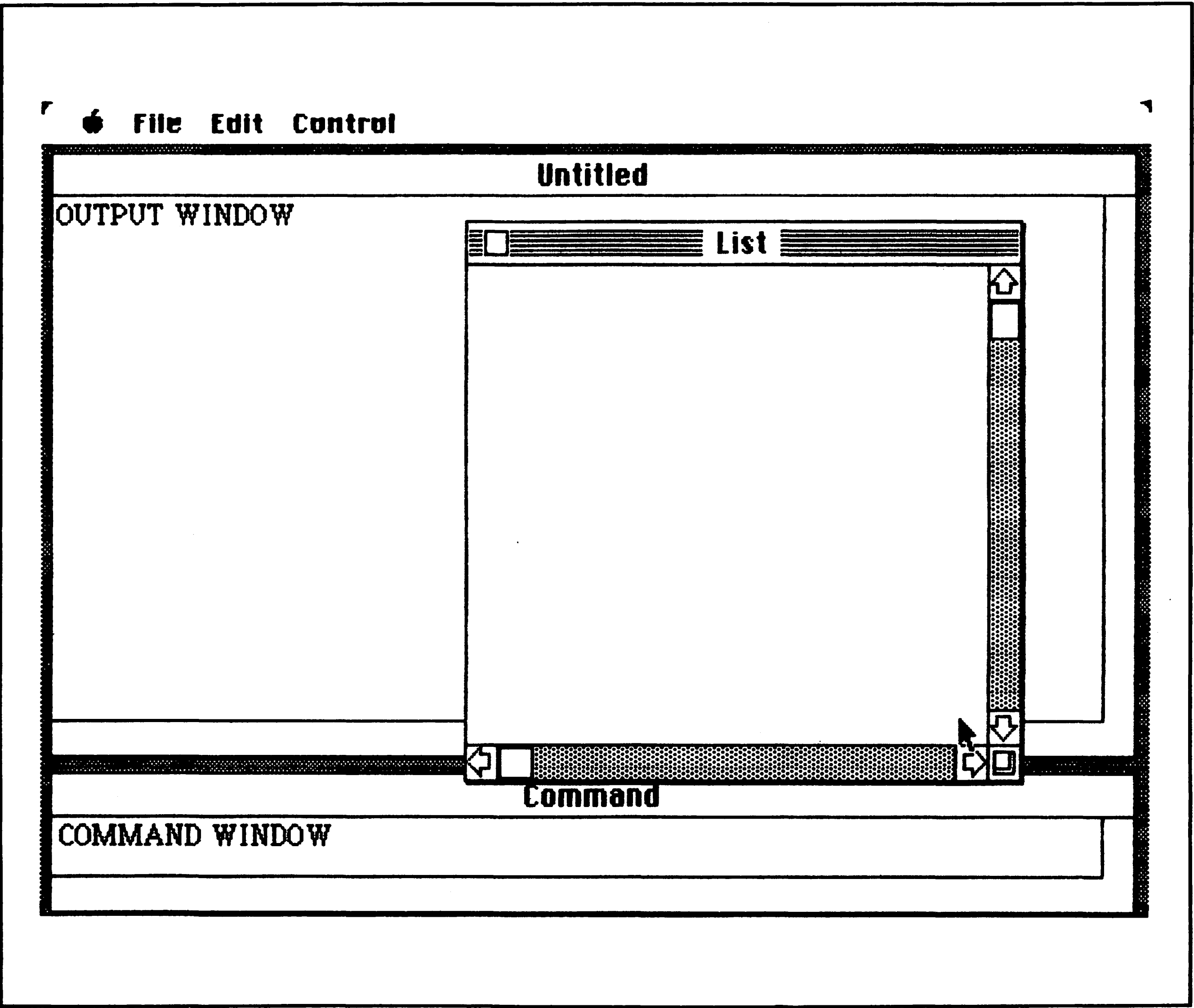


Fig. 5-2. The four windows of the Microsoft BASIC screen.

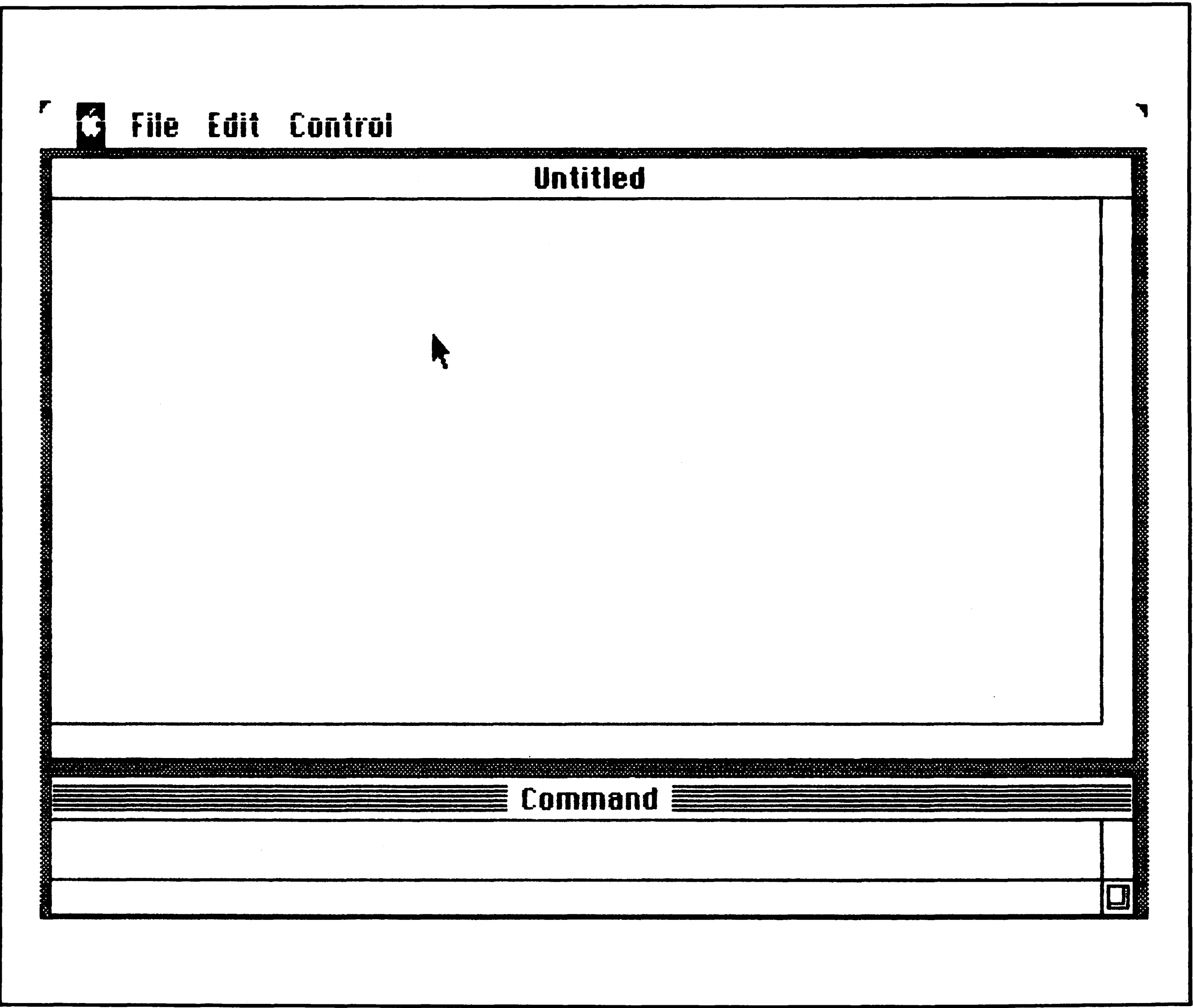


Fig. 5-3. The Output and Command windows of the Microsoft BASIC screen.

gram, the list window automatically disappears. You may have three list windows on the screen at any one time, which is especially nice, since a single list window can display only 12 lines at a time. You can scroll the list window up and down to see other portions of your program, but by moving three list windows to the screen and scrolling each to a different line number, up to 36 lines can be viewed at once. By using the mouse, program lines can be taken directly from the list window and displayed in the command window for editing. The list window can be positioned anywhere on the BASIC screen by moving it to that position using the mouse pointer.

The fourth and final window is the menu bar, at the very top of the screen. The menu bar, as always, still contains for the APPLE menu, the Macintosh desk accessories. The FILE menu contains six commands that address opening and closing files, saving files, and exiting Microsoft BASIC. The EDIT menu contains three commands that are used to write and edit programs. The CONTROL menu has seven commands that control various aspects of program output, including the LIST command.

All four of these screens are immediately available upon entering Microsoft BASIC, although you will not see the list window until the LIST command is typed at the keyboard or accessed from the CONTROL menu.

USING THE WINDOWS

Figure 5-3 shows the Microsoft BASIC screen as it appears when this mode is first entered. At the top of the screen you can still see the menu bar. Below it, the output window (untitled) is displayed. At the bottom of the screen is the command window.

Before typing any program in Microsoft BASIC, it's a good idea to use the WIDTH statement. This statement is used to determine the number of characters that can be displayed on a single line. The maximum is 60 characters per line, but you can also set the computer up to display only 40 characters by typing WIDTH 40. This is typed in direct mode, which means that no line number precedes the command. When you press RETURN, the WIDTH statement is executed immediately.

On most computers, when the end of a line is reached, any additional information is printed below that line, starting at the left margin. However, when you first enter Microsoft BASIC on the

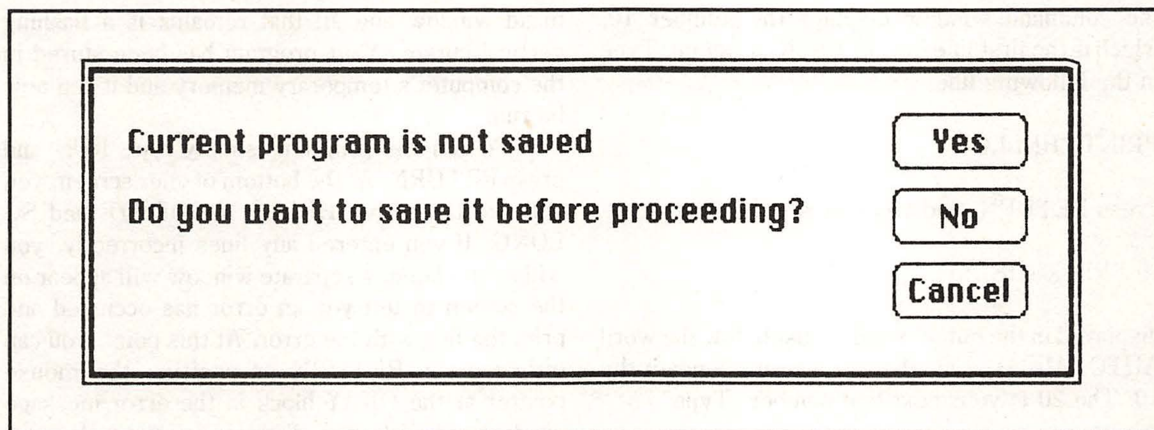


Fig. 5-4. Screen window prompt when exiting a program.

Macintosh, the screen can display (maximum of 60), the rest will never be viewed. If you enter a WIDTH statement with a value of 60 or less, all characters that you type will be displayed in the output window, even if a line exceeds 60 characters. Those beyond the maximum screen width will appear below the first line.

For now, clear the screen by typing CLS and pressing RETURN. Also type NEW and press RETURN. This will remove any existing programs that you may have typed in earlier. If there is a program currently residing in memory, the screen will prompt you as to whether or not the program is to be saved to disk (Fig. 5-4). If you want to save the program, position the mouse pointer on the YES block and click once. If you do not wish to save the program, select NO and click. If you want to cancel the NEW command to, perhaps, go back and work with the program, then click CANCEL. Assuming you did not select CANCEL, the NEW command has erased the residing program from current memory and you are presented with a blank screen.

Let's enter a sample program for test purposes. Microsoft BASIC offers an automatic line numbering feature that is accessed by typing AUTO and pressing RETURN before you begin writing a program. Do that now. When you press RETURN, the command window displays the number 10, which is the first line in your BASIC program. Type in the following line:

PRINT "HELLO"

Press RETURN, and you will see the line

10 PRINT "HELLO"

displayed in the output window just below the word AUTO. Also, the number 20 appears beneath the 10. The 20 is your next line number. Type

PRINT "GOODBYE"

and press RETURN. This new line appears in the output window, and the number 30 appears below it. Now, type REM and press RETURN. Repeat this at least 15 times. In other words, type REM and press RETURN at least 15 times. In BASIC, the REM statement stands for remark. It allows programmers to insert comments after it that explain the program. While these lines are part of a BASIC program, they are not executed by the computer, because they are there simply to aid anyone who might be viewing a program listing. A new number still appears in the command window, so type

PRINT "SO LONG"

and press RETURN. Again, a new line number appears in the command window, but we are finished with this program, so we need a way of telling the computer to stop giving us new line numbers automatically. To do this, we manually "break out" of the current routine. Immediately to the left of the space bar on the keyboard is a key that contains a flower-like design. To exit the automatic line numbering routine or to stop any program while it is executing, hold down this key and press the C key. Now, the number disappears from the command window and all that remains is a flashing vertical cursor. Your program has been stored in the computer's temporary memory and it can now be run.

To run the program, simply type RUN and press RETURN. At the bottom of your screen, you will see the words HELLO, GOODBYE and SO LONG. If you entered any lines incorrectly, you will hear a beep, a separate window will appear on the screen to tell you an error has occurred and print the line with the error. At this point, you can either press RETURN or position the mouse pointer at the OKAY block in the error message window and click once. Either way, the faulty program will appear in the command window.

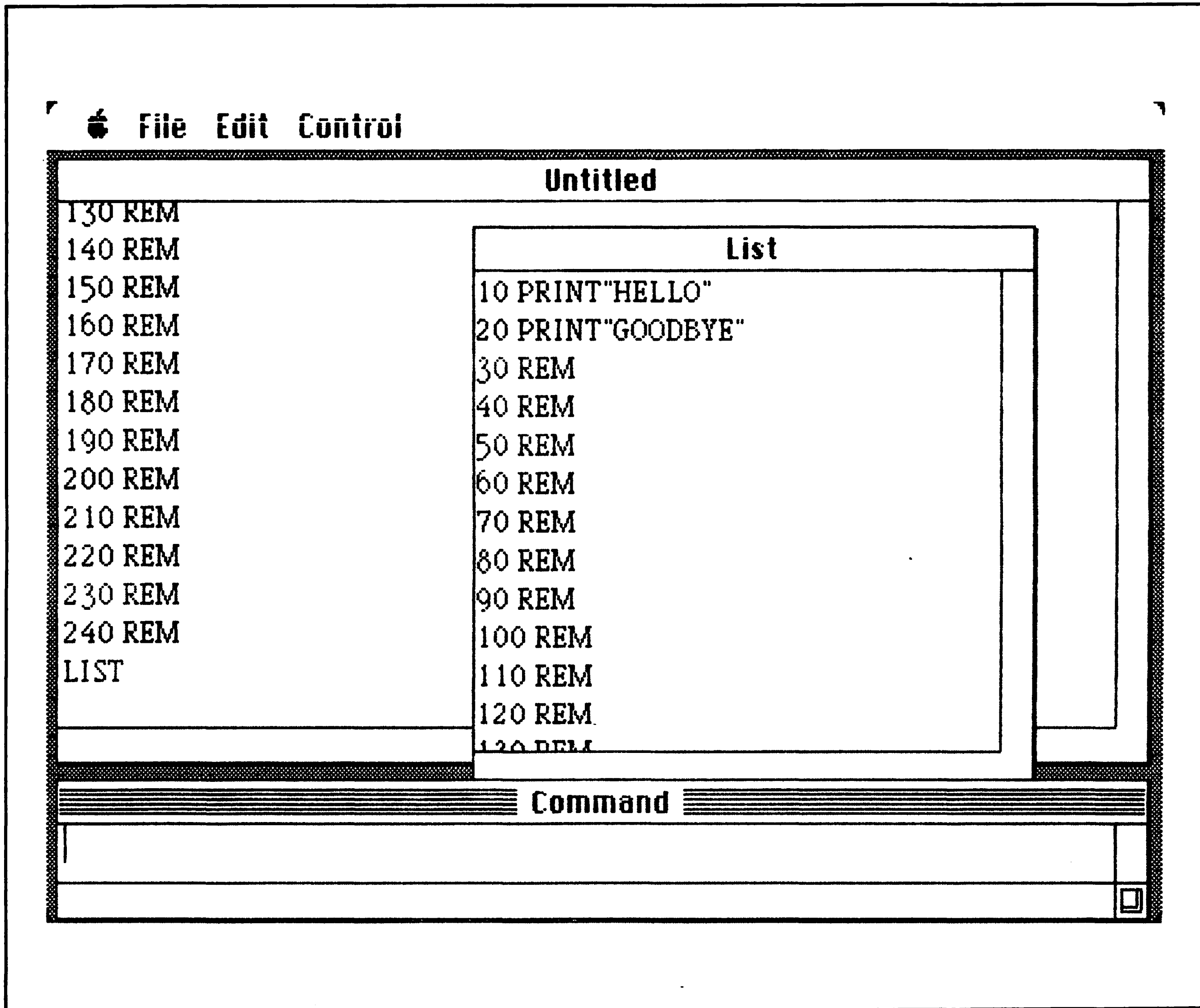


Fig. 5-5. The List window.

The program we are working with is very simple, so there should be no errors. However, if there is an error, simply use your backspace key to wipe out the line and reenter the line correctly. There's a much easier way to edit on the Macintosh, but we will discuss editing later.

For now, let's assume that your program has run properly. At this point, we wish to list the program lines contained in the program. To do this, type LIST and press RETURN. A new window appears to the right (Fig. 5-5), entitled LIST. In the window you will see a portion of the program originally input, but you won't see all of it. I asked earlier that you input at least 15 REM statements, to assure that your program would be too long to be displayed in its entirety in the standard list window. Assume at this point that we wish to look at the last line in the program, which is not presently displayed.

Place the mouse pointer in the right margin of the list window and click once. When this is done, all of the margins change. The one on the right and the one at the bottom now contain arrows to use for *scrolling*. Scrolling is the process of rotating the information in the list window up or down. Now, still working in the right margin, move the mouse pointer to the bottom arrow (Fig. 5-6). Press the mouse button and hold it down until the scrolling process stops. At the top of the list window, you will now see the last line of your program. Now, place the mouse pointer on the top arrow in the right margin of the list window and press the button again. Your program now scrolls downward and you can stop it any time by releasing the button. When you press either arrow, a white box travels up and down the right margin to indicate how much of your program has been displayed. Now, place the mouse arrow at any blank area in the list window and click once. The borders revert to their unfilled condition, and we are back to where we started.

You can remove the list window from the screen when you've finished viewing it by pressing

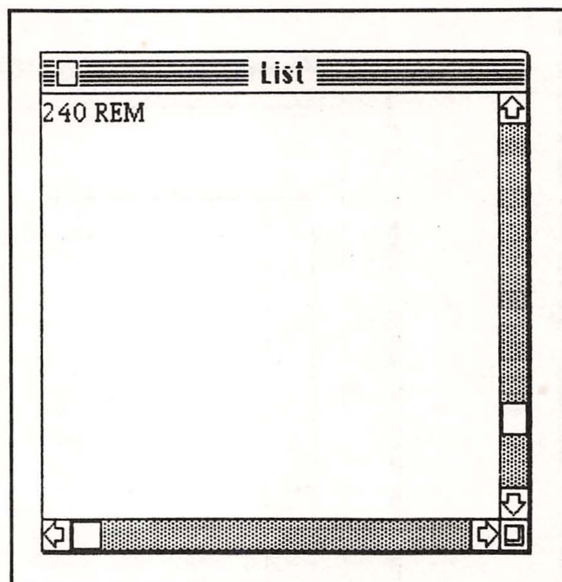


Fig. 5-6. Scrolling the List window upward.

RETURN or by placing the mouse pointer at any point in the output window and clicking. Use any one of these methods to clear the list window from the screen. Now, the list window seems to disappear from the screen, but if you look at the area between the top of the command window and the bottom of the output window, you will see a white space to the right (Fig. 5-7). This is your list window! It has been placed *behind* the output and command windows. Place your mouse pointer on this tiny window segment and click once. Eureka! The list window is back and in the scrolling mode. Remove it again by pressing RETURN. This time you will have to press RETURN twice. The first press exits the scrolling mode and the second places the list window behind the other two windows. Now type

```
10 PRINT "THIS IS A TEST OF THE HORI-
   ZONTAL SCROLLING FUNCTION"
```

and press RETURN. Move your mouse pointer to

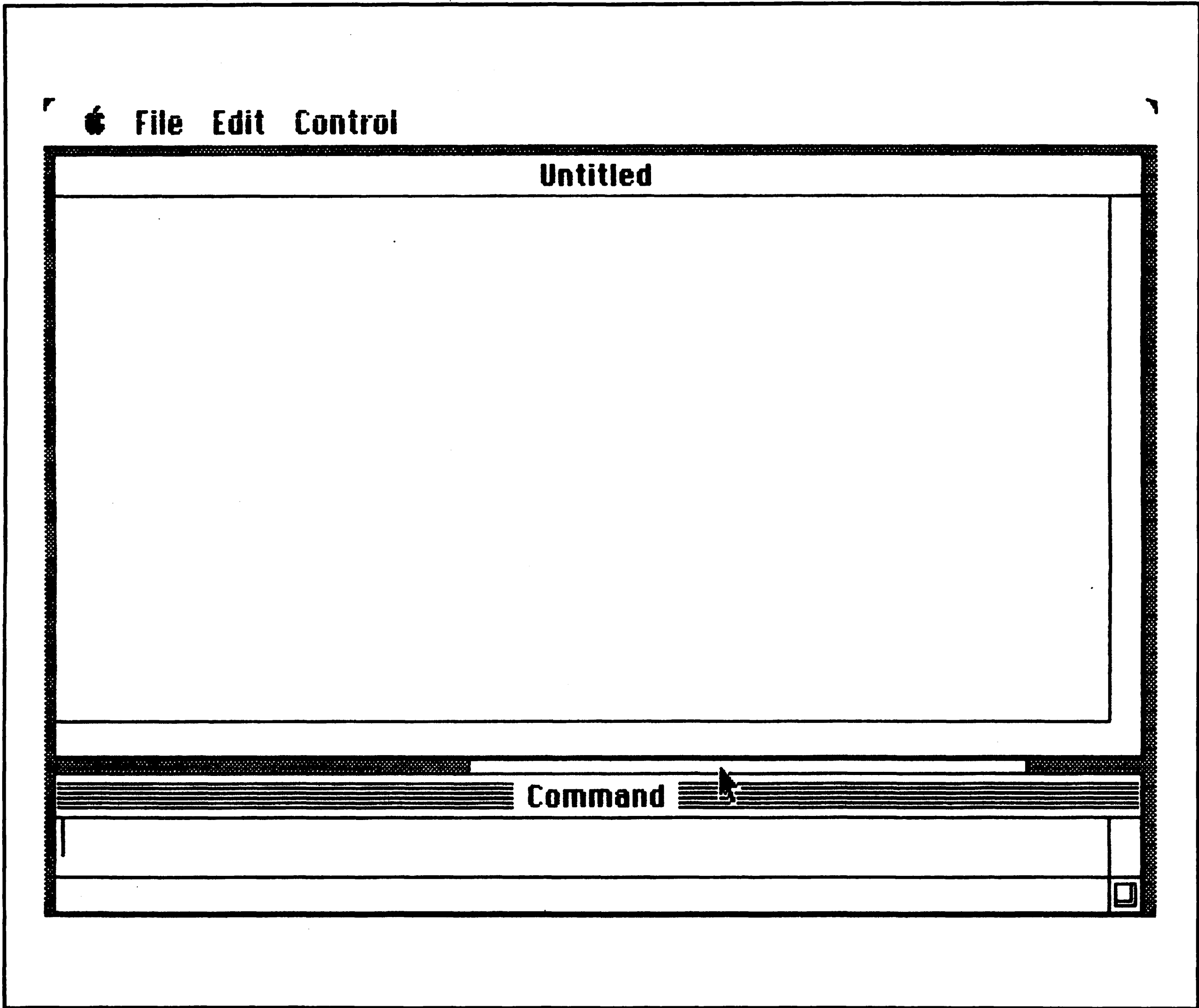


Fig. 5-7. The arrow points to the List window that has been placed behind the Output and Command windows.

the visible portion of the list window between the command and output windows and click once. Again, the list window appears with the original line 10 altered. It is now much longer than it was before. In fact, it is so long that the entire line cannot be seen in the list window (Fig. 5-8). However, there is also a left to right scrolling feature in the bottom margin. To see the rest of line 10, place your mouse pointer on the righthand arrow in the bottom margin. Press the mouse button or click it several times, and you will see the rest of line 10 appear (Fig. 5-9). If you keep pressing the button, line 10 will scroll completely off the screen. To get back to where you began, place the mouse pointer on the lefthand arrow in the bottom margin and hold the mouse button until you can again see the line numbers. This demonstrates that the list window allows us to scroll program lines up, down, left, or right.

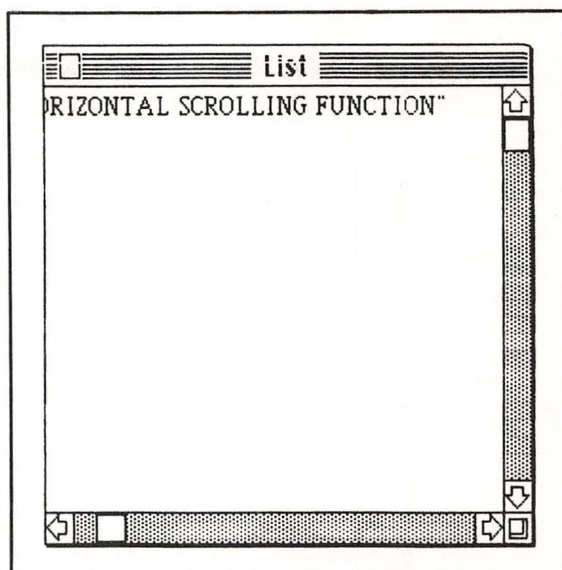


Fig. 5-9. The List window scrolled to the left.

EDITING IN BASIC

It's time now to learn a bit about the editing functions available in Microsoft BASIC. Our assignment is to change line 10 to:

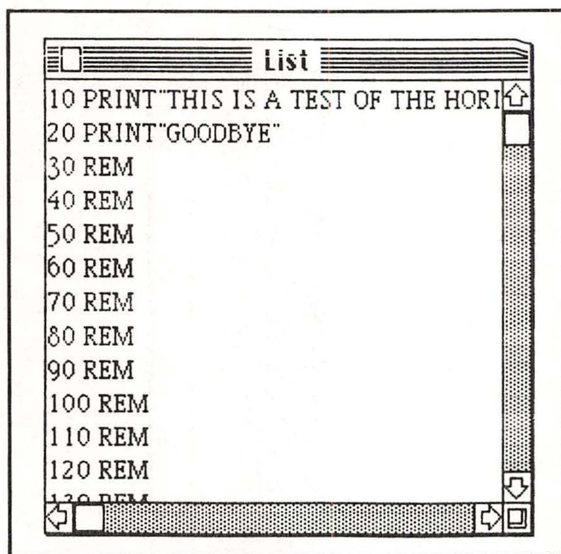


Fig. 5-8. Example of a program line (10) that exceeds the horizontal width of the List window.

```
10 PRINT "TESTING EDITING FUNCTIONS"
```

We can do this by simply typing in this new line. As long as we use 10 as the line number, the original line 10 will be replaced with the new one. However, often, we want to change part of a line rather than replacing it completely. In Microsoft BASIC, the mouse is used to edit. Place the mouse pointer at any point to the right of line 10 in the list window. For example, you might place it on the P in PRINT. Do this and click the mouse once. The list window must be in scroll mode to do this. If it is not, click *twice*. At this point, you should see all of line 10 appear in the command window at the bottom of the screen (Fig. 5-10). At the end of the line you will see the flashing cursor. Now, press the backspace key at the top right of the keyboard and hold it down. You will see the flashing cursor move to the left, obliterating those portions of the line it passes

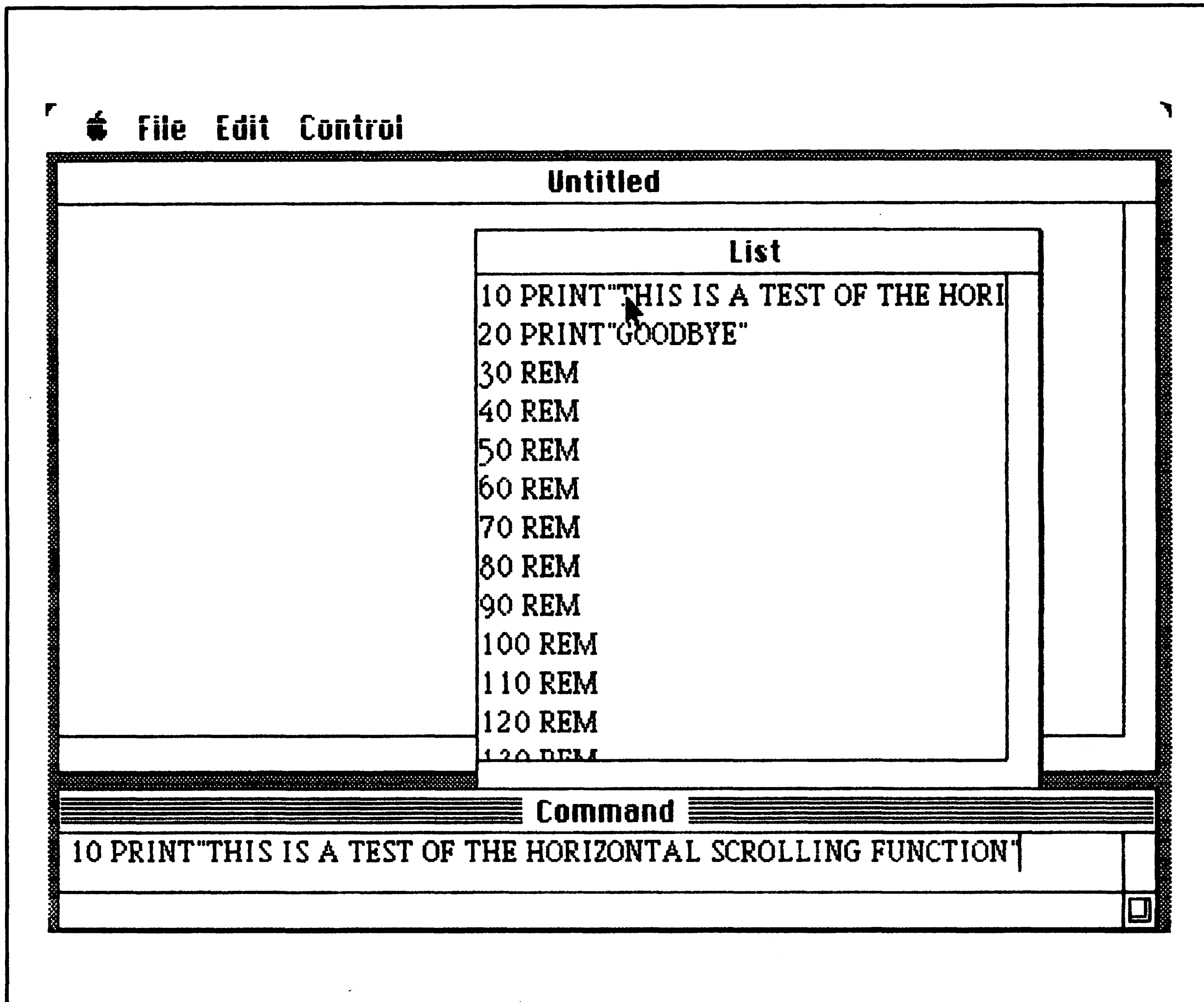


Fig. 5-10. By placing the mouse arrow on a line in the List window and clicking, the line is displayed in the Command window and is ready for Editing.

over. When the cursor nears the PRINT statement, release the backspace key and then press it, releasing it quickly, a number of times until you have erased everything after the first quotation mark. Now, type

TESTING EDITING FUNCTIONS"

and your line should now read:

10 PRINT "TESTING EDITING FUNCTIONS"

Press RETURN. Line 10 in your list window has now been changed to reflect your editing (Fig. 5-11).

To further exercise editing, let's change line 10 to read:

10 PRINT "TEST EDITING FUNCTIONS"

To do this, click line 10 again (twice) to move it to the command window. Wouldn't it be nice if instead of obliterating the entire line back to the second T in TESTING and retyping the remaining portion, we could simply delete the ING from TESTING and leave the rest of the text alone? Fortunately, we can. Using your mouse pointer again, place it at the space between the words TESTING and EDITING. Click once. You should see your flashing vertical cursor leave its position at the end of the line and take up residency at the location of your mouse pointer. Use the backspace key to erase ING from TESTING by pressing it quickly releasing it three times. When you do this, each time you press the backspace key the letter preceding the cursor is erased and all information after the cursor is moved one character to the left. By clicking three times, you erased the ING and your line now reads:

10 PRINT "TEST EDITING FUNCTIONS"

Press RETURN, and line 10 in the list window now shows the edited line.

For another test, assume that line 10 is to be changed to:

10 PRINT "TEST THE EDITING FUNCTIONS"

In the previous example, we erased a portion of the program line. In this one, we wish to insert the word THE between TEST and EDITING. Place the mouse pointer on line 10 and click twice. The line now appears in the command window as before, and the flashing cursor is at the end of the line. Place the mouse pointer between the words TEST and EDITING and click once. The cursor should now appear at this point. Hit the space bar once and a space is inserted after the T. Type THE. Your line now reads the way we want it to.

Note that as each letter is inserted, the text moves to the right. Some of you may have had a different result than the one described here. There are two places between the end of TEST and the beginning of EDITING which the flashing cursor

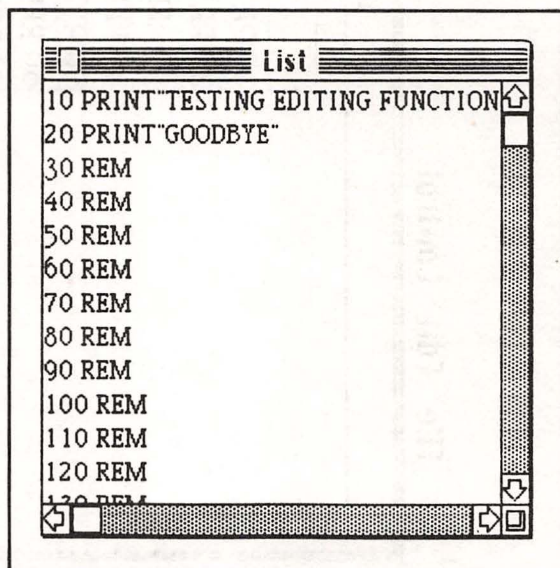


Fig. 5-11. Line 10 after it has been edited.

can assume. If you placed your arrow exactly in the center of the space between these words, the flashing cursor was positioned immediately after the T in TEST. However, if you were a little to the right of center, the cursor was placed immediately before the E in EDITING. When you typed the space according to my instructions followed by the word THE, you ended up with:

```
10 PRINT "TEST  THEEDITING FUNC-
   TIONS"
```

This is easy to correct. Add another space and place your cursor just to the left of the T in THE. Click again and hit backspace once. Your line should now read correctly. Press RETURN to enter this edited line in the list window.

MULTIPLE LIST WINDOWS

As stated earlier, we can display more than one list window on the screen. Remove your current list window by pressing RETURN. Again, you can see it peeking out between the output window and the command window. Type LIST again and press RETURN. The list window again appears on the screen, or at least would seem to, but this is not the same list window. To the left or right of the list window currently displayed (probably the left) you will still see a portion of another window peeking through, as indicated by the arrow in Fig. 5-12. This is your first list window, whereas the one fully displayed is the second list window. Each time you type LIST (up to three times), a new list window is generated. Place your mouse pointer on the portion of the window peeking through the click once. It will then be displayed on the screen, partially overlapping the list window already displayed. To view them both simultaneously, move your mouse arrow to the top margin, press the mouse button and hold it. Move your mouse toward the left and you will see the outline of the window move toward the left. Keep moving left until the left edge of the ghost window aligns with the left margin of the output window. Also align the top edge of the ghost window with the top of the output window. When you get the ghost window positioned, release the mouse button. You can see that your visible list window now occupies the space outlined by the ghost window. Most of the second list window can also be seen, but if you can't see all of it, place the mouse pointer in the top margin of the second window and click once. Now the screen margins will fill in. Press the mouse button again and hold it. Move your mouse toward the right of the screen. Again, you will see a ghost window. Align this one with the top right of the output window. When you release the mouse button, both windows will be displayed on the screen (Fig. 5-13). You may have to do a bit of shuffling, since the space is very tight, but you should be able to reach a point at which both windows are completely visible. Note that both windows contain the same information. The program listings begin with line 10 and proceed until they run off the window. However, we can use the scrolling function in one window to display those program lines that cannot be seen in the other.

If your right window is not presently in the scrolling mode (scrolling arrows visible), click one of the margins to access this mode. Place the mouse pointer on the bottom arrow in the righthand margin and click it until the program lines listed in the right window take over where the left window stopped (Fig. 5-14). In my left window, I can see lines 10 through 120 completely. Therefore, I have scrolled the righthand window until line 130 appears at the top. This allows me to see the entire program on the Macintosh screen at one time.

We can also access a third window, and by carefully positioning it on the screen between the other two windows, show a total of three windows. This is a little cumbersome, however, since our first two lists windows could be hiding almost anywhere. Sometimes, they're completely behind the

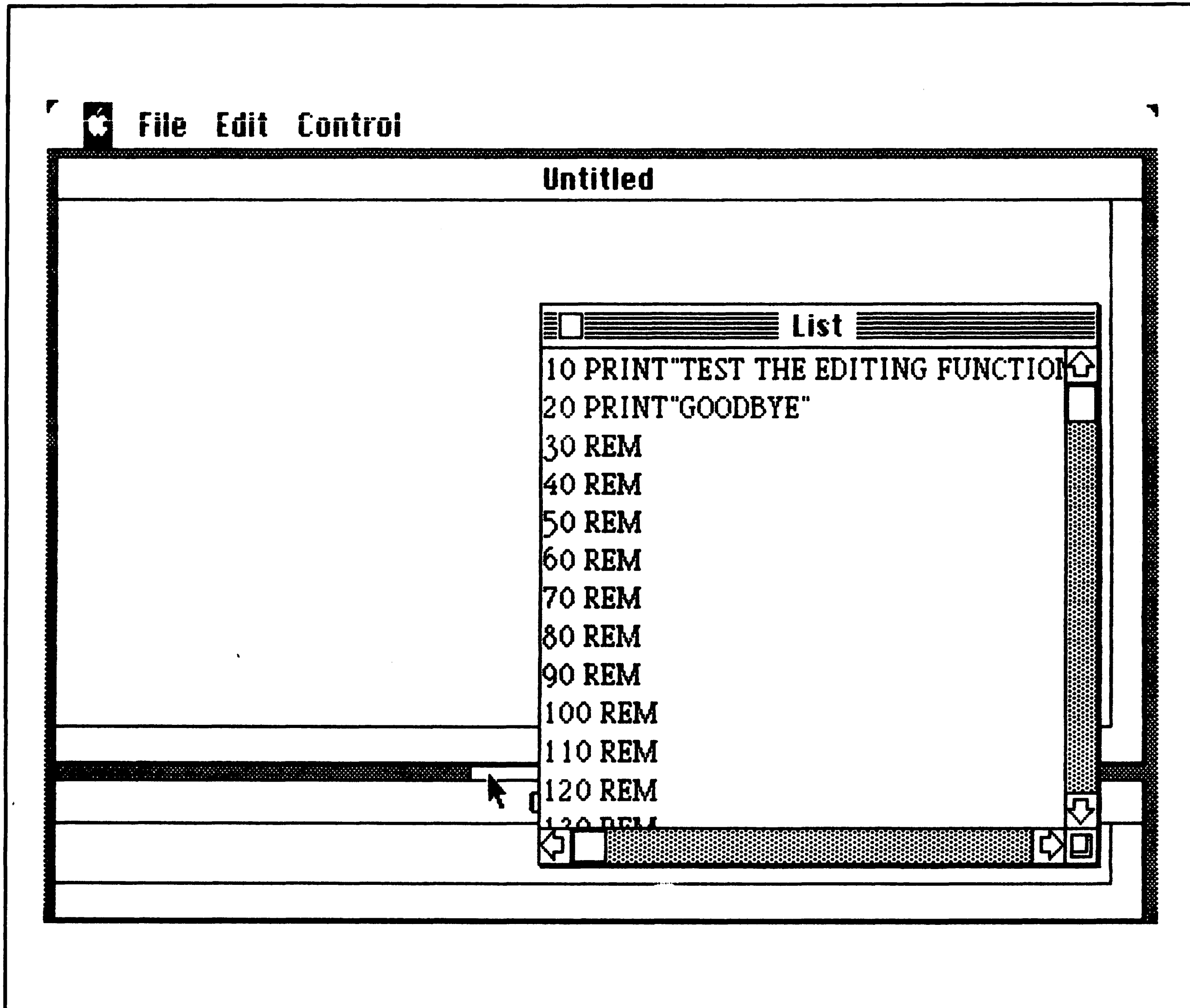


Fig. 5-12. One List window is seen in the foreground; The arrow points to the other, located behind the Command and Output windows.

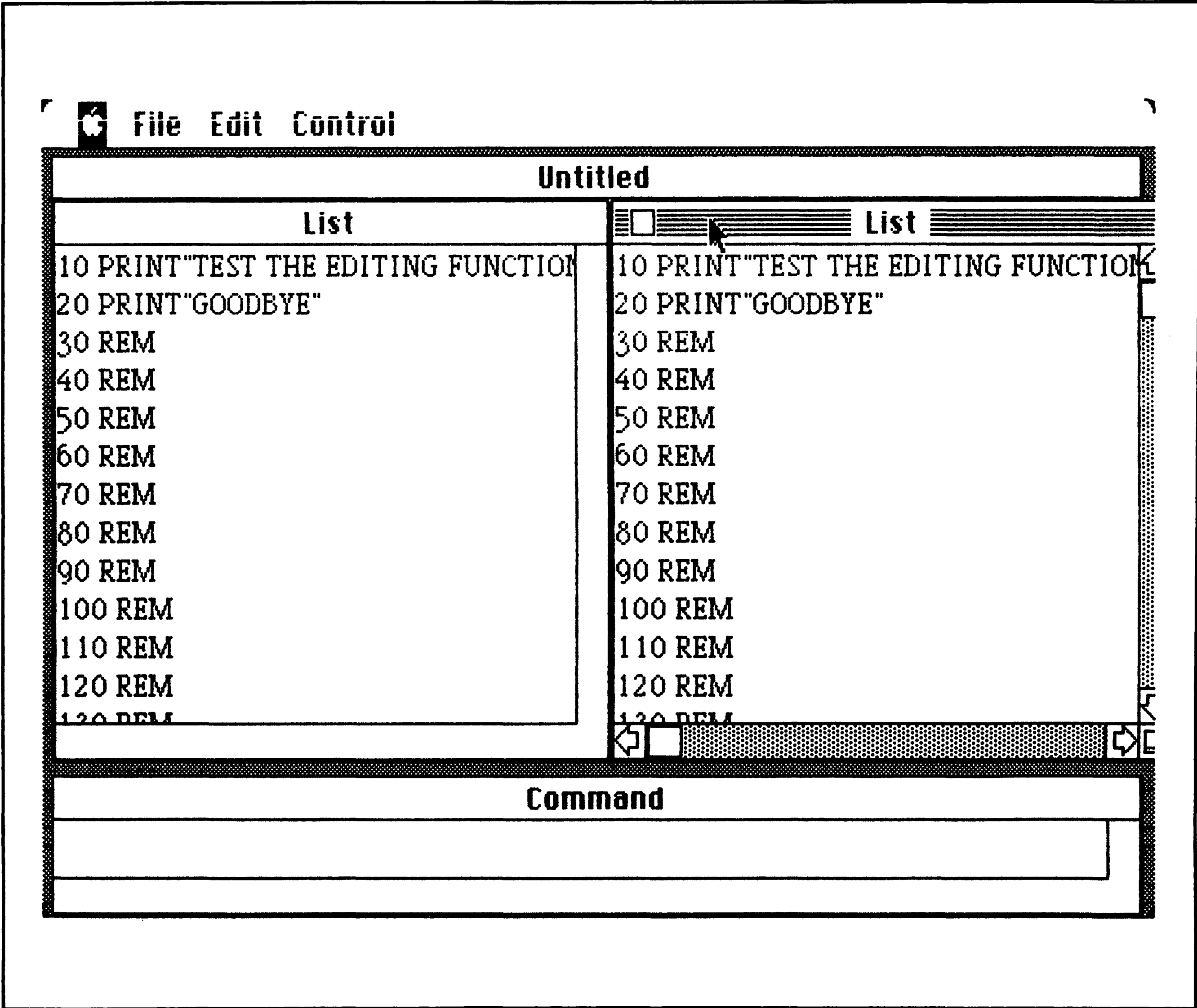


Fig. 5-13. Two List windows displayed on the Macintosh screen.

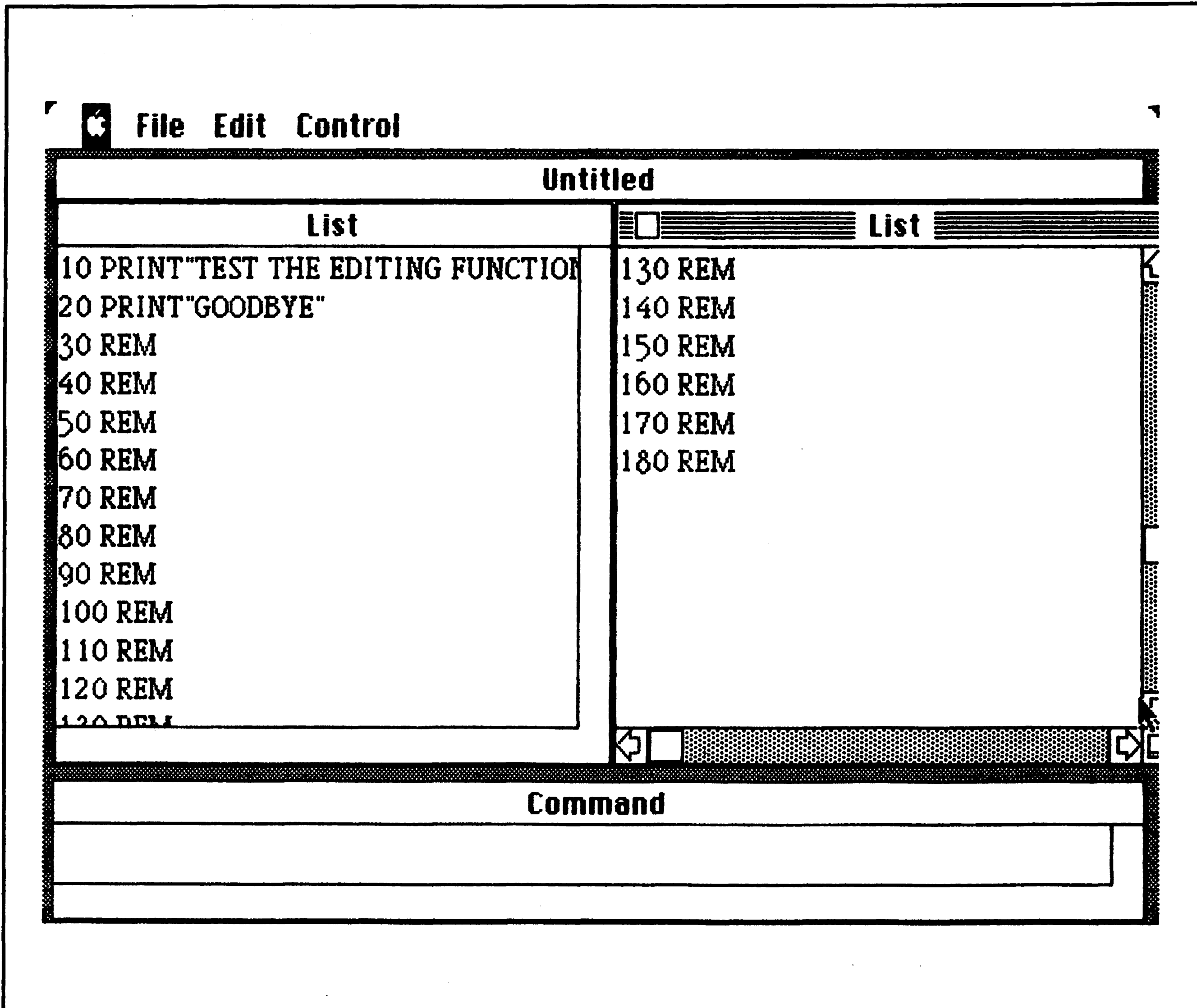


Fig. 5-14. The List window on the left displays lines 10-120. The one on the right has been scrolled upward to display the remaining lines.

output window and can't be seen at all. The best way to get a three-window display is to drag both windows back to their original positions at the right of the screen. Place one over the other and type LIST again. This will bring up the third window. Move it to the left top of the screen and click another one. Drag the second one to the right top of the screen. You should now see one more window peeking out between the bottom of the righthand list window and the command window. Click it and drag it to near the center of the screen. You will have to shuffle all the windows to have all three visible (Fig. 5-15). This will probably mean dragging the right one partially off the screen.

If each program line is not too long, you can conceivably gain some advantage by using a third window, but you probably won't need it. However, it's not necessary to display all three at once. Assume you have a program that fills more than two windows. First, create two list windows using the LIST command. Position them on the screen as already described, and scroll them so that the second picks up where the first left off. Create a third list window by typing LIST and scroll it so that it picks up where the second left off. Then click the third window off the screen. You can now view the first two windows, and if you need to view the third, simply click it back onto the screen. This is a much more practical way to use the three-window capability. For now, erase all three windows. We are now back to our standard screen with a menu bar, output window, and command window.

MOVING OTHER WINDOWS

Just as we can move list windows to different locations on the Macintosh screen, we can do the same with the output window and the command window. The menu bar, however, is fixed and cannot be moved. Our next assignment is to move the output window to another location on the screen. Do this by placing the mouse pointer at any location in the top margin of the output window (where the title is displayed). Press the mouse button and hold it. Move your mouse and you will see the ghost image of the output window travel in the same direction. You can (almost) pull it off the screen, put it down over the command window, or place it anywhere you want.

To move the command window, simply click *its* top margin once and then click again and hold. The command window can be moved to the top of the screen so it covers the output window title. You can also move the command window to the top of the screen (beneath the menu bar) and drag the output window toward the bottom. This reverses the normal screen format. You can call either window to the forefront by simply clicking it.

Sometime you could end up in an arrangement whereby one window is completely covered by another, and you cannot find a portion to click. If this happens, move the top window partially off the one beneath it to gain access to the covered window. Sometimes, manipulating the Macintosh screen in Microsoft BASIC is like playing a game of hide and seek. After a few hours of practice, you should be quite comfortable with these features.

All of these windows (list, output, and command) contain another movement feature that is quite useful. Perhaps you've noticed by now that in the lower right corner of the output window, there is a small box. Maybe you've even tried clicking it with the mouse to see what happens. I call this the size box. It allows you to increase or decrease the height of the output window.

To test this feature, reposition your windows so that the output window is at the top of the screen and the command window is at the bottom. Click the output window into *drag* mode. Drag mode is in effect whenever the horizontal lines are in the top margin (title line) of the output window. When the drag mode is in effect, you can see the drag box to the bottom right. Place the pointer in the drag box and press the mouse button. Hold it down and pull the arrow toward the bottom of the screen. Pull it

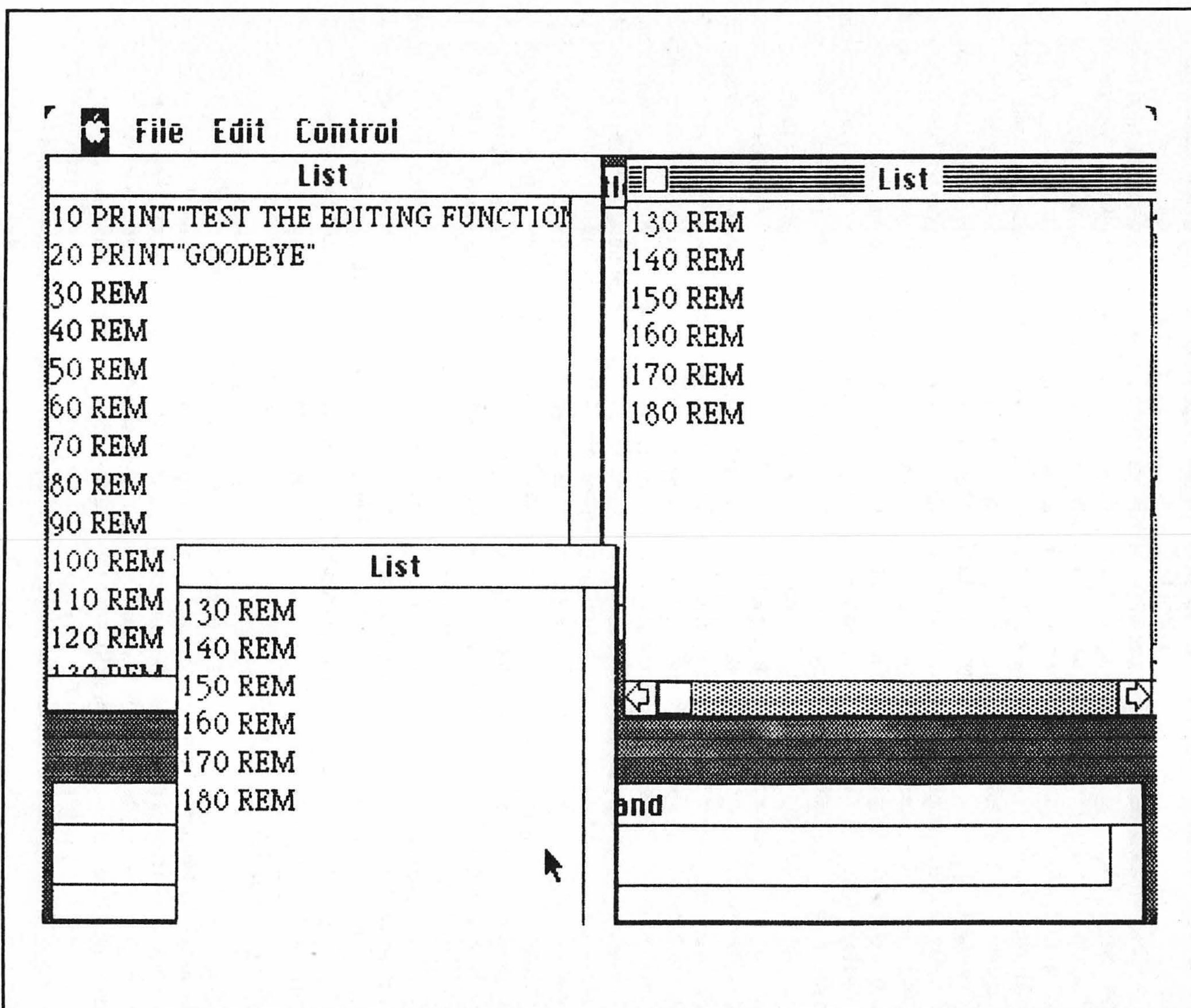


Fig. 5-15. A maximum of three List windows may be displayed at any one time.

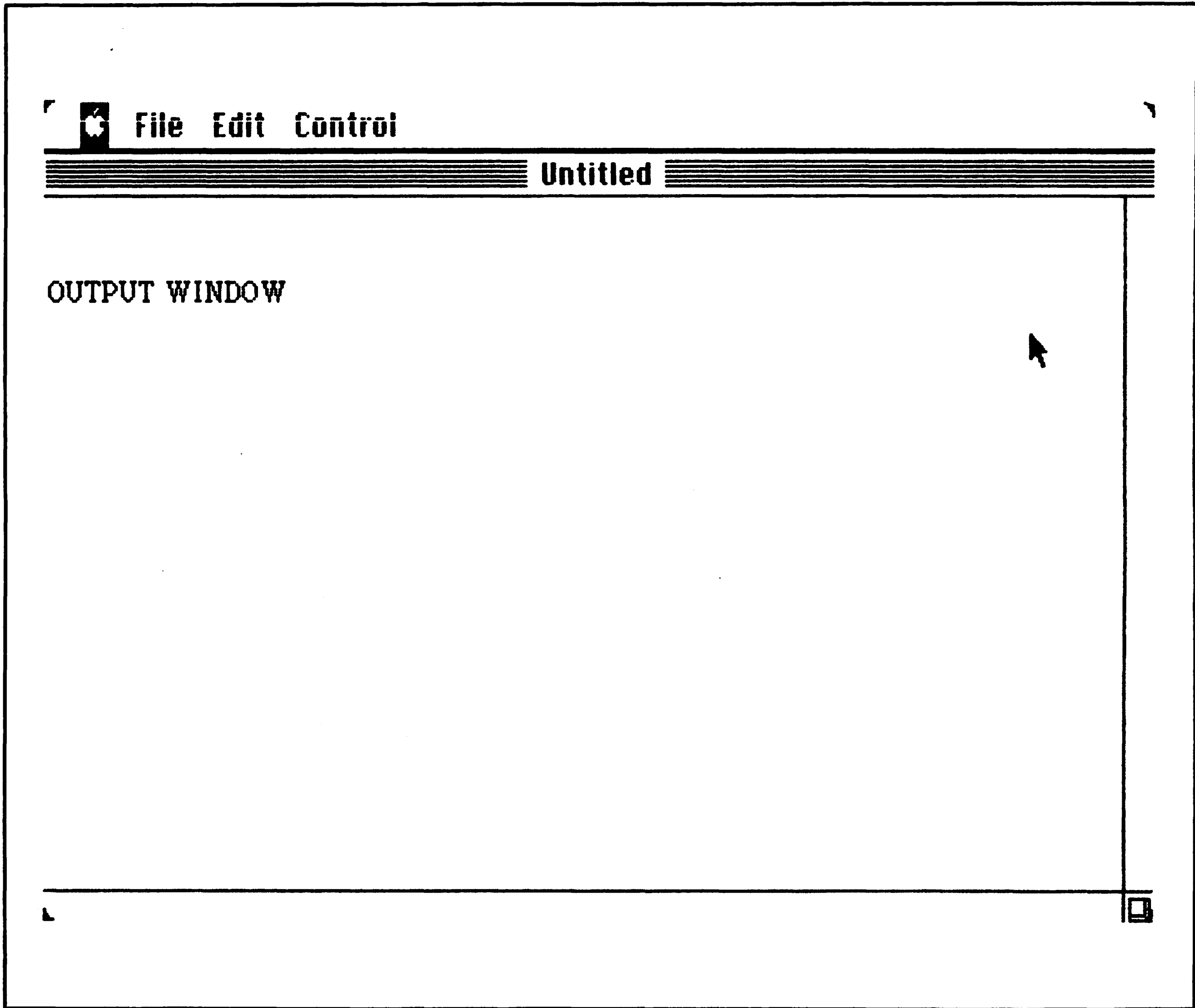


Fig. 5-16. The enlarged Output window is created by moving the Size Box at the lower right of the screen.

straight down. If you move left, the entire output window will be positioned to the left. When the drag box of the ghost image is at the bottom right of the screen, release the mouse button. Eureka! The entire window fills the Macintosh screen (Fig. 5-16). Don't worry about your command window at this point. All you have to do to access it is press RETURN. Your command window is brought to the forefront, but now, anytime you run a program, it will be run on the entire screen because the command window will disappear when the program is run. However, this will play havoc with multiple window listings, since you won't be able to see the windows at all when they lie behind the output window.

What I generally do is write a program with the screen configured as it normally comes up when you first enter Microsoft BASIC. After I've handled all debugging with the aid of the list windows and am sure the program is running correctly, I move the output window to the bottom of the screen for full screen viewing of the program run. When I write another program, I simply move the top of the output window toward the top of the screen again. If you have difficulty seeing the list windows, decrease the height of the output screen even more.

We can also increase the height of the command window. It will first be necessary to move the command window so that it begins about three-quarters of the way down the screen. Place your mouse pointer in the size box at the lower right of the command window and drag it down to the bottom. The command window is now twice its normal size (Fig. 5-17). When initially configured, it is the smallest size possible. Using these methods, you can set up a screen display just about any way you want it. You can adjust the window displays for personal taste as well.

Reposition the output and command windows to their original formats, and type LIST again. The list window will appear on the screen. Click it into scroll mode. In the lower right corner of the list window you will see another size box. Position your mouse pointer there and drag the window down as far as it will go. The window will now display nearly 18 lines instead of the normal 12. You can do the same with a second list window by changing its height and then have the capability of displaying 36 lines on the screen at any one time. Although you have to change the size of both windows, this method is still very useful because it allows you to display 36 program lines in two extended windows. This is the maximum that can be displayed in three standard windows. The extended window viewing is far better, because the windows are easier to manipulate, and you can see every part of each window. I recommend this method over the three simultaneously-displayed windows.

THE MENU BAR

The menu bar is set up to act as an immediate aid to programming. It allows you to do many things already outlined, but more efficiently. The first menu bar selection is, again, APPLE logo. This still contains the accessory programs—yes, even while you're in BASIC! The other options will be discussed below.

The FILE Menu

To view the suboptions in the FILE menu, place the pointer on FILE and hold the button down. Your choices will be displayed as New, Open . . . , Close, Save, Save As . . . , and Quit (Fig. 5-18). New is a BASIC command that we used earlier. It simply erases any program currently in memory. Slide your pointer to New and release the button. You will be prompted as to whether or not you wish to save the current program. This same sequence of events occurs when you type NEW at the keyboard, but it's faster to do it with the mouse. Access the FILE menu again and click Open . . . to open a new program. Again, you will be asked if you wish to save the program currently in memory. Another

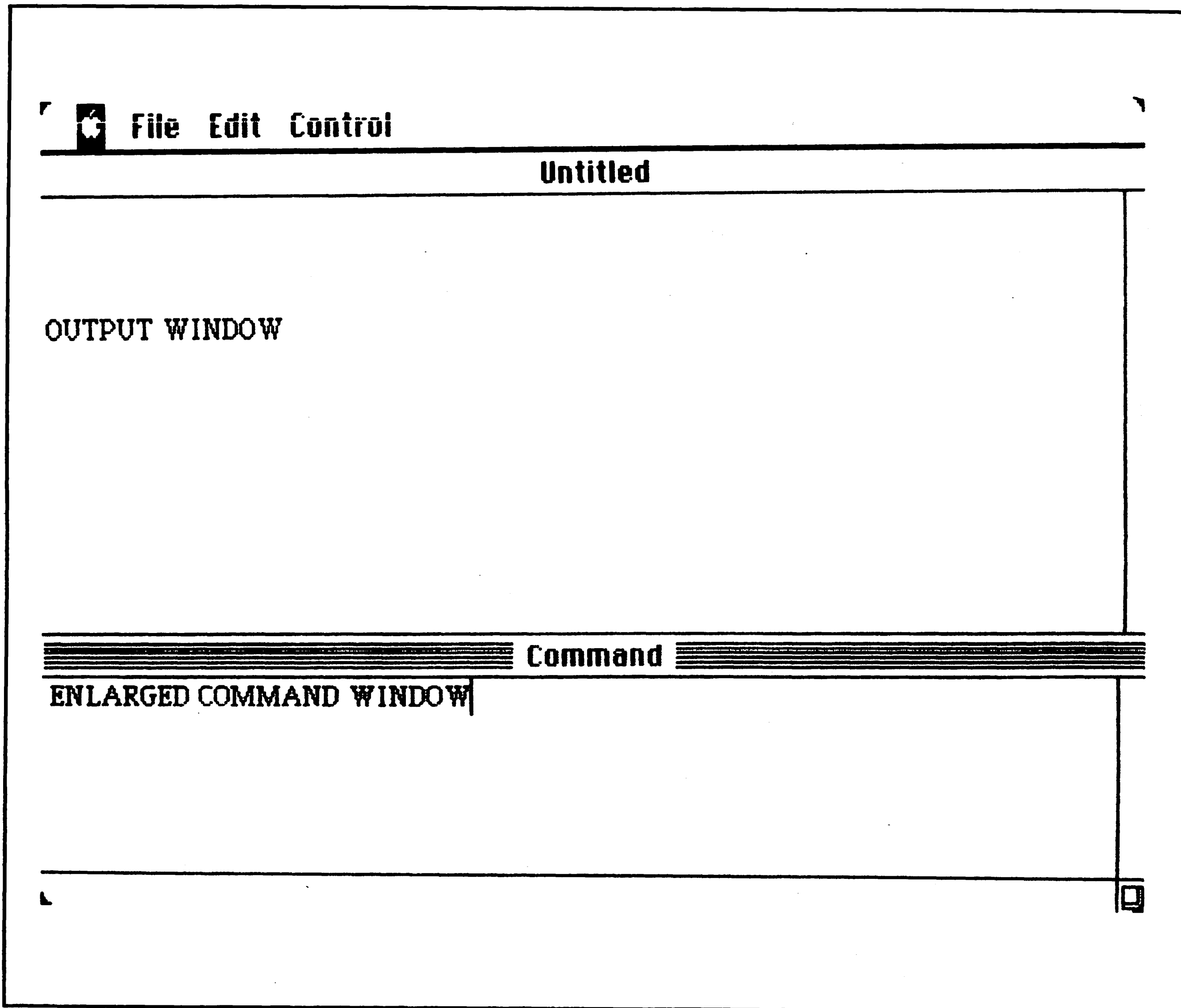


Fig. 5-17. The Command window may be doubled in height by pulling the Size Box in a downward direction.

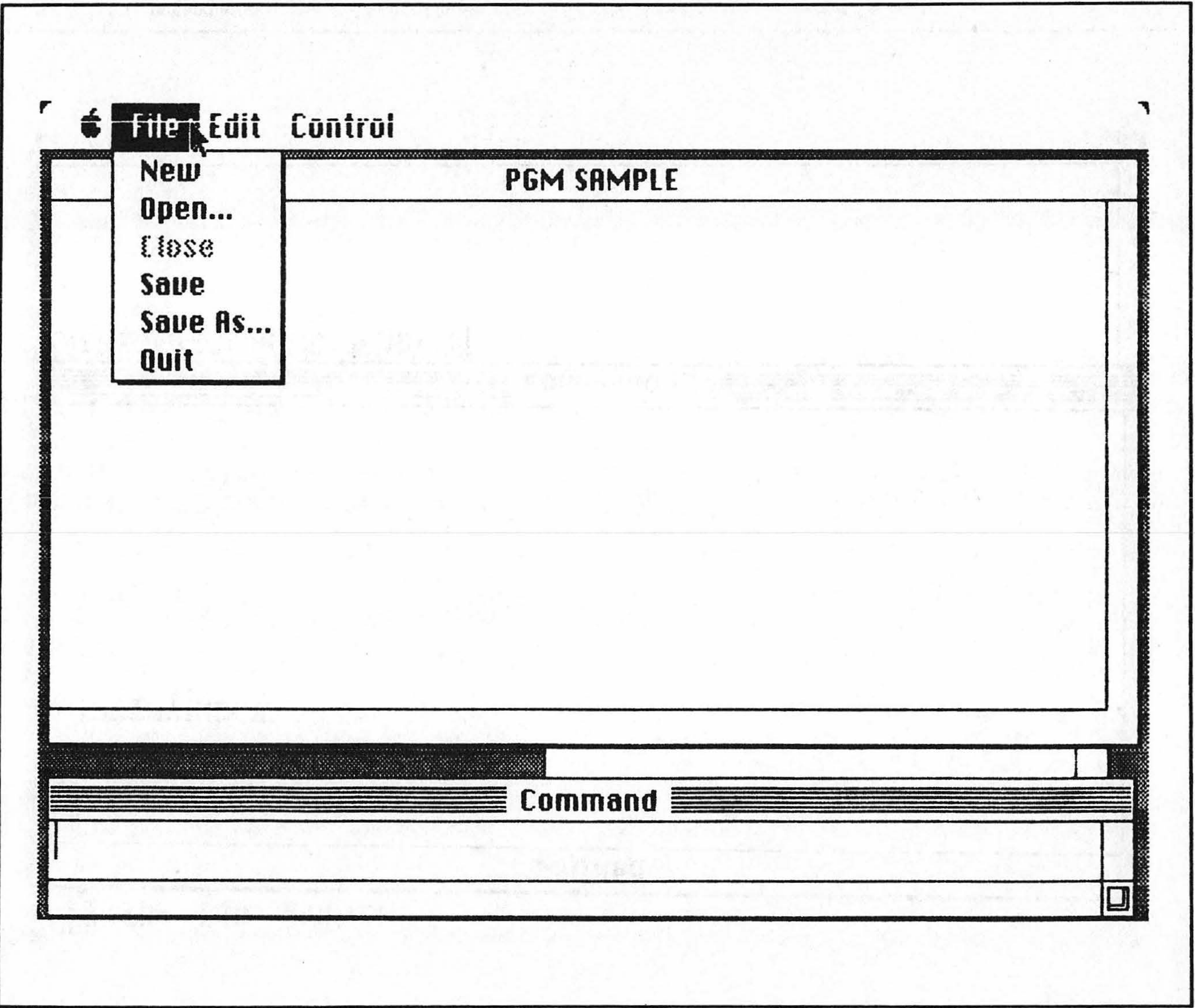


Fig. 5-18. The Microsoft BASIC File menu.

window of options will then be displayed and it will be asking you for a file name. The Open suboption is used to load a program that has been written and stored on disk. When you input a file name, the file will automatically be read from disk and loaded into immediate memory. Then type RUN and press RETURN (on the keyboard) to run the program.

The Close suboption is similar to New; it is designed to erase the current program from memory. However, you are again prompted as to whether or not you wish to save the program, because you may have made changes to the program after it was loaded. If you choose YES, the program will be saved with the changes you made. If you choose NO, the program will not be resaved, but the original will still be stored on the disk.

The next suboption is Save. This one is simple. It saves the program currently in memory (the one you're working on) to disk. If it's a new program, you will be prompted to give it a name by which you will access this program later. If it's a program that was originally loaded from disk, since it already has a name, so the revised version is saved under the same name, replacing the original version of the program.

The next suboption is Save As . . . It works just as Save does, but gives you the option of changing the name of the program. For instance, you could load a program named MYPROGRAM and save it using Save As . . . under a new name, such as HISPROGRAM. However, the original MYPROGRAM will still remain on disk, along with the newly named program.

The final suboption is Quit. If you select it, it tells the computer to exit MS-BASIC and go back to the Macintosh menu. If there is a program currently in memory, you will be given the option of saving it before MS-BASIC is exited. The Quit option is identical to the SYSTEM command in BASIC, but the mouse accesses it more easily.

The FILE menu on the menu bar allows easy access to disk read/write capabilities. I have found that it is far easier to access these capabilities with the mouse than by typing the appropriate commands as is normally done in BASIC. Certainly, you can accomplish any of the FILE menu functions with BASIC commands but not as efficiently.

The EDIT Menu

The EDIT menu contains only three options, Cut, Copy, and Paste (Fig. 5-19). These are extremely helpful options that allow you to quickly edit BASIC program lines. To demonstrate these functions, clear any program currently in memory by typing NEW. If you want to save the program, select YES from the prompt window. If not, select NO. Your screen will return to normal and we are ready to write a new program.

Type the following line:

```
10  PRINT "I AM ALWAYS FINE"
```

Press RETURN and then type RUN. You should see the phrase in quotation marks. Now, list the program. The list window will be displayed. While in scroll mode, click line 10 with the mouse so that this line is displayed in the Command window. Now for our assignment.

In this test, we wish to change line 10 to:

```
10  PRINT "I AM FINE ALWAYS"
```

Previous instructions have taught us that we can do this by deleting the word FINE, placing the cursor between AM and ALWAYS, and inserting the word FINE. However, there's another way to accomplish the same thing using the EDIT menu. All we need to do is transpose FINE and ALWAYS. Here's how it's done. Place the mouse pointer just before the F in FINE and click once to position the arrow just before the F. Now, place the pointer at the same spot again and slowly move it letter by letter. You will see each letter become reversed as you move. When the entire word FINE is in reverse, release

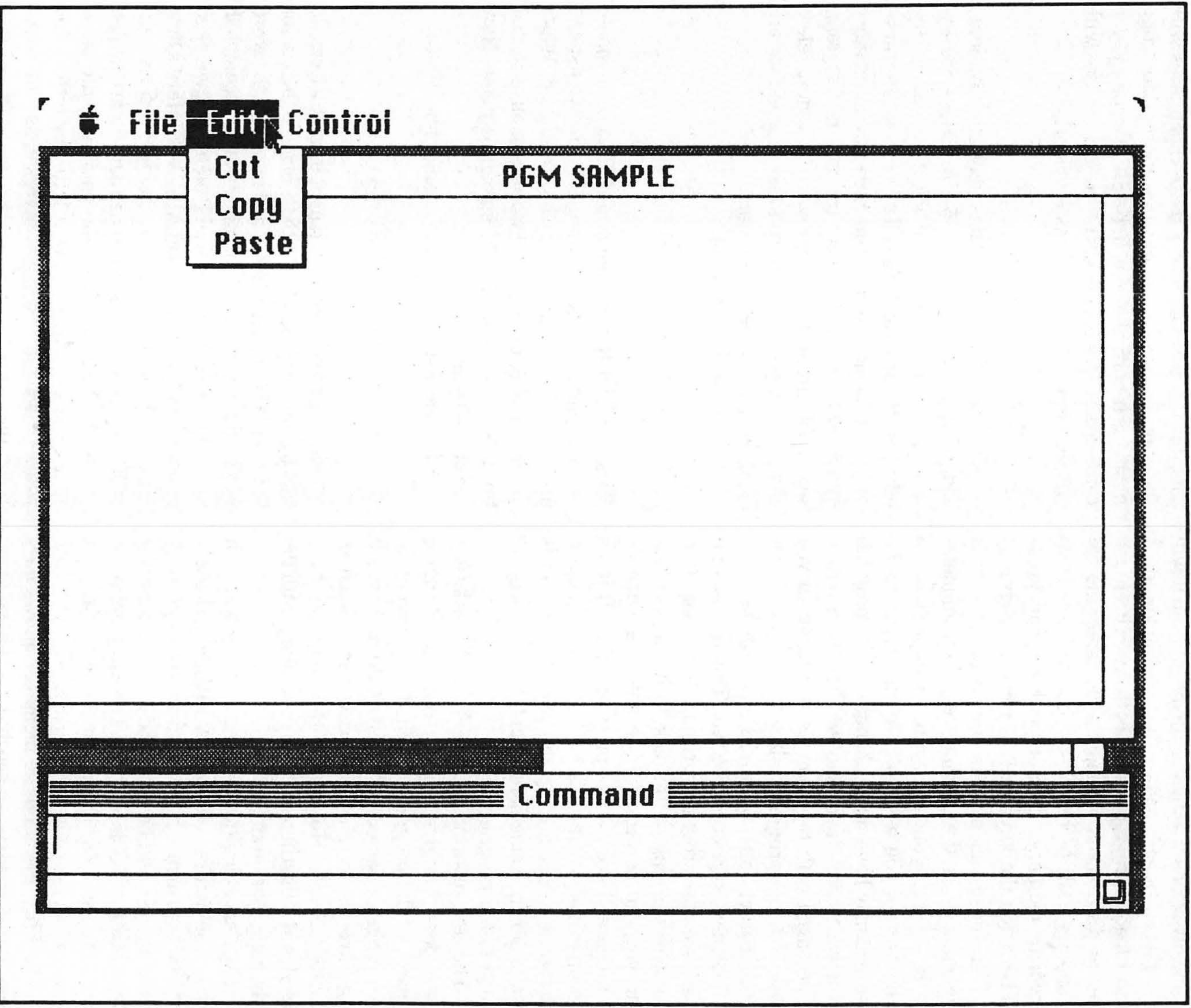


Fig. 5-19. The Microsoft BASIC Edit menu.

the mouse button. Do not include the quotation mark. Place the mouse pointer over the EDIT menu and select the Cut option. Release the mouse button. The word FINE should now disappear from the EDIT menu and the line reads:

```
10  PRINT "I AM ALWAYS"
```

Now, position the mouse pointer just ahead of the letter A in ALWAYS. Click once. This positions the cursor to a point just ahead of the A. Go back to the EDIT menu now and select Paste. When you release the button, the command window then displays:

```
10  PRINT "I AM FINEALWAYS"
```

The flashing cursor is between the E in FINE and the A in ALWAYS. To clean this up, simply press the space bar and the line is corrected. When you press RETURN, you will see that our assignment has been completed.

This assignment could have been accomplished a little more quickly if we had opted to move the space immediately preceding the F in FINE along with the word. This way, we would not have had to insert a space via the keyboard. Let's try it again, Type

```
10  PRINT "I AM ALWAYS FINE"
```

and press RETURN. Access the list window again and you will see that we are back to where we started. Click line 10 so that it appears in the command window. This time, place the mouse pointer immediately after the S in ALWAYS. While still holding the button down, move the pointer through the letter E in FINE. Again, make certain that the ending quotation mark is not included with the reversed word. Release the mouse button and click Cut from the EDIT menu. Reposition the mouse pointer on the command line immediately following the letter M in AM. Click Paste in the EDIT menu and you will see that our line reads the way we want it to.

This line can be entered into the computer by simply pressing RETURN. As you become more and more accustomed to using Cut and Paste, you will learn more about the editing functions of the Apple Macintosh. Here are a few more tips.

It is sometimes difficult to block in a word or phrase for cutting. This takes a fine touch to access just the letters you want. You can access an entire word by placing the cursor immediately before it or after it and clicking twice. Every letter in that word will be blocked in until a space is reached. If it's not necessary to grab the space, the double click method will work fine. You can access a whole portion of a line by clicking and holding prior to the first word to be grabbed and then pulling the pointer downward. This will access the entire line from the pointer to its end. In any event, once you get the hang of it, it will become second nature to you.

The third option in the EDIT menu is Copy. This works just like cut, only it does not remove the word from the line, but provides a copy of the word. If you want to duplicate a word already present in a program line, simply use Copy instead of Cut. If you want to copy an entire line, simply place the mouse pointer at the beginning of the statement portion of the line, click and pull downward. Then click Copy to store the line in memory. You can then insert this same line with another line somewhere else in the program by clicking where you want to insert the words. Again, Cut and Copy work almost identically, but Cut removes the word from the present line to be inserted later, while Copy allows you to insert the word later *without* removing the word from its present position.

Cut can also be used to simply erase a word or a number of words or letters from a line. Block in the area to be deleted and select the Cut option from the EDIT menu. The area specified is removed from the line and need not be reinserted if desired.

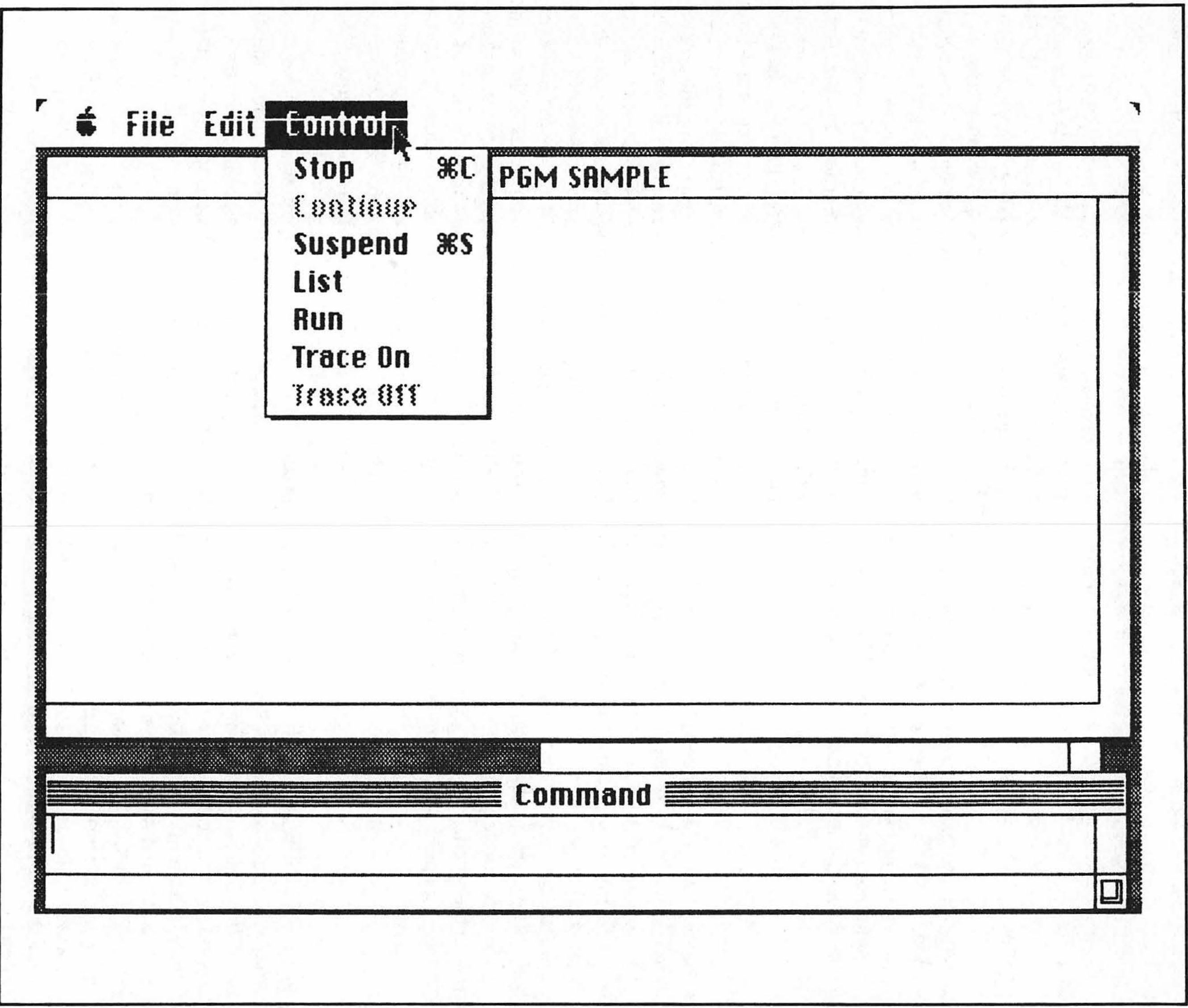


Fig. 5-20. The Microsoft BASIC Control menu.

This saves a lot of time that would be used up by pressing the backspacing key to delete letters. All in all, the editing functions available in Macintosh Microsoft BASIC are superb.

The CONTROL Menu

The CONTROL menu in the menu bar offers seven suboptions, as shown in Fig. 5-20. Stop terminates a program that is running. It then displays in the output window "Break in[program line]", indicating the program line that was being executed when Stop was selected. Stop also puts the Command window back on the screen and clears its contents. You can also stop a program by pressing both the COMMAND key (to the left of the space bar) and the C key.

Continue resumes the running of the program that was stopped using the Stop option, just as entering CONT at the keyboard would.

Suspend temporarily halts the program run. However, it does not break the program as the Stop command does. Execution resumes when you press any key, just as if you had typed COMMAND/S at the keyboard.

List simply lists the program currently in memory in the List window, just as when you type LIST and press RETURN at the keyboard.

Run runs the program currently in memory just as if you had typed RUN and pressed RETURN at the keyboard.

All of the suboptions discussed thus far simply take the place of equivalent keyboard commands. You can access these options with the mouse faster than with the keyboard, so after a while, you will find yourself using the CONTROL menu quite frequently.

The two remaining suboptions address the Trace function. Trace On activates the function, and Trace Off deactivates it. Trace is used for debugging, because it lists the line numbers of the currently running program as they are executed. The listing occurs in the list window. If a program is not running properly, it is helpful to be able to see where in the program the malfunction may be occurring. By observing the Trace prints, you can detect improper branches or program tie-ups as they occur. Trace Off should be selected when the debugging is complete.

In the FILE and CONTROL menus, every option is not always available. For instance, in the FILE menu, Close is not available if you have not opened a program. Likewise, in the CONTROL menu, Continue is not available if you haven't already stopped a program. Trace Off is not available if the trace has not been turned on. The options that are open to you in either of these two menus are printed in boldface. Those that are not are displayed more dimly. When you select those that are not open to you, the type will not be changed to reverse.

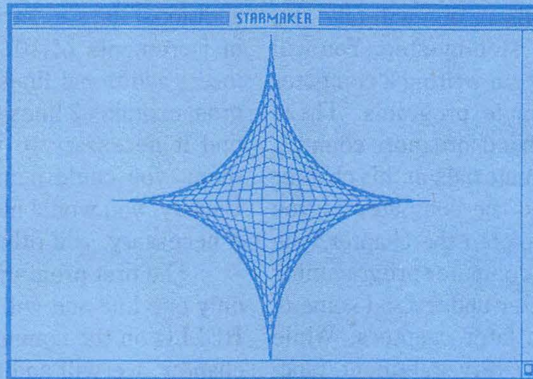
SUMMARY

Microsoft BASIC addresses many of the Macintosh special features. The language and the computer seem to be well-matched. The special editing and display functions of Microsoft BASIC certainly speed up writing, editing, and debugging a program.

This chapter has outlined the general use of the various windows and the editing functions needed to program on the Macintosh. The next three chapters cover use of Microsoft BASIC to write your own programs. The job will be far easier if you know ahead of time how to manipulate the various windows that are put there to help you. Since you're bound to make typographical errors that will have to be corrected, you will find that you will depend heavily on built-in editing functions. It is far better to practice those functions now and know the procedures before you start than to wait until you're in the middle of a BASIC learning exercise.

Computer competency comes from learning the different operations involved in programming a step at a time. This chapter has discussed the preliminaries to using the BASIC language. The next chapter will begin teaching you the language itself.

Chapter 6



Programming the Macintosh in Microsoft BASIC

The Apple Macintosh can be programmed in a language called BASIC. The original BASIC was developed at Dartmouth College and named with an acronym for *B*eginner's *A*ll-*P*urpose *S*ymbolic *I*nstruction *C*ode. While all programs written in BASIC are very similar, there are some differences between Macintosh BASIC and Apple II or *II*e BASIC, because they are for different machines. Both types of computers are programmed in BASIC, but the Macintosh uses one dialect while the Apple II series uses another. Therefore, when you switch from computer to computer, it is necessary to know something about the exact nature of the dialect of BASIC for which it is programmed.

BASIC is quite powerful and uses words that are easy for people to understand, yet tell the computer what to do. Each of these words is used as either a command, a statement, or a function. Unlike a foreign language such as French or German, the BASIC language is a limited language, so it is quite easy to master in a fairly short period of time. Each word in BASIC has an English equivalent that is descriptive of the operation the computer will perform upon execution. Take the PRINT statement, for instance. This is a statement common to all dialects of BASIC that tells the computer to print something on the display screen. Therefore, the word PRINT indicates that a printing operation will take place. This is easy to translate to English. An INPUT statement tells the computer that it should wait until the user types some information at the keyboard before it proceeds with the program, so the computer waits, just as it has been instructed to. There are many other close computer waits, just as it has been instructed to. There are many other close translations from BASIC to the English language. All of these words will be discussed in this chapter.

If you've had some experience programming computers in BASIC and are simply making a transition from another computer to the Macintosh, you may wish to skip over this chapter. This chap-

ter is really aimed at those persons who have never used BASIC before and who are relatively new to computers. The following discussion will take you on a programming journey, step-by-step. You will be shown exactly how to begin writing a computer program, starting with simple programs. These programs will then be expanded into more complex programs. If you study the materials in this chapter and type each program into the computer as you proceed, when you've completed the chapter, you will know a great deal about computer programming and will be prepared to better understand some of the programs presented in later chapters. While this chapter will not discuss every statement, function, and command in Macintosh Microsoft BASIC, it does discuss those most often used.

The Microsoft BASIC package comes complete with its own disk and a manual that covers the language. So here we don't go into great detail about the actual statements and functions of Microsoft BASIC, but instead, teach the beginner to program. Once you learn to program, the Microsoft manual is an excellent reference source. But you may also refer to it to set up the computer.

Please read through the opening chapters of the Microsoft reference manual before proceeding onward. The simple setup procedure is outlined in the manual. It is not my intent to simply rehash what you can easily obtain from the manual.

A FIRST PROGRAM

A BASIC computer program must consist of a minimum of one program line. Some programs will have hundreds or even thousands of lines, but they are developed one line at a time.

In Microsoft BASIC, each line must have a number. Normally, lines are numbered in increments of 10, so they are numbered 10, 20, 30, . . . or even 100, 110, 120, . . . You can also start numbering with 1 and increase each number by 1, but, this is not normally done because few programs are written in finished form from the start. Usually, you will want to go back through the program and insert additional lines in various locations. If you number in increments of 10, there is plenty of space to insert additional lines between lines. If your program contains 2 lines numbered 10 and 20, and you find it necessary to insert another line between them, you could number it 11 or 12 or 13, etc. Usually, you would number it 15 so that you could, if necessary, add other lines before and after it.

The first program we will write will consist of only one line and will be used to display the word HELLO on the computer screen. Throughout this chapter, we will add to this one-line program.

Begin by typing the number 10. Then press the space bar and type PRINT "HELLO", with no period after it. Press RETURN. That's all there is to it. You have written your first BASIC program. Your screen should now look like this:

```
10  PRINT "HELLO"
```

There may be other printing on the screen that appeared when you first activated the computer. Don't worry about this. Just make sure that the line discussed above is on the screen. When you pressed the RETURN key, your program was committed to computer memory and your programming job was complete.

Examine the line on the screen carefully. Make sure it looks like the example given here. Now that the program has been input, it's time to tell the computer to execute it. In Microsoft BASIC (and nearly every other dialect as well), this is done by typing RUN and then pressing RETURN. As soon as you have done this, you should see the word HELLO appear just below your program line. If for some reason the computer cannot run your program, it will display what is known as an *error message* on the screen. This is a message that gives some indication why a program line cannot be run. Often, you will see the error message SYNTAX

ERROR appear on the screen. This means that you have input a statement in BASIC that is unknown to the computer or is in a form that the computer cannot understand. If you misspelled the word PRINT in your statement, the syntax message would appear.

Chances are your program ran correctly on the first try. With a program this simple, there's very little opportunity for error. Some of my syntax instructions (instructions for writing the *statement*), in writing this first program deal with custom more than with what the computer requires. On the Macintosh, if you had typed any of the following lines, the computer would still print HELLO on the screen:

```
10PRINT"HELLO"
10PRINT"HELLO
10PRINT"HELLO
```

Some other dialects of BASIC might not accept the absence of a space between the line number and the first statement or function. Still others require that the quotation marks be on either side of the word to be displayed. Microsoft BASIC does not require the space or the quote, although it is good programming practice to include a space between the line number and to always enclose all words to be printed to the screen in quotation marks. The absence of the closing quotation marks makes no difference in this simple program, but when you write more complex programs, you can really get into serious trouble by not sticking to the proper format. Again, always include at least one space following the line number and always enclose all characters to be printed in quotation marks.

CLEARING THE SCREEN

When you ran the Hello program, you undoubtedly noticed that while the word HELLO was printed in accordance with the program, all of the information that was displayed on the screen before the program was run remained there as well. When the program was run, you saw the word HELLO displayed, but you also saw your program above it, along with the word RUN. In most situations, you will want to clear all information from the screen before your program is actually run. Most programs contain a BASIC statement at the beginning to clear the screen before anything is printed. To accomplish this, we use a CLS statement. CLS is an acronym for "clear the screen." When a statement containing this instruction is encountered, all information currently on the screen is completely erased. This gives us a clean slate for any information that will be printed by succeeding program lines.

The Hello program you typed in is still contained in computer memory. To see the program lines that are in the computer's memory, simply type LIST. This *command* tells the computer to list the program currently in memory. Try typing *LIST*, now. Be sure to press RETURN to end the command. You should immediately see your program lines appear on the screen.

We will now expand this one-line program to do the following:

1. Clear the screen.
2. Print HELLO on the screen.

Since we again wish to print the word HELLO, we will simply use the program line already in memory that accomplishes this. All we have to do is include another line to clear the screen. This line must go before the original program line because we wish to clear the screen and then print HELLO. The first program could easily have started with a 1 instead of 10, but fortunately we started numbering the lines with 10. To expand the previous program, type in the following line:

```
5  CLS
```

Now, press RETURN. Type LIST and press RETURN again. You should see the following:

```
5  CLS
10  PRINT "HELLO"
```

Your program is now twice its original size because it contains two lines instead of one. When this program is executed, the computer will first clear the screen to run the first line and then display the word HELLO to run the second. All program lines are executed in the order they are numbered in. Again, type RUN and press RETURN. All you should see on the screen this time is:

```
HELLO
```

The screen was cleared, and HELLO was printed.

TWO MODES IN BASIC

You probably haven't realized it, but by following the previous instructions, you have been using two modes of inputting information to the computer. These modes are *direct mode* and *program mode*. Whenever a line number is typed and followed by a BASIC statement line, this is the standard program mode. In other words, we are inputting a program, which is a set of instructions that are not to be executed until we run the program. However, when the word RUN has been typed in, it has not been preceded by a line number. This is the direct mode of input. When the direct mode is used, we are telling the computer to do something right now (as soon as RETURN is pressed). RUN is known as a *command.* It is an immediate call to the computer to execute the program currently in memory. List is also a command that was entered in direct mode to tell the computer to list the program in memory. Either of these commands can also be used in programming mode when preceded by a line number. Such a line is referred to as a *statement.*

Not many other dialects of BASIC allow you to use RUN and LIST in statements, but Macintosh Microsoft does, and such statements will come in handy when writing complex programs. The use of such commands in programming mode statements will be discussed later in this chapter.

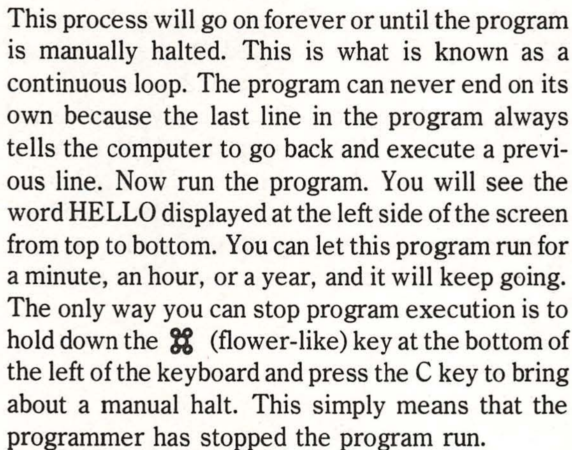
CREATING A LOOP

A *loop* is a common occurrence in BASIC programs and you will hear about loops again and again. The English language defines *loop* as a circle or a continuous repetition. This is what a loop in BASIC is as well. Let's take our previous program and make a loop so that it prints HELLO again and again and again. Type in line 20 of the following:

```
 5  CLS
10  PRINT "HELLO"
20  GOTO 10
```

Hey! There's a new word in there that we haven't discussed before. This is one of the most useful statements in BASIC and is common to all dialects. GOTO tells the computer to do just what the English language equivalent would indicate. It tells the computer to go to another line of the program and execute that line. It will continue to execute all successive lines until directed to do otherwise. Before running the program, let's discuss it further. Line 5 clears the screen. Line 10 prints the word HELLO on the cleared screen. The GOTO statement in line 20 tells the computer to go to line 10 and execute the program from that point on. Here's what will happen, at least from the computer's point of view:

1. Clear the screen (line 5).
2. Print HELLO on the screen (line 10).
3. Go back to line 10 (line 20).
4. Print HELLO on the screen (line 10).
5. Go back to line 10 (line 20).
6. Print HELLO on the screen (line 10).



7. Go back to line 10 (line 20).

This process will go on forever or until the program is manually halted. This is what is known as a continuous loop. The program can never end on its own because the last line in the program always tells the computer to go back and execute a previous line. Now run the program. You will see the word HELLO displayed at the left side of the screen from top to bottom. You can let this program run for a minute, an hour, or a year, and it will keep going. The only way you can stop program execution is to hold down the ⌘ (flower-like) key at the bottom of the left of the keyboard and press the C key to bring about a manual halt. This simply means that the programmer has stopped the program run.

This program only clears the screen once because line 5 is executed only once. The program loop, then, is between lines 10 and 20. These lines will be executed over and over again. We could also include line 5 in the loop simply by changing line 20 to:

```
20   GOTO 5
```

This change can be made using the editing functions, or you can simply type the line all over again. When this program is run, the word HELLO will be displayed in the upper left corner of the screen, but it will be constantly flickering because the screen is being cleared before the word is printed. When the computer writes on a cleared screen, it always begins printing in the upper left corner unless it is told (programmed) to begin elsewhere. When the screen is not cleared, each succeeding printed character will be displayed at the left side of the screen one row below the previously printed characters.

Admittedly, none of the programs we've written so far have been useful for anything other than instruction. The computer has not been used to display useful information on the screen. Please be patient. It is absolutely mandatory that you understand the basics behind computer programming before you can begin to make your programs "intelligent." Be assured that if you can understand the concepts behind the programs already discussed, you're getting closer to being able to write some good programs on your own. So far, you've learned a little about the PRINT statement, CLS, and GOTO. Hopefully, you're also becoming comfortable with writing programs. Do not proceed further unless you completely understand the program examples presented.

MORE ABOUT LOOPS

The previous program displayed the word HELLO over and over again on the screen. The GOTO statement is also referred to as a branch or a GOTO branch. There are other types of branch statements that we will be learning about later, but GOTO is the one used most often.

Endless loops such as the one encountered in the previous program are useful in some programs, but most of the time, it is necessary to exit the loop after the certain number of passes have occurred. Think of a pass as one cycle of the loop. By modifying the previous program, we can have the loop end after a specified number of passes. The following program is simply the same old program with two extra lines.

```
 5   CLS
10   PRINT "HELLO"
15   C=C+1
16   IF C=10 THEN END
20   GOTO 10
```

Two lines have been added between previous program lines 10 and 20. Line 15 would not seem to contain a statement, but it does, or at least the computer knows that it does. The *assignment* or the LET statement which assigns a value to a variable. In this case, the variable is C. Most computers

handle LET, but do not require its use. Line 15 could be written:

```
15  LET C=C+1
```

However, to program efficiently, you should input a line in the way that requires the least typing. In Microsoft BASIC, the LET statement is optional. The line C=C+1 is equivalent to LET C=C+1. The computer assumes the LET statement in line 15. Therefore, there is really no need to use LET in Microsoft BASIC.

The method of assignment shown in line 15 is sometimes referred to as a *counting* assignment. Each time line 15 is executed, variable C will be *incremented*, or *increased* by 1.

Line 16 contains another new statement for this discussion. This is called the *IF-THEN* statement. This is a test statement. It checks to see if a certain condition is true, and if it is, then it may bring about a branch or execute another statement. Let's go through a partial sample run to see how lines 15 and 16 change the program. Line 5 clears the screen. Line 10 prints HELLO on the blank screen. Line 15 is then encountered. At this point in the execution, C is equal to 0 because C has not yet been assigned a value. Value assigned in line 15 will be assigned to the C to the left of the equal sign. The old value of C is to the right. Since the old value of C is 0, the new value of C will be equal to 0 + 1, or 1.

Now, line 16 says if C is equal to 10, then end the program. When line 16 is executed for the first time, C will be equal to 1, so line 16 technically does nothing. It will only execute when C is equal to 10. Next, the GOTO statement in line 20 is encountered, and the loop begins. Line 10 prints HELLO again, and line 15 is then encountered for the second time. Remember, the new value of C (the one to be assigned) is located to the left of the equal sign, while the old value of C is to the right. Since line 15 has already been executed once at this point, the old value of C is now equal to 1. Line 15 states that C is equal to C + 1, making the newest value of C equal to 1 + 1, or 2. Again, line 16 detect that the new value of C is not equal to 10, so the END statement is not executed. On the next pass of the loop, C is incremented to 3 (2 + 1). The loop will continue to cycle until C is finally incremented to a value of 10. At this point, line 16 detects the value of 10 and ends the program. The END statement is common to nearly every dialect of BASIC and is a signal to the computer to end the program. Here is another way this program could be written:

```
 5  CLS
10  PRINT "HELLO"
15  C=C+1
16  IF C=10 THEN 30
20  GOTO 10
30  END
```

This program does exactly the same thing as the previous one. You can now run either program and you will see that HELLO is printed 10 times on the screen, and the program terminates. What was previously an endless loop has now been transformed into a loop with 10 passes. In the second program, line 16 has been altered to bring about a branch to line 30. Line 16 could also have been written:

```
16  IF C=10 THEN GOTO 30
```

In Microsoft BASIC, the GOTO does not have to be included in an IF-THEN statement branch. It is assumed. When this program is executed, C is again incremented by 1 during each pass of the loop. When it is equal to 10, line 16 detects this condition and brings about a branch to line 30. Line 30 contains the END statement, which terminates program execution. The first example was the more efficient, but this second example will do exactly the same thing, although it requires an additional program line.

You will notice in the second example that

when C is incremented to 10 in line 15, line 16 detects this condition or value of C and branches directly to line 30, skipping line 20 (the GOTO) was not executed at all on the last press of the loop.

Branch statements are extremely important to BASIC programs and there are a variety of different types of them. Some other branch statements will also be discussed in this chapter.

FOR-NEXT LOOPS

Probably the most useful loop found in BASIC is the FOR-NEXT loop. Just like IF-THEN, FOR-NEXT may be thought of as one complex statement. In order to demonstrate the FOR-NEXT loop, let's start from scratch again and erase the program in memory. To do this, you use a command. (You may remember that a command is entered in direct mode without a line number.) Type NEW and press RETURN. Respond to the window prompt by placing your mouse-controlled arrow on the "NO" and click. You can confirm that the program has been erased by using the LIST command. When you enter LIST, nothing is displayed in the LIST window. Therefore, the program is gone.

The following program demonstrates the use of a FOR-NEXT loop. It does exactly the same thing that the previous program did—it prints HELLO on the screen ten times.

```
10  CLS
20  FOR X=1 TO 10
30  PRINT "HELLO"
40. NEXT X
```

As before, line 10 clears the screen. Line 20 begins the FOR-NEXT loop. Only the FOR portion of the statement is contained on this line. Think of FOR as starting the loop and NEXT as ending it. Any lines between FOR and NEXT are a part of the loop. In this case, there is only one line between FOR and NEXT. Line 30 contains the PRINT statement that will display HELLO on the screen.

The letter X is a variable and X=1 to 10 means it is assigned a value of from 1 to 10 each time the loop is executed. Here's how it works. When line 20 is executed for the first time, X is assigned a value of 1. Line 30 is then executed and HELLO is displayed. When NEXT X is encountered in line 40, the computer branches back to the line containing FOR. The loop now begins its second pass when line 20 is executed. Variable X is now assigned a value of 2. Unless told otherwise, the FOR-NEXT statement will automatically increment its value by 1 during each pass. Line 30 is executed again. At line 40 the program automatically branches to line 20, and the loop continues cycling. When 9 passes have been completed, the last branch to line 20 occurs. The variable X is now equal to 10. Line 30 is executed, but when line 40 is encountered, the loop has timed out. This means that X has reached its maximum assigned value of 10 and the loop is automatically exited. At this point, the program will end because there are no other lines following line 40. It is not necessary to use an END statement in this case, because when there are no other lines to execute, there is an automatic termination. The program could include an additional line following the loop sequence, such as:

```
50  END
```

However, since the program automatically terminates when there are no more program lines to execute, line 50 is superfluous. If it helps you understand the program better, it can certainly be included.

This example has shown how a loop may be used to execute a program line or lines contained within the loop a specific number of times. We can demonstrate another useful purpose of loops by changing line 30 to:

```
30  PRINT X
```

This is the first time the PRINT statement has been used to print the value of a variable on the screen. In this case, quotation marks are not used. When the PRINT statement is used with a variable, the value of that variable will be displayed on the screen. Run the program in its present form and you will see that when the PRINT X statement in line 30 executes, the screen displays the value of X. A sequence of the numbers 1 through 10 will be seen on the screen. **Remember,** do *not* use the quotation marks when you wish to display the value of a variable on the screen. To see what happens when you do, change line 30 to:

```
30  PRINT "X"
```

When this program is run, the letter X will be displayed on the screen 10 times. The quotation marks tell the computer not to print the value of variable X, but to print the letter X.

For the next example, change line 30 back to:

```
30  PRINT X
```

As was mentioned earlier that a FOR-NEXT loop will automatically increment its variable by 1 unless told to do otherwise. The FOR-NEXT loop is also known as the FOR-NEXT-STEP loop. The STEP portion indicates the value by which the variable is to be incremented. In the previous program, the computer assumed the step value in line 20, because it was not included. When you typed line 20 as

```
20  FOR X=1 TO 10
```

the computer saw it as:

```
20  FOR X=1 TO 10 STEP 1
```

It automatically incremented X by 1 on each pass of the loop. Let's change the program at this point to bring about a different step value. To do this, we will use another command that is designed to make editing of programs extremely easy. In direct mode, type EDIT 20 and then press RETURN. This is a command to the computer telling it that you wish to edit line 20. When this command is entered, line 20 will appear on the screen and the flashing cursor will be seen beneath the 2 in 20. Use the mouse edit function described in the last chapter to move the cursor to one character past the 10 in line 20. Now, press the space bar and then type STEP 2. Your program line should look like this:

```
20  FOR X=1 TO 10 STEP 2
```

This line now tells the computer to increment X by 2 during each pass of the loop. Here's how it will work. When line 20 is first executed, the value of X will be 1, the lowest number in the assignment. When the loop cycles for the second time, X will be incremented by 2, making it equivalent to X=X+2 on the second and all succeeding loops. On the first cycle, however, X will always be equal to the lowest value in the assignment line. In this case, the value is 1. Now, run the program by typing RUN and pressing RETURN. Since line 30 has been written to display the value of X, you will now see the numbers 1, 3, 5, 7, and 9 displayed, each on a separate screen line. This is the result of creating a FOR-NEXT loop that is stepped by 2.

You may be surprised that the final value of the loop was not displayed. This value is 10. However, this number does not come up when X is stepped by 2. Following the number 9, the next two-step jump would be 11. Since X can only be equal to a maximum value of 10, the loop terminates. This example provided an odd number count. This will always be the case when a step or 2 or any even number is used and the loop starts on an odd number. However, if the loop started with 2 (X=2 TO 10 STEP 2), the value of X would always be even. With a starting value of 2, the count would be 2, 4, 6, 8, and 10.

FOR-NEXT loops may also start with 0. If we change line 20 to read

```
20  FOR X=0 TO 10 STEP 2
```

the count sequence will be 0, 2, 4, 6, 8, 10. Notice here that an extra number is added. With a starting value of 2, only 5 numbers were displayed. With a starting value of 0, 6 were displayed. Let's go further with this. Change line 20 to:

```
20  FOR X=0 TO 10
```

Remember, since in this case the step value has not been included in our statement, the computer assumes a step value of 1. Run the program. You will see that the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 are displayed on the screen. How many passes has the loop made? The answer is no longer 10. The loop has made 11 passes and you will see 11 numbers displayed on the screen.

When programming, it is always necessary to remember that 0 is a number. The computer treats it just like any other number, so if you want a loop to start at 0 and end at 10, it is the same (cycle-wise) as creating the loop that counts from 1 to 11. Either way, 11 passes will be made. When learning math in school, the importance of the number 0 is greatly under-emphasized. The process of not thinking of 0 as a viable number creates many headaches for new computer programmers. Remember this and when you debug programs you may find it a bit simpler.

Now, you may want to review the entire section on loops and especially the section on FOR-NEXT loops. When programming in BASIC, you will rely heavily on FOR-NEXT loops. These will be used in nearly every program to achieve the greatest efficiency. If you do not understand how a FOR-NEXT loop works, reread the previous material until you understand them completely. When programming in any type of computer language, loops are extremely useful and allow complex operations to be relegated to a minimum number of program lines.

GOING FURTHER WITH FOR-NEXT LOOPS

Now that you're sure you understand the preceding section on loops and FOR-NEXT loops, let's delve further into the subject. All the previous examples used positive numbers and counted upward, or incremented. However, FOR-NEXT loops are not restricted to positive numbers or to incrementing or adding to these numbers. Using the previous program, change line 20 to:

```
20  FOR X=-10 TO 1
```

Now run this program. You will see the value of X displayed on the screen starting with -10 and moving up to positive 1. In this case, the loop makes 12 passes and you will see that when X is equal to 0, this is just as much a pass as when it is equal to any other number. The loop is still incrementing by +1. When X is equal to -10 during one pass, it will be equal to -10 + 1, or -9 during the next. When the pass is completed where X is equal to -1, the next pass will assign X the value of -1 + 1, or 0. Again, this is still a positive step pass, in that X is always incremented by +1.

However, we can also make the FOR-NEXT loop count backward. If you've skipped ahead of me, you may have tried changing line 20 to:

```
20  X=10 TO 1
```

This is often a first-time attempt to get a loop to decrement or decrease the value of X. This won't work, at least not in this form. Here, someone has attempted to get the system to count backwards from 10 to 1, but when a loop is to be decremented, it is necessary to insert a negative STEP value. Try this:

```
20  FOR X=10 TO 1 STEP -1
```

Now run the program. It should work perfectly, and you will receive a count on the screen starting with 10 and ending at 1. This loop is almost identical to:

```
20  FOR X=1 TO 10
```

In both examples, a total of ten passes will be made. However, if you're going to use the value of X within the loop, the situation is completely different. In the first example, X will be equal to 10 on the first pass, 9 on the second, 8 on the third, and so on. In the second example, X will be equal to 1 on the first pass, 2 on the second, 3 on the third, and so on. Remember, to get a loop to count backward or decrement, it is absolutely essential to use, a negative step value. If you wish to experiment, change the negative step value to −2 or −3. In each case the value of X will be decremented by the step value.

In all of the FOR-NEXT loop examples so far, whole numbers have been used as step values, but FOR-NEXT loops are certainly not limited to them. Let's change line 20 again to:

```
20  FOR X=1 TO 10 STEP .5
```

Here, instead of stepping by a whole number on each pass, the computer will step by the value .5. Now run the program. You will see that this step value is reflected in the loop. The variable X will now be equal to 1, 1.5, 2, 2.5, etc. This particular loop will make 19 passes, since the step value from 1 to 10 in increments of .5 yields 19 different numbers between and including 1 and 10. You could also use −.5 and a loop value starting at 10 and ending at 1 to produce the same figures in descending order.

Variables may be inserted for any numeric value in the loop. Let's erase the program currently in memory by typing NEW and starting over again. Type in the following program:

```
10  CLS
20  A=1
30  B=10
40  C=.5
50  FOR X=A TO B STEP C
60  PRINT X
70  NEXT X
```

This program will, again, count from 1 to 10 in increments of .5. Here, however, variables have been used in the loop. In line 50, X will have a low value of A, a high value of B, and a step value of C. In this case, the letters A, B, and C have been used to represent real numbers just as X has been used to represent real numbers. 20 through 40 assign the values to A, B and C. Remember, these are LET statements even though LET is not actually used. Unless the values of A, B, and C are changed by other LET statement lines in the program, they will remain equal to the values they are assigned. Line 50 can be translated to

```
50  FOR X=1 TO STEP .5
```

because A is equal to 1, B to 10, and C to .5.

At this point, I'd encourage you to review all of the materials presented thus far on loops and begin experimenting on your own. Try any values you want in the FOR-NEXT loops, and see if you can get your loops working properly. If you use a wide value range such as

```
FOR X=1 TO 20000
```

it will take your computer some time to count through all these numbers. It will also take far longer to display all these values. Computers work very rapidly (in millionths of a second), but they don't work instantaneously. The more program lines contained in a FOR-NEXT loop, the longer the execution time. For this reason, it is essential that any lines that do not absolutely have to be in the loop be placed outside it. Unnecessary lines in

FOR-NEXT loops (ones that could have been placed outside the loop) are one of the biggest abuses of the FOR-NEXT loop in computer programming.

MORE ON PRINT STATEMENTS

As you recall, a PRINT statement was the first kind introduced in this chapter. PRINT has many uses, although only two have been shown thus far. We know that any characters enclosed in quotation marks following the PRINT statement will be displayed on the screen exactly as typed. If the quotation marks are omitted and a variable is used instead, the value of the variable will be displayed.

Erase the program currently in memory and type in the following lines:

```
10  CLS
20  FOR X=1 TO 5
30  PRINT "HELLO"
40  NEXT X
```

This is almost identical to a previous program, but this one will be used to take a closer look at the PRINT statement. When you run this program, you will see HELLO displayed 5 times on the screen in a vertical format. Each word is written directly beneath the previous one. Now, let's edit program line 30. As before, simply type in EDIT 30 and press RETURN. Line 30 will then be displayed on the screen. Use the mouse to move the cursor one character past the closing quotation mark. Now, click the mouse and type in a semicolon and press RETURN. Program line 30 should now look like this:

```
30  PRINT "HELLO";
```

Run the program. Look what happened! The word HELLO was again printed five times, but instead of each being printed below the previous one, the HELLOs were printed next to each other. The semicolon, then, instructs the computer to display the characters in a horizontal format with the second HELLO immediately to the right of the first. The result is HELLOHELLOHELLOHELLO HELLO. Of course, all of the words are run together, making one long word, but this can easily be corrected. Edit line 30 again. Use the mouse to move the cursor to the point just beneath the closing quotation mark. Click the mouse and hit the space bar once. Line 30 should now look like this:

```
30  PRINT "HELLO ";
```

Press RETURN and run the program again. This looks much neater. The computer is doing the same thing it did before, only there is an extra character in quotation marks. This character is a space which serves to separate the words. I stated earlier that a zero is treated like any other number by the computer. The same applies to the space character. While you can't always see the character produced by the space bar, the computer can and it treats a space just as it does any other character.

When using PRINT statements, you have the option of displaying information in horizontal relation to one another just described by using a semicolon following the quotation marks. Each item printed will be in a vertical relation to the others when the semicolon is omitted. Now, change line 30 to:

```
30  PRINT X
```

When this program is run, the numbers 1 through 5 will be displayed one under the other, in a *vertical format*. At this point, you can change line 30 to:

```
30  PRINT X;
```

The semicolon means that the values of X will be

printed next to each other, in a *horizontal format.*

Now, let's combine a quoted word and a numeric variable in a single PRINT statement line. Change line 30 to:

PRINT X "HELLO"

Now run the program. Here, you will see that the value of X is displayed followed by a space and then the word HELLO. The space is automatically inserted by the computer and has nothing to do with the spacing between the variable in the program line and the quoted phrase. We can reverse this by first printing the quoted phrase and then the value of X. Change line 30 to:

```
30  PRINT "HELLO" X
```

When this program is run, HELLO will be displayed followed by the value of X. Many dialects of BASIC require the use of a semicolon to separate quoted phrases or quoted phrases and variables to be printed in horizontal format with one PRINT statement. The horizontal format here applies to a single line only. In these examples, variable X and the word HELLO are to be printed next to each other, although on the next pass of the loop, this same horizontal line is printed beneath the one before it. Now change line 30 to:

```
30  PRINT "HELLO" X;
```

This is a true horizontal format because everytime line 30 is executed the items are printed next to the last items until they've printed all the way across the screen. It is not necessary to insert a space with in the quoted phrase, because whenever a numeric variable is printed, a space is automatically included before and after it. Standard computer BASIC syntax usually specifies that a semicolon be used between connected words or words and variables handled by one PRINT statement line. You will more often see lines like

```
30  PRINT "HELLO";X
```

than you will see

```
30  PRINT "HELLO" X
```

These lines will both do exactly the same thing. Each will print the value of X to the right of HELLO. This is still a vertical format PRINT statement, since a semicolon does not follow the end of the line. A semicolon between HELLO and X is not required in Macintosh Microsoft BASIC, but it is in most other dialects of BASIC. To get a complete horizontal format, you can change line 30 to:

```
30  PRINT "HELLO";X;
```

which would be standard for other dialects of BASIC.

When a comma is used to separate two elements of a PRINT statement, the results are quite different. Change line 30 to read:

```
30  PRINT "HELLO",X
```

Now run the program. Here, the word HELLO is printed and then followed by 10 spaces before the value of X is printed. This creates an effect similar to that of a tab key on a typewriter. When the comma is used, the computer displays information in two print zones. The word HELLO is in one zone, and the value of X is in the other. Each zone is 15 characters wide. This is handy for displaying menus, charts, or any information in columns.

VARIABLES

We have already dealt with one type of variable in the preceding discussions. However, in BASIC programming, there are two specific types. A vari-

able is a symbol (usually a letter) that is used to represent a value or quantity. In BASIC, most variables are comprised of letters, although they can be combinations of letters and numbers, but never numbers alone. The variables we have been dealing with are called *numeric* variables because the letters are used to represent numbers. In discussing FOR-NEXT loops, the variable X was used to represent the count value in the loop. Any other letter would have sufficed as well, or even a combination of letters, such as XYZ or a letter/number combination, such as X1.

In addition to numeric variables, there is another type of variable that is used to handle what are known as *string values*. String values are often a "string" of words or letters, or combinations of letters and numbers. Just as we used numeric variables to represent numbers, we can use alphanumeric, or *string variables* to represent letters, numbers, or combinations of numbers and letters. The difference between a numeric variable and an alphanumeric variable is that even though an alphanumeric variable can be a number, no arithmetic can be performed with that variable. So, if a number is just being displayed, you may prefer to store it in a string variable.

A string variable is represented in exactly the same way a numeric variable is, with one exception. A string variable always terminates with a dollar sign ($). Where the letter A may be used as a numeric variable, A$ can be used as an alphanumeric variable.

The following program demonstrates the use of a string variable to represent the word HELLO.

```
10 CLS
20 A$="HELLO"
30 FOR X=1 TO 10
40 PRINT A$
50 NEXT X
```

Here, the word HELLO is assigned to the string variable A$. Whenever a string variable is assigned, its value must be contained in quotation marks, regardless of whether it consists of letters, numbers, or combinations thereof. If the line were typed as A$=HELLO, the program would not run and you would receive a syntax error message. When this program is run, the word HELLO will be displayed 10 times. To the computer, A$ means HELLO because this is its assigned value. By inserting a semicolon following A$ in line 40, the word HELLO will be repeated in *horizontal format*. To see how the semicolon is quite important, change line 40 to:

```
40 PRINT A$X
```

Remember, in Microsoft BASIC, it's not always necessary to separate the elements in a PRINT statement by a semicolon to have them printed next to each other on the same line. When this program is run, the word HELLO will be printed and to its right (separated by a space), the value of numeric variable X will be printed. Although this line 40 works just fine, we could also have written in this line as

```
40 PRINT A$;X
```

and achieved the same results. Again, the semicolon is unnecessary.

However, let's reverse this sequence and print X followed by A$. Change line 40 to:

```
40 PRINT XA$
```

Again, we have not used the semicolon for separation. Run the program. Horrors! Nothing is happening. This is due to the fact that *in this case*, it is absolutely essential to use a semicolon between the numeric variable X and the string variable A$. You fooled the computer! Remember, string variables

may be composed of a letter, several letters, or a combination of letters and numbers. The dollar sign always marks the end of the string variable. In the first example where A$ was printed and then followed by X, the computer recognized the dollar sign as terminating one type of variable and then recognized X as another variable. However, in the second example, the computer interpreted XA$ not as the numeric variable X followed by the string variable A$, but simply as one string variable named XA$. There was no assignment to a string variable named XA$ in the previous program line. Therefore, the computer assumes that XA$ is equal to nothing . . . not zero, but nothing. The actual assignment the computer made might have looked like this if it were written in a program line:

```
XA$=""
```

Notice that there is not even a space character here. XA$ is not equal to anything. Therefore, when the computer interpreted line 40, it read it as "print nothing 10 times." That's exactly what it did. It allotted a screen row each time the loop cycled, and on that screen row, it did exactly what it thought it was supposed to do—it didn't print anything.

To get the program to work properly, change line 40 to:

```
40  PRINT X;A$
```

Now run the program. You should see the value of X displayed and on the same line, separated by a space, the word HELLO. This combination is printed in vertical format 10 times. Now change line 40 to:

```
40  PRINT X;A$;
```

When the program is run, the combination of X and A$ is displayed in horizontal format across the screen. A portion is printed on the next lower row simply because it was impossible to squeeze 10 printings of the numeric variable and the word on one program line. When a computer display exceeds the length of one line, as determined by a width statement earlier in the program, the remainder is automatically displayed on the next line. If a width statement has not been used, the Macintosh assumes the line is of infinite length.

NAMING VARIABLES

Since our discussion will involve more and more about variables, it is appropriate to discuss how to name them in a little more detail. First, you cannot use all of the characters on the keyboard as part of a string variable name (value). You may use any number and any letter (uppercase or lowercase), and you may also use the at sign (@) and the underline character (___). I'm speaking here of the variable itself and not the assignment made in quotation marks. You could use a variable name such as:

```
@437KP$ = "HELLO"
```

You could not use

```
,,73%W$ = "HELLO"
```

because commas are not allowed in a string name. CAR$ = "HELLO" is fine. CART = "HELLO" is not. The dollar sign correctly appears at the end of the name, but another dollar sign is included in the name.

When naming variables, you cannot use any of the key words, words used in statements or as commands, in Microsoft BASIC. For example, LISST = 1 is fine, because LISST is not a statement, function, or command. LIST = 1 is unacceptable and will result in an error message, because this word is used in Microsoft BASIC. This is not true with string variables, however. LET LIST$ = "HELLO" is fine, because LIST$ is not a statement, command, or function. However, there are a

few statements in Microsoft BASIC that do end with a dollar sign. These are CHR$, TIME$, MID$, RIGHT$, DATE$. If you attempt to use any of these as string variable names, an error message will result.

MANIPULATING NUMERIC VARIABLES

Numeric variables can be manipulated or processed in more ways than string variables, because the computer can compute with them. For instance, if numeric variable A is assigned a value of 10 and numeric variable B is assigned a value of 5, a computer line such as

```
40  C=A/B
```

is permissible. The slash indicates that A is to be divided by B. The computer will automatically substitute the assigned values of A and B, complete the mathematical operation, and assign to variable C the resulting value of 2 (10/5=2). Other symbols that represent mathematical operations are defined in the following list:

```
C = A + B   (add A and B)
C = A - B   (subtract B from A)
C = A * B   (multiply A by B)
C = SQR(A)  (find the square root of A)
C = A ^ B   (raise A to the power of B)
```

String variables, even though they may be used to represent numeric quantities, cannot be used to perform mathematical operations. For instance take the following program:

```
10  A$ = "4"
20  B$ = "8"
30  C$ = B$/A$
40  PRINT C$
```

You may input this program if you want to see an error message, but it will not run. Lines 10 and 20 are perfectly legal (as in line 40, for that matter). It's line 30 that creates problems. These are string variables and standard mathematical operations cannot be performed on string variables. If you change line 30 to

```
30  C$ = B$ * A$
```

or

```
30  C$ = B$ - A$
```

you will still get the same error message "type mismatch," which simply means that you tried to perform a mathematical operation on a string variable. It simply won't work. There is a way that string variables can be converted to numeric variables, but we'll get into that later.

One "mathematical" operation can be performed with string variables. To see it, input the following program

```
10  A$ = "3"
20  B$ = "4"
30  C$ = A$ + B$
40  PRINT C$
```

I still stick to my original statement that standard mathematical operations cannot be performed with string variables. The above program would seem to indicate a mathematical function using string variables, but when you run the program, you will find that this is not the case. When the program is run, the screen will display the number 34, which is the "value" of C$—the result of "adding" A$ to B$. You can see that a standard mathematical operation has *not* been performed, because the sum of 3 + 4 is not 34. Instead, the computer simply printed the values of A$ and B$ side by side. To further clarify this, change lines 10 and 20 to:

```
10  A$ = "HELLO"
```

```
20  B$ = "GOODBYE"
```

After these changes are made, run the program again and the computer will display the line:

'HELLOGOODBYE"

When two *string variables* are added, the second is simply tacked onto the end of the first.

INPUT STATEMENTS

In many types of computer programs, the user is expected to input information from the keyboard while the program is running. The computer uses this information in the context of the program. Sometimes, it will display the answer to a mathematical problem based upon the numbers typed at the keyboard. At other times, the keyboard input is used to help the computer decide which program lines to execute. The INPUT statement resembles the PRINT statement in many ways.

When the computer encounters an INPUT statement in a program, it temporarily halts execution and will not resume until something has been typed at the keyboard and the RETURN key is pressed. The information typed in at the keyboard is then assigned to a variable. This may be a string variable or a numeric variable, depending on the information the computer is looking for.

The following program illustrates the use of INPUT to assign the user input from the keyboard to a numeric variable. It will then process a mathematical function using the input as one of its variables.

```
10  CLS
20  INPUT A
30  X = A * 2
40  PRINT X
50  END
```

You should see some very familiar statements and operations in this program. The only thing that is new is the INPUT statement in line 20. This program has been written to allow the user to input any number. This number is then multiplied by 2, and the answer is assigned to the variable X, which is then printed on the screen. When you run the program, line 10 will clear the screen. When line 20 is executed, the computer will halt its run until you type in a number and press RETURN. The number is assigned to variable A. Note that this letter follows the INPUT statement. The computer is saying to you, "Give me the value of variable A." When you press RETURN, this signals the computer that the value has been input. Line 30 assigns to a new variable, X, the value of A times 2. Line 40 tells the computer to display the value of variable X on the screen.

Run this program. The screen will clear, and all you will see is a question mark. This is a sign that the computer is looking for input from the user. Now, type in the number 10 and press RETURN. Immediately, the computer will display a 20 on the screen (10 * 2 = 20), and the program terminates. You can use this program over and over again to automatically double any value you input at the keyboard.

Since the variable used with the INPUT statement in line 20 is not terminated with a dollar sign, this is a signal to the *computer* to exact a numeric input rather than a string input. (Later we will get into how the user knows what to type.) Therefore, anything typed in via the keyboard must be a number. The computer will accept nothing else. Run the program again, and when the question mark appears, type in a few letters. As soon as you hit RETURN, you will get the error message "Redo from start." The computer is saying "No way!" It knows that it must have numeric input to continue execution, and it will not continue until it gets numeric input. You can simply press RETURN without typing in anything else and execution will continue. By simply pressing RETURN, you have

entered a value of 0 and its output answer will still be 0, since 2 times zero is still 0. This program can be expanded upon endlessly to perform many different types of mathematical operations. It also needs to be cleaned up a bit, because the number you type in remains on the screen while the answer is displayed. The following program clears the screen, allows you to input a number, clears the screen again, and then prints only the answer. This cleans up a lot of the garbage used to provide the computer with the information it needs to execute the program. It is the same program with an additional line.

```
10  CLS
20  INPUT A
25  CLS
30  X = A * 2
40  PRINT X
50  END
```

Line 25 includes the extra CLS statement which removes your input from the screen (but not from the computer's memory). As soon as you enter your number, it will be displayed on the screen, but when you press RETURN, execution begins again and line 25 clears the screen once again, leaving a clean surface for the display of X's value.

One problem with this program lies in the fact that it can display only one answer before it stops, so you have to run the program again and again to allow for different inputs and different answers. This is where an endless loop might be handy. To set up an endless loop, let's get rid of the END statement in line 50 and replace it with:

```
50  GOTO 20
```

The program now contains an endless loop and it will behave very much like the first program that demonstrated an endless loop. However, the screen won't be swamped with a continuously rotating pattern of numbers because of the CLS statements and the INPUT statement, the latter of which halts execution until input is received via the keyboard. Run the program now. As before, you will see the question mark, so type in a number. When you press RETURN, the screen will clear and the answer will be displayed. However, just beneath this answer you will see another question mark. This means that the program has done its job and branched back to line 20. You can now enter another number. You can keep on doing this as long as you want. When you wish to stop the program, simply hold down the ⌘ (flower) key and press the C key simultaneously. This brings about a manual halt.

This simple program is easy to understand, but it still lacks some of the refinements that a computer like the Macintosh is capable of providing. Wouldn't it be nice if the computer could tell us what it was looking for when it reached an INPUT statement? Fortunately, the INPUT statement can be used much like the PRINT statement. Just before execution is halted, a message called a *prompt* or cue may be printed on the screen to tell the user what the computer is looking for. The following program shows how to get a prompt printed.

```
10  CLS
20  INPUT "TYPE IN ANY NUMBER";A
30  CLS
40  X = A * 2
50  PRINT X
60  GOTO 20
```

This is the same program as before, only an instruction or prompt has been included in quotation marks following the INPUT statement line and preceding the designated variable. When line 20 is executed, the message in quotation marks is displayed and then execution is halted awaiting keyboard input. A semicolon is used here to separate the variable from the quoted phrase. The semicolon at this point has the same effect as it does

when used with the PRINT statement. In this case, it means that the value you type in for A will be displayed to the right of the prompt. You may also use a comma in place of the semicolon and the prompt message will be displayed without the question mark. Anytime a quoted phrase is used with an INPUT statement, it is always necessary to separate it from its variable with a semicolon or comma. Also, you must use a variable following the prompt or SYNTAX ERROR will appear on the screen.

The use of the INPUT statement speeds up the running time of a BASIC program. Line 20 could also be replaced with these two lines:

```
20  PRINT "TYPE IN ANY NUMBER"
25  INPUT A
```

This is the equivalent of line 20 in the above program, but here, two lines have been used. The PRINT statement is used to display the prompt, whereas the standard INPUT statement is used in line 25 to read your keyboard input. This is wasteful programming, since the more program lines you have, the more memory it requires for storage and, often, the longer it takes for the program to execute.

Let's expand this program one more time to take care of one little problem that crops up when an endless loop is programmed. You always have to manually halt execution (⌘ and C). The following program contains an exitable loop.

```
10  CLS
20  INPUT "TYPE IN ANY NUMBER";A
25  IF A = 333 THEN END
30  CLS
40  X = A * 2
50  PRINT X
60  GOTO 20
```

Line 25 is the new line. Here, an IF-THEN statement has been used to end the program when A is equal to 333. You will remember that A represents the keyboard input. This program will continue to allow you to input numeric values to be doubled until you type in the number 333. Run the program and type in any number except 333. When you wish to exit the program, simply type in 333 and the program is terminated. It would not be convenient to use a FOR-NEXT loop in this situation, since the programmer would have no way of knowing how many numbers the user might wish to pass through this program. This gives the user the capability of inputting as many numbers as he or she wishes and then bringing about an exit upon keyboard command. This is not a great deal simpler than using the manual halt, but it reflects good programming practice. In such a situation, you might wish to alter the prompt in line 20 to:

```
20  INPUT "TYPE IN ANY NUMBER—TYPE
    333 TO EXIT";A
```

This lets the user know exactly what is expected and how to exit.

Since we already know that variables may be of two types, numeric or string, it is wise to assume that the INPUT statement may be used to accept either type of variable from the keyboard. This is certainly the case, and the following program shows an example of a string variable accepted at the keyboard.

```
10  CLS
20  INPUT "TYPE IN ANY WORD";A$
30  CLS
40  FOR X = 1 TO 5
50  PRINT A$
60  NEXT X
```

This program is used to display any keyboard input five times. While the prompt tells you to type in any word, you may also type in numbers or combina-

tions of letters and numbers. Remember, a string variable will accept almost anything, except a comma. A string variable must not contain commas.

Here's how the program works. Line 10 clears the screen. Line 20 prints the prompt and halts execution until something is input from the keyboard. When you press RETURN, the screen is cleared again and the FOR-NEXT loop is entered at line 40. This loop counts from 1 to 5 and prints the value of A$ each time. When the loop is exited, the program automatically terminates. Run this program and type in any string you wish, as long as it doesn't contain a comma. Your input will be displayed five times on the cleared screen. If you simply press RETURN, nothing is displayed, because A$ will be equal to "", or nothing at all.

The INPUT statement requesting a string variable is often used simply to halt program execution in order to allow on-screen information to be displayed for as long as the user desires. Take, for example, the following program.

```
10  CLS
20  A$ = "HELLO"
30  B$ = "GOODBYE"
40  PRINT A$
50  CLS
60  PRINT B$
```

Run it. The intention here is to display HELLO and then GOODBYE on the screen . . . but not at the same time. This is the reason for the CLS statement in line 50. A$ and B$ are equal to HELLO and GOODBYE, respectively. Line 40 prints HELLO on the screen. We then want to clear this message from the screen (line 50) and print GOODBYE. To a beginning programmer, it may look good in principle, and indeed, the program works just as I've described. However, when you run it, you will immediately see a problem. The computer works so rapidly that you don't really get to see HELLO. As soon as it's printed, the CLS statement in line 50 clears the screen, and GOODBYE is printed. All you really see is GOODBYE. It is in a situation such as this that an INPUT statement can be very useful. It's not that we're looking for any specific information from the keyboard, but rather that we want to halt execution to give the user the choice of deciding when execution should begin again. The END statement wouldn't work here, because that would stop execution altogether and force us to rerun the entire program, only to end up with the same problem.

This situation can be solved very easily by adding a new line to the program.

```
45  INPUT C$
```

Now run the program again. You will now see the word HELLO displayed on the screen. There will also be a question mark below it, indicating the program is waiting for user input. Instead of typing in any characters, simply press RETURN when you're ready to start execution again. Now, line 50 clears the screen and GOODBYE is displayed. You might also include a prompt with the INPUT statement, such as "PRESS RETURN TO CONTINUE," and this is often the case with many programs that include user instructions at the beginning.

INPUT is an extremely useful statement that is incorporated again and again in BASIC programs, that requires or are enhanced by user control.

FORMULAS

Computers are mathematical beasts. In reality, the writing of a computer program involves coming up with a mathematical formula that tell the computer how to do the job. This is what we've been doing throughout this chapter. In some cases, the formulas were quite apparent; in other cases, they weren't. In one example, the keyboard input number was doubled. The formula for doubling any number is X = A * 2, where A is the number to be

doubled and X is assigned the new value. This is a simple formula and one most of us can do in our heads provided the value of A is not a very high or very complex number.

Many formulas for calculating your grocery bill, doing complex trigonometry, or even figuring the size of a sail for a sailboat have been developed and are already available. Electronics is a field where this is most apparent. Committing such applications to the computer is a fairly simple task if you already know the formula. Even highly complex formulas can quickly be put into the computer by translating them into equations the computer can understand.

As an example, let's take the formula $P = I^2R$, used in electronics to figure power in watts. Here *I* stands for current in amperes and *R* stands for resistance in ohms. To write a program that would calculate power (P) in watts based upon current (I) and resistance (R), all we have to do is make the proper assignments to variables and we're home free. Here is a program containing the formula;

```
10  CLS
20  INPUT "CURRENT";I
30  CLS
40  INPUT "RESISTANCE";R
50  P = (I^2)*R
60  PRINT P
```

This program allows the user to enter the value for current and the value for resistance. It will then use these values in the formula contained in line 50 to calculate the power in watts. The formula in line 50 is the same one presented in the discussion. Parentheses are used to indicate that I should be raised to the power of 2 first. The result should then be multiplied by R. Without the parentheses I might be raised to the power of the product of 2 times R.

If you had to work these formulas on paper, it could take some time. But once the computer has the formula properly written into a program, and you have entered variable values the answer is available almost instantly.

The program could also have been written without the need for user input by omitting the INPUT statements and substituting real values for I and R. Line 50 might then read:

```
50  P = (4^2)*5
```

where current is equal to 4 amperes and resistance is 5 ohms.

Surely you must be thinking that anyone who would be working with such a formula would already know that 4 squared is 16 and that 5 times 16 is 80. This is absolutely true. But let's take the same example and let I be equal to 4.22896 and R equal 5.14398. Do *that* one in your head!

BUILT-IN FUNCTIONS

In BASIC, a *function* may be thought of as a type of statement that is really a mathematical formula preprogrammed into the computer and is used to perform operations on numeric and/or string variables. Functions are very powerful tools. Some will allow you to manipulate string variables, while others will be used strictly with numeric variables to return geometric, trigonometric, and other types of mathematical results based on the variable values.

The LEN Function

The LEN function in Microsoft BASIC stands for length. As its name would imply, it will calculate the length of a string, which is calculated by counting the number of characters in the string. For instance, the LEN of BOX would be 3, because BOX contains 5 characters. The following program demonstrates the LEN function.

```
10  A$ = "HELLO"
20  X = LEN(A$)
30  PRINT X
```

The LEN function is used in line 20 and the string variable whose length it is measuring is enclosed in parentheses. In this example, A$ is equal to the word HELLO, which we can easily see contains five characters. In line 20, the numeric variable X will assume the value of the length of A$. The assignment to X is made by the LEN function. Line 30 simply instructs the computer to print the value of X on the screen. When this program is run, the number 5 will be displayed, since this is the number of characters in A$.

One must always remember that to the Macintosh, a space is just as much a character as any other letter, number, or symbol on the keyboard. Therefore, if A$ were equal to "HELLO MOM", the space between the O and the M would also be counted as a character. In this example, X would be equal to 9, because there are 5 characters in HELLO, 3 characters in MOM, and 1 space character between the two words.

The following example shows another way to write a program using the LEN function. This one uses the INPUT statement to read A$ from the keyboard. The rest of the program is nearly identical to the previous one. CLS statements have been added to clear the screen. This program will tell you the length of any group of words, numbers, or characters you input via the keyboard.

```
10  CLS
20  INPUT A$
30  CLS
40  X = LEN(A$)
50  PRINT X
```

When this program is run, the screen will be cleared and you will see the question mark prompt. You can type in anything at this point. When you press RETURN, the screen will be cleared again and the number of characters input will be printed on the screen.

The program below is set up with an exitable loop, which allows you to continue to input data from the keyboard. When you wish to exit the program, all you have to do is press RETURN without typing a word.

```
10  CLS
20  INPUT "TYPE IN ANY PHRASE TO BE
    MEASURED";A$
30  CLS
40  X = LEN(A$)
50  IF X = 0 THEN END
60  PRINT X
70  GOTO 20
```

Line 10 clears the screen and line 20 allows you to input the characters to be represented by A$. A prompt is included with the INPUT statement to instruct the user as to what is expected. Line 30 clears the screen again, and line 40 uses the LEN function to assign X the value of the number of characters in A$. Line 50 contains the exit routine. It simply says if the value of X is zero (no characters input) then end the program. Line 60 prints the value of X on the screen (assuming X is not equal to zero). Line 70 uses the GOTO statement to branch back to line 20, where the INPUT statement is again encountered. Remember, the RETURN key does not count as a character and must be pressed every time you enter any numbers, characters, or symbols from the keyboard.

It may be difficult for you to imagine any useful purpose for the LEN function in a practical program, but as your programming experience increases, you will find that you will be using it more and more. Some possible uses include a typing test program. Here, the LEN function would indicate the number of characters typed. By knowing the number of characters input and the time it took to

input them, typing speed can be calculated quickly. The LEN function is also useful in some types of programs as a check to make certain the proper user input has been supplied. For example, a game program might require the user to input a word that is no longer than 5 characters. The LEN function can be used to test the length of the user's input an error message can be displayed if the input exceeds the specified length.

The INT Function

The INT function stands for integer. It is used only with numeric variables and gives the integer equivalent of this variable. An integer is simply a whole number (positive or negative). The numbers 3, 5, 6, 9, and 10 are integers, whereas 3.5, 9.3, and 10.1 are not. The INT function simply *truncates* or lops off the decimal portion of a number. The number 0 is also an integer. Therefore, the integer equivalent of the number 0.003 is 0. Remember, the INT function lops off all numbers past the decimal point. The following program demonstrates the use of the INT function.

```
10  A = 10.5
20  X = INT(A)
30  PRINT X
```

When this program is run, the computer screen will display the number 10, which is the integer equivalent of the real number 10.5.

The INT function does *not* round numbers. We might modify this by saying that the INT function always rounds down. It will always return the largest integer that is less than or equal to A in the above program. This seems fairly simple until one starts dealing with negative numbers. If A were equal to -3.6, the INT function would return -3 for the value of X. Remember, the INT function always returns the number that is less than or equal to A. You might expect that the INT of -3.6 would be -3 until one stops to think that -3 is a larger number than -3.6. The INT function will return -4 as the integer of -3.6.

The RND Function

The RND function stands for random. This function is almost always used to program games of chance on the Macintosh. You can think of RND as a variable that is equal to any value between 0 and 1, not including either 0 or 1. Therefore, RND will always be equal to a value that is more than 0 and less than 1.

The RND function works in close conjunction with a RANDOMIZE statement. RANDOMIZE simply shuffles the random number generator. RND picks the random number that is output. Whenever you use RND in a program, you must use the RANDOMIZE statement at the beginning of the program to reshuffle the random number generator. If you don't do this, the numbers output from RND will follow the same pattern during every program run. Between RANDOMIZE and RND, you can generate what can be classified as random numbers. From a technical standpoint, these numbers are not really chosen at random like drawing a number out of a hat. The computer actually follows a mathematical formula to make its selection, but it is one that is too complex for most of us to anticipate. This form is technically known as pseudo-random number generation.The program below will demonstrate the RANDOMIZE statement and the RND function.

```
10  RANDOMIZE
20  FOR X = 1 TO 10
30  PRINT RND
40  NEXT X
```

When this program is run, the computer will generate an automatic prompt due to the use of the RANDOMIZE statement in line 10. The prompt will read RANDOM NUMBER SEED (−32768 to 32767)? The computer is telling you that it wants

you to type in a number between −32768 and +32767 to be used in the randomizing formula. After you've typed in this number, simply press RETURN and the screen will then display 10 random numbers. Run the program again and type in a different number in response to the prompt. The random numbers will appear again, but this time they will be completely different because of the new seed number.

The RND function is rarely used by itself as it is in line 30. Most often, it becomes a multiplier. A simple game of flipping a coin can be programmed to illustrate the use of RND as a multiplier. Since the Macintosh may be used to play games, we will spend some time on this subject. First, let's think of the possibilities that occur when a coin is flipped. From a practical standpoint, only one of two possibilities can occur. The coin can either land with heads up or tails up. Certainly, there is a billion to one chance that the coin might land on its edge and stay there, but the chance is so slim that it need not be taken into account. Since we have two possibilities, we want the computer to output one of two possible numbers at random. One number would represent heads, while the other would represent tails. We want to make sure that only two numbers are possible.

```
10  CLS
20  RANDOMIZE
30  CLS
40  X = RND * 2
50  INPUT "PRESS RETURN TO FLIP
    COIN";A$
60  PRINT X
70  GOTO 40
```

This program is a good start for a coin flipping game. The RANDOMIZE statement is used in line 20 to allow you to input a random seed number. In line 40, the numeric variable X is equal to RND times 2. The number 2 was chosen because we're looking for two possibilities, heads or tails. We'll let the number 1 represent heads and 2 tails. The INPUT statement in line 50 is used to give us some control over when the coin is flipped. Each time RETURN is pressed, the value of X, which represents the flip, will be printed on the screen. The GOTO statement in line 70 branches back to line 40, where X is assigned another random number. Each time the RND function is executed, a different random number will be output.

Run this program and press RETURN five or six times. You will immediately notice that at no time does the number 1 or 2 appear on the screen. What you have is a series of numbers with long decimals. However, if you look closely you will see that all of these numbers fall into two categories. One group is equal to more than 1, while the other group is less than 1. The computer is on the right track. It does provide us with two possibilities, numbers more than 1 and numbers less than 1. We have a start.

Of course, our goal is to write a program that will output either a 1 or a 2. Think back to the previous function that was discussed. The INT function will always return a number that is a whole number. Since the numbers 1 and 2 are indeed integers, we might well be able to use INT in our coin flip program. Change line 40 to:

```
40  X = INT(RND*2)
```

Now, the program will always output an integer. Run it again and press RETURN several times to flip the computerized coin. We're getting very close now, because only two numbers are being output, although they are 0 and 1 rather than 1 and 2. This would probably suffice if we let 0 equal heads and 1 equal tails. However, this was not the original intent of the program. We want the numbers 1 and 2 to be output. The purpose of this will become apparent shortly. What do we do?

The answer is simple. By adding 1 to each of

the numbers being output, we will always arrive at one of two possible numbers, 1 and 2. That's our goal! Change line 40 to:

```
40  X = INT(RND*2)+1
```

Make sure the +1 appears outside of the parentheses. Now run the program again. Eureka! We have arrived at our goal. The computer is now outputting the numbers 1 and 2 as originally planned. Let's go one step further and make the game more lifelike using IF-THEN statements. Let's change line 60 completely and add another line:

```
60  IF X = 1 THEN PRINT "HEADS"
65  IF X = 2 THEN PRINT "TAILS"
```

Run the program with these revisions, and you will see that our coin flip game is not complete. Congratulations! You have just written your first game program! It wasn't all that difficult, was it?

Now, as for my emphasis on tailoring this program to output the numbers 1 and 2 to represent heads and tails, let's see why this may be important for future programs. Some games of chance use devices that normally output numbers rather than symbols. The first thing that comes to mind is a dice game. Each die contains numbers from 1 to 6 on its sides. In the coin flip game, we really could stay pretty much with the starter program that simply output decimal numbers that were either less than or more than 1. We could have used IF-THEN statements to tell the computer to print heads if X was less than 1 or tails if X was more than 1. We cannot do this with dice. We have to have whole numbers that range from 1 to 6.

We will now write a simple dice program using a computer version of one die. In computer terms, this will be a random numbers program that will output whole numbers between 1 and 6. You will remember that we used two numbers in the coin flipping program to be multiplied by RND. This number was chosen because we were looking for two possible conditions. In a dice game, we are looking for 6 possible conditions, so let's replace the 2 with a 6. The finished program is shown next.

```
10  CLS
20  RANDOMIZE
30  X = INT(RND*6)+1
40  INPUT "PRESS RETURN TO ROLL THE
    DIE";A$
50  PRINT X
60  GOTO 30
```

Hey! This program is less complex than the finished coin flipping program! Why? We don't have to convert the computer's numbers into words such as heads or tails. Run the program. After you've typed the seed number, you will be greeted with a prompt, and each time you press RETURN, a number ranging from 1 to 6 will appear on the screen.

Certainly, you might complain that only one die is incorporated. The simple fact is that most of the work has already been done, and to add another die, it's only necessary to modify one line in the original program and add one more. Change line 60 to

```
60  PRINT X;Y
```

and add this line:

```
45  Y = INT(RND*6)+1
```

Remember, each time the RND function line is executed, a different random number is produced. If this number is not significantly different from the previous random number returned, the same integer will result. For instance, if the first random number returned is .6, then .6 times 6 equals 3.6, the integer displayed will be 3. If the next random number is .5, then .5 times 6 is 3.0, the integer of

which is still 3. By adding another line (45), we have inserted another RND function line and assigned Y the output of this line. It is possible for both X and Y to be equal, just as you can throw a double with real dice. It is even more possible for the two to be unequal. Run the program and you will see that sometimes you get a double, and sometimes you don't.

Let's add another line to make the program even more interesting.

```
55 IF X = Y THEN PRINT "DOUBLE"
```

The IF-THEN statement checks for a condition of X being equal to Y. If this is true, this same line will print the word "DOUBLE" on the screen. Add this line:

```
56 IF X = 1 AND Y = 1 THEN PRINT "SNAKE
   EYES"
```

The IF part of this IF-THEN statement checks to see whether or not both X and Y equal 1. Because this condition is known as "snake eyes" in some dice games, a message indicating this is displayed, in addition to the DOUBLES message.

Line 56 contains something new. It contains the word AND. AND is called a logical operator. Sometimes, you will also see an OR in a similar statement. This is very simple to understand. AND and OR combine two IF-THEN statements. A condition is true only if the values on both sides of the AND are true. In this case, if X is equal to 1 and Y is equal to 1, then do whatever follows the THEN statement. With this in mind, you should now be able to insert your own program line (57) that will test for the condition of "box cars" (both die equal to 6).

The CINT Function

The CINT function is like the INT function in that it will always return an integer; however, CINT performs this conversion process by rounding up or down, depending on the decimal value. Any fractional value of 0.5 or higher will cause CINT to round up. Any value lower than 0.5 will be rounded down. The following program demonstrates the CINT function.

```
10 X = 24.3
20 Y = 24.6
30 A = CINT(X)
40 B = CINT(Y)
50 PRINT A
60 PRINT B
```

When this program is run, the numbers 24 and 25 will appear on the screen. The variable A represents the CINT value of 24.3. Since .3 is less than .5, CINT rounds down. Variable B represents the CINT value of 24.6. Since .6 is more than .5, CINT rounds up.

CINT is often used in place of INT when performing calculations with monetary values. Traditionally, a value of .5 or higher causes a rounding up to the next higher cent. Likewise, if the value is less than .5, rounding is to the next lower cent.

MORE STRING FUNCTIONS

In Microsoft BASIC, there are several powerful functions that are dedicated to the handling of string variables. These are used heavily in word processing programs and also crop up quite often in many other types of programs. These are not terribly difficult to understand if you take them a step at a time. Through the step-by-step process of learning a new language, it is easy to develop a program with a high degree of efficiency in a short period of time.

LEFT$

The LEFT$ function is used to pull a sequence

of characters from a string starting with the lefthand side of that string. The following program illustrates its use.

```
10  A$ = "HELLO"
20  X$ = LEFT$(A$,2)
30  PRINT X$
```

This program assigns the word HELLO to the string variable A$. Line 20 assigns a value dictated by the LEFT$ function to the string variable X$. Line 30 prints the value of X$ on the screen. Run this program. You will see that the screen displays HE. The reason for this becomes evident if you look at line 20. This line tells the computer to assign to the string variable X$ 2 characters from A$. It assigns the *first* 2 characters because LEFT$ always starts counting from the left and counts the number of places specified by the number that follows. In this case, line 20 tells the computer to count 2 characters in from the left of A$. The first 2 characters, starting at the left of word HELLO, are HE. Therefore, X$ becomes equal to HE. If you change the 2 in line 20 to 1, then only an H will be printed, because this is the first character from the left in A$. Change this number to 4 and see what happens. The LEFT$ function is very useful if you wish to retrieve only a portion of a string. Remember, LEFT$ will search the number of places specified from the left of the string.

RIGHT$

The RIGHT$ function works just like LEFT$, except it starts searching from the right end of the string, as the following program demonstrates.

```
10  A$ = "HELLO"
20  X$ = RIGHT$(A$,3)
30  PRINT X$
```

When this program is run, the computer will display LLO, because these letters are the 3 rightmost characters in A$. It's hard to generate an error message with either of these two functions. If you specify more letters than are contained in the string, the computer simply assigns the entire string to the variable, which would be like saying

```
X$ = A$
```

in line 20.

MID$

While LEFT$ and RIGHT$ are very useful functions, they always force us to accept all the characters in a sequence from the left end of the word or the right end of the word. We cannot go into a string and pull out a middle section using LEFT$ or RIGHT$. So, Microsoft BASIC also provides MID$, so we can do just that. It works very much like LEFT$ and RIGHT$, except we have to give it 2 numbers, one to indicate the position within the string to start its search at and another to end its search. All numbers are given in relation to the left of the string. The following program demonstrates the use of MID$.

```
10  A$ = "HELLO"
20  X$ = MID$(A$,2,3)
30  PRINT X$
```

When this program is run, the computer will display the value of X$ as ELL. This is because we told the computer to search the string starting with the second character and ending 3 characters after that. MID$ does not count 3 characters after the second character, but 3 characters total, including the second character. With the MID$ function, we can easily pull out any portion of a string.

TIME$

Your Macintosh has a built-in clock that is

battery-operated when normal ac power is removed. The TIME$ function returns the current time maintained by this internal clock. Once the time is initially set, the Macintosh will automatically update it on a second-by-second basis. Let's set it with the following program.

```
10  X$ = "09:36:22"
20  TIME$ = X$
30  PRINT TIME$
```

Line 20 could also have been written as TIME$ = "09:36:22", so we could delete line 10. Either method will work, since X$ represents the value to be assigned to TIME$. When you run this program, nothing spectacular happens. The value of X$ will be printed on the screen. Now, wait a few seconds and in direct mode (without a line number) type PRINT TIME$. When you press RETURN, the value of TIME$ will be displayed on the screen, but note that TIME$ is no longer equal to the value that reflects the seconds that have passed since the program was run. The clock is now keeping time.

The following program will allow you to set the clock to the current time and then see the clock constantly displayed at the center of the screen.

```
10  CLS
20  INPUT "CURRENT TIME IN HOURS,
    MINUTES, AND SECONDS";X$
30  TIME$ = X$
40  CLS
50  PRINT TIME$
60  GOTO 40
```

The input statement in line 20 allows you to enter the current time. Simply look at your clock. Your input is assigned to the string variable X$. In line 30, the TIME$ function is initially assigned the value of X$. The screen is then cleared, and the value of TIME$ is displayed on the screen. The GOTO statement in line 60 branches back to line 40, where the screen is cleared and TIME$ is printed again. Every time TIME$ is displayed on the screen, it will be the updated version, which is controlled by the Macintosh clock. However, TIME$ is technically updated only once every second, at least as far as we are concerned. This program will cause the TIME$ value to be displayed several times per second. Fast computer operations are desirable in many programs, but here, we have to do something to slow the computer down to avoid screen flicker. Insert the following line:

```
55  FOR X = 0 TO 1000:NEXT X
```

This is a delay loop that simply directs the computer to count from 0 to 1000 before executing the GOTO statement in line 60. This avoids the unwanted flicker and gives an excellent clock display.

ANOTHER BRANCH STATEMENT

Early in this chapter, the GOTO statement was discussed. You will recall that it is a branch statement that tells the computer where to go or which branch to take within the program. Branch statements are useful, because they allow us to use previously written sections of the program to perform the same task again—saving programming time.

There is another branch statement in Microsoft BASIC that is extremely useful and may be thought of as a GOTO statement that can go to a line, execute lines, and then return to where it left off. It's called GOSUB, or GOSUB-RETURN. GOSUB and RETURN are keywords that work together, just as IF and THEN are.

GOSUB stands for go to subroutine. A subroutine is like a miniature program in itself. Like GOTO, GOSUB is used with a line number. This tells the computer where to begin the subroutine. However, in BASIC, a subroutine accessed by a GOSUB is always ended with RETURN. RETURN tells the computer to go back to GOSUB and exe-

cute the next consecutive line. The program demonstrates the use of these statements.

```
10  CLS
20  INPUT "TYPE IN ANY NUMBER TO BE
    DOUBLED";X
30  CLS
40  GOSUB 70
50  PRINT Y
60  END
70  Y = X * 2
80  RETURN
```

While this program in no way makes completely adequate use of GOSUB and RETURN, it does effectively demonstrate their use. Lines 10 through 20 do some familiar things. The GOSUB statement is found in line 40. It tells the computer to go to a subroutine that begins at program line 70. Skip down to line 70. This is the start of the subroutine. It assigns to Y the value of X times 2. Line 80 ends the subroutine with a RETURN statement.

The RETURN statement tells the computer to go back to the line following the line containing the GOSUB statement. In this case, the GOSUB statement is found on line 40, so the RETURN statement sends the computer to line 50. Line 50 displays the value of Y (2 times X) on the screen, and the program ends in line 60. You will recall that this statement was not needed in our other programs, but in this case it is required. If you omitted the END statement in line 60, the computer would automatically go on and execute the remaining lines, which we do not want it to do.

As a test, let's remove the END statement in line 60 altogether. In Microsoft BASIC, this is a simple task. In direct mode, simply type 60 and hit RETURN. You can list your program at this point and you will see that line 60 has been deleted. Run the program again. You can still input a value and the GOSUB will still access the subroutine. The RETURN statement in line 80 sends the computer back to line 50, where the value of Y is printed. However, you are then faced with an error message. The computer is telling you that it encountered a RETURN without GOSUB. Here's what happened. Since the END statement was omitted, after line 50 the computer executed line 70. The next line executed was 80, and here's where the computer got distressed. It read a RETURN statement and said, "I'm not supposed to be here because a GOSUB hasn't instructed me to be here." Thus, the error message was returned. Anytime the computer encounters a RETURN, and GOSUB did not send it there, you will receive an error message. It worked find during the first entry to the subroutine, because the GOSUB in line 40 told it to be there. But when it got there the second time, it had no GOSUB instruction and told you so through the error message. This demonstrates the fact that the END statement is crucial to the proper operation of this program.

READ/DATA STATEMENTS

A very useful pair of statements in BASIC are READ and DATA. DATA statements contain what are often called data elements. The DATA statement's sole purpose, is simply to hold elements. The READ statement pulls items from the DATA statement lines, in sequential order. This means that the first item is read first, the second is read second, the third is read third, and so on. The following program demonstrates the use of READ and DATA statements.

```
10  CLS
20  INPUT "PRESS RETURN TO READ A
    DATA ELEMENT";A$
30  CLS
40  READ X
50  PRINT X
60  GOTO 10
70  DATA 14,28,32,64,100
```

Run the program. Each time you press RETURN in response to the prompt, a DATA element will be displayed on the screen. The data element is always displayed on an otherwise blank screen because of the CLS statement in line 30. Notice that each time you press RETURN, the next data element is read (line 40) and displayed (line 50). The numbers themselves are contained in the DATA statement lines. After one item is read, it is not read again. On the next pass, the next sequential DATA item is read and displayed.

If you keep going, an error message will appear. This occurs after the last data element was read. The OUT OF DATA IN LINE 40 error message indicates that all of the data items have been read, but the READ statement in line 40 is still looking for more. Often, READ statements are placed in FOR-NEXT loops to eliminate such errors. But you must make certain the number of data elements equals the number of passes through the loop or you can still run out of data.

We can correct this error message situation in the sample program by using a new statement. Add the following line:

```
55  IF X = 100 THEN RESTORE
```

The RESTORE statement tells the READ statement to go back to the first element in the DATA statement line. If you think of the READ statement as counting through DATA elements, you can think of RESTORE as resetting this count to 1. Run the program again. You can press RETURN as long as you wish, and you will get no error message. When the last number in the DATA statement line has been read, the first number will come up again on the next pass. The READ statement will then go through all the remaining numbers before reverting back to the first position again. If you want the program to end as soon as the last DATA element is read, change line 55 to:

```
55  IF X = 100 THEN END
```

All we're doing here is setting up an IF-THEN statement to terminate the program after last element in the DATA statement is read. This is an indication that there are no more DATA elements and the program terminates.

You will notice here that the variable following the READ statement is a numeric type, since it contains no dollar sign ($). This is an indication that the DATA statement line contains only numeric values or numbers. However, READ/DATA can also be used to store and access string information, as evidenced by the following program.

```
10  CLS
20  INPUT "PRESS RETURN TO READ A
    DATA ELEMENT";A$
30  CLS
40  READ X$
50  PRINT X$
60  IF X$ = "GOODBYE" THEN END
70  GOTO 20
80  DATA HELLO,HOW,ARE,YOU,GOODBYE
```

Run this program and you will see the words displayed on the screen exactly as the numbers were. The READ/DATA statement combination stores values in DATA statement lines. When a data element is accessed using READ, the variable following the word READ is assigned the value of the data element.

MULTIPLE STATEMENT LINES

All programs so far have used only one statement per program line, but it is quite easy to include several statements on a single program line. Take the following program for example.

```
10  CLS
20  X=10
```

```
30  Y=20
40  Z=X*Y
50  PRINT Z
```

It can be rewritten as follows:

```
10  CLS
20  X=10:Y=20:Z=X*X
30  PRINT Z
```

Here, line 20 has been used to include the assignments that were in lines 20 through 40 before. Notice that a colon is used to separate each of the assignments. The computer runs this program exactly the same way as it ran the previous program. We could even go one step further and write the program as one single line, as shown below.

```
10  CLS:X=10:Y=20:Z=X*Y:PRINT Z
```

Notice that colons separate each individual statement on the line. If you omit a colon, you will receive an error message. You can use as many as 255 characters per line. If you use more than 255, the computer automatically knocks off the extra characters.

For the sake of clear programming, you should not often put a tremendous number of statements on any one line. There is an advantage to using as few lines as possible from a memory saving standpoint, because each new line number takes up additional memory. The last example required slightly less memory storage than the first. Sometimes, however, clear programming can better be adversely affected by committing a group of statements or assignments to a single program line. When line 20 contained all assignments to X, Y, and Z, at least they all had something in common, so could be placed on the same line for that reason. However, with all statements on one line, it is difficult to read the program line.

LOGICAL OPERATORS

Logical operators perform special operations with value. We will be concerned with AND and OR here. We often use these with IF-THEN statements to bring about special branches. The following program is a good example of the use of AND.

```
10  CLS
20  INPUT "TYPE ANY NUMBER FROM 0 TO
    10";X
30  INPUT "TYPE ANOTHER NUMBER FROM
    11 TO 20";Y
40  IF X=5 AND Y=18 THEN PRINT "THOSE
    WERE THE NUMBERS I WAS LOOKING
    FOR"
50  END
```

Line 40 tells the computer to print the quoted phrase only when X is equal to 5 and Y is equal to 18. If X is equal to 5 and Y is equal to a number other than 18, the phrase will not be printed. If X is not equal to 5, the phrase will not be printed even if Y is equal to 18. When the AND operator is used, both conditions must be true. In this case, X must be equal to 5 and Y must be equal to 18 before the phrase will be printed.

You can go further by adding a few more INPUT statement lines similar to those in lines 20 and 30. Line 40 might read:

```
40  IF X=5 AND Y=18 AND Z=40 AND ZZ=55
    THEN PRINT "THOSE WERE THE NUM-
    BERS I WAS LOOKING FOR"
```

Now let's discuss the OR operator. Change line 40 in the original program to:

```
40  IF X=5 OR Y=18 THEN PRINT "BINGO"
```

Here, line 40 tells the computer to print BINGO if X is equal to 5 *or* Y is equal to 18. If X is not equal to 5,

but Y is equal to 18, the word will be printed. As long as X is equal to 5 or Y is equal to 18, the word will be displayed. Only one of the tests within a logical OR operation must be true to bring about the proper result. If both are true, that's fine as well.

You can also use AND and OR together, as the following program demonstrates.

```
10  CLS
20  INPUT "TYPE ANY NUMBER FROM 0 TO
    10":X
30  INPUT "TYPE ANY NUMBER FROM 11 TO
    20";Y
40  IF X=5 AND Y=15 OR X=3 THEN PRINT
    "BINGO"
```

Look at the condition set up in line 40. If X is equal to 5 and Y is equal to 15, then the computer will print BINGO. However, if X is not equal to 5 and Y is not equal to 15, but X is equal to 3, the word will also be printed. You can think of this as two IF-THEN tests on the same line. The first tells the computer to print the word if X is equal to 5 and Y is equal to 15. The seconds tells the computer to print the word if X is equal to 3.

AND and OR can also be used with string variables.

RELATIONAL OPERATORS

A relational operator is a symbol that causes the computer to compare two values. We are already familiar with one of these relational operators, the equal sign (=). However, there are other relational operators as well. The inequality sign is used to state that two values are not equal. It looks like this:

< >

While the computer treats this like a single symbol, you must use the comma and period keys in uppercase to type these two separate characters. The inequality sign is used just like the equal sign; like this:

A<>B

There are two other relational operators. These are the greater than and less than operators shown below.

< Less than
> Greater than

Here is the format in which they are used:

A < B A is less than B
A > B A is greater than B

Just think of the wide side of the lopsided V as being the more than side and the pointed side as pointing to the variable that is less than the other one.

Now, we can combine the less than/greater than symbols and the equal sign, as below:

A <= B A is less than or equal to B
A >= B A is greater than or equal to B

We can program with the relational operators to catch user input errors. The program below needs a number of from 0 to 10. Line 30 makes sure that the number input is not less than 0.

```
10  INPUT "TYPE ANY NUMBER FROM 0 TO
    10";X
20  IF X < 0 THEN PRINT "INVALID NUM-
    BER":GOTO 10
```

Line 20 says if X is less than 0, then print INVALID NUMBER and ask the question again.

We might also wish to make sure that the input is not larger than 10. The proceeding program illustrates such an *error trap*.

```
10  INPUT "TYPE ANY NUMBER FROM 0 TO
    10";X
20  IF X > 10 THEN PRINT "INVALID
    NUMBER": GO
```

Line 20 tells the computer to print the phrase if X is greater than 10.

We don't need two programs lines to do all of this if we remember the previous discussion on logical operators. This program demonstrates the combined use of logical and relational operators.

```
10  INPUT "TYPE ANY NUMBER FROM 0 TO
    10";X
20  IF X< 0 OR X >10 THEN PRINT "INVALID
    NUMBER"
```

Line 20 tells the computer to print the phrase if X is less than 0 or greater than 10.

The program below demonstrates the use of the less than or equal to operator:

```
10  INPUT "TYPE ANY NUMBER FROM 1 TO
    10";X
20  IF X <= 0 THEN PRINT "INVALID
    NUMBER"
```

Line 20 tells the computer to print the phrase if X is less than or equal to 0. Note that in line 10 we are asking for a number from 1 to 10. We could also have handled this by specifying that the phrase is to be printed if X is less than 1.

Some of the chapters that follow this one will present program listings that make heavy use of relational and logical operators, so be sure to re-read this material that you're not quite clear on it. The logical and relational operators allow us to make more efficient use of programming space and will be used often.

ARRAYS

An array is a group of values set up as a table. All of the values are contained within the array are in sequential numeric order. If the array is named A, then A(0) contains one value, A(1) contains another, A(2) yet another, and so on.

To set up an array, we use the DIM or *dimension* statements. This determines the size of the array, or more appropriately, the number of values it can contain. The following shows the format for DIM:

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

The DIM statement sets up an array named A that can hold 11 items. Why 11? Because the first element in the array will be specified as A(0). You must remember that 0 is a valid number, and if you count from 0 to 10 on your fingers, you will run out of fingers before you get to 10. You can simply remember that an array can hold one more element than the DIM statement value. The following program demonstrates some of the working of an array.

```
10  CLS
20  DIM A(5)
30  A(0) = 10
40  A(1) = 20
50  A(2) = 30
60  A(3) = 40
70  A(4) = 50
80  A(5) = 60
90  PRINT A(2)
```

Here, the array was Dimensioned to contain 6 elements. Lines 30 through 80 assign numbers to the various array positions. You will notice that we treat A(0) or any other element in the array as another variable. Line 90 tells the computer to print the value of the element contained at A(2).

This may seem very awkward, and you may wonder why you couldn't use standard variables here instead of an array. This is a good point, but

remember that this is a demonstration program to familiarize you with arrays. The next program puts the array to more effective use.

```
10  CLS
20  DIM A(5)
30  FOR X = 0 TO 5
40  A(X) = (10 * X) + 10
50  NEXT X
60  PRINT A(2)
```

This program fills the array with the same values as before, only it does it much faster with a FOR-NEXT loop. When you run the program, the screen will still display 30, the value assigned to A(2). The value of X is substituted for the element number. In line 30, the FOR-NEXT loop assigns the variable X a value of from 0 to 5. On the first pass of the loop, A(X) in line 40 is really equal to A(0). On the second pass of the loop, it is equal to A(1). When the loop times out, it's equal to A(5). Line 40 assigns A(X) a value that is equal to ten times X plus the number 10. On the first pass, X is equal to 0, and ten times 0 is still 0. But 0 plus 10 is equal to 10. Therefore, A(X), or A(0) on this pass, is equal to 10. On the next pass, A(X) of A(1) is equal to one times 10 plus 10, or 20.

It is not possible to easily assign a standard numeric variable in this manner. We can only do this with an array, where we can substitute a variable for the element number in the array.

The arrays discussed thus far are called *numeric* arrays. We can also use string arrays that work similarly. To designate a string array, we would use this format:

```
30  DIM A$(5)
```

This establishes a string array named A$ which will hold 5 elements. Assignments can be made just as they are in the following program:

```
31  A$(0) = "HELLO"
32  A$(1) = "GOODBYE"
33  A$(2) = "HI"
```

and so on.

The following program uses a FOR-NEXT loop with an INPUT statement in the loop to assign words or phrases to an array. The latter part of the program reprints these words or phrases:

```
 10  CLS
 20  DIM A$(5)
 30  FOR X = 0 TO 5
 40  INPUT "TYPE IN ANY WORD";W$
 50  A$(X) = W$
 60  NEXT X
 70  CLS
 80  FOR X = 0 TO 5
 90  PRINT A$(X)
100  NEXT X
```

When this program is run, you will be presented with the input phrase "TYPE IN ANY WORD." When you type in the first word and press RETURN, this word is assigned to W$. Line 50 assigns the first value to the first A$(X), A$(0). The loop recycles and you are again prompted to type in any word. Your next word is stored in W$ and it, in turn, is committed to A$(1) in line 50. This occurs during each of the six cycles of the loop. Line 70 clears the screen.

Lines 80 through 100 display the contents of the array A$ on the screen. The loop in line 80 counts from 0 to 5 and line 100 prints the contents of A$(X) on the screen. Remember, X will again be cycled from 0 to 5, and each of these elements contains a different value, assuming you typed in a different word each time you were prompted to do so. The element position is always an integer (no decimals!) and may be represented by a numeric variable even though the array itself may contain string values.

SUMMARY

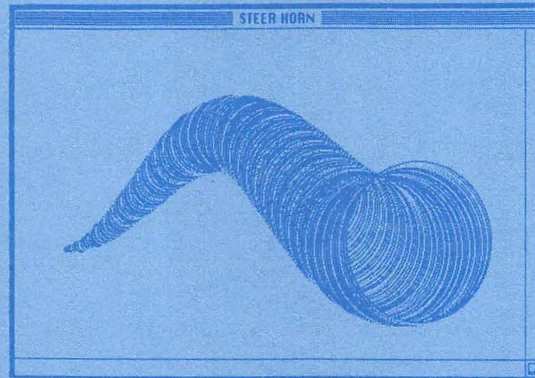
This chapter has dealt with some of the Microsoft BASIC language. Most of the functions and statements discussed will be used many times in almost every BASIC program you write. This discussion has by no means explored the full extent of this powerful language, since to do so would require many volumes. This chapter will provide the beginner with a tool for getting started in Microsoft BASIC. From this point on, the reference manual included with your Microsoft BASIC software package can get you over the rough spots.

If you're not clear on the use of the keywords presented here, do yourself a favor and reread the information. It is mandatory that you understand those portions of the BASIC language before moving further in this text.

Learning any new language, be it Russian, French, German, or BASIC, involves taking it a word at a time. When you understand the meaning or purpose of a single word fully, it is far easier to move on to a new word, because you can then see how it relates to the one you already know.

I urge all readers to rewrite each of the working programs presented in this chapter, using your own imagination to alter them. Try a few unusual routines and see how the computer responds. You can't harm the computer this way, and you could find you're on your way to becoming a real programmer.

Chapter 7



Programming Graphics in BASIC

The Macintosh can be quite accurately described as an excellent graphics computer. The graphics capabilities of Microsoft BASIC are extremely powerful. A single statement may do what a lengthy program does in other dialects of BASIC. The major graphics statements include CIRCLE, LINE, PSET, PRESET, PUT, GET, and several others. There is also a graphics function called POINT. In addition to all that is offered by Microsoft BASIC, the Macintosh contains an extensive set of sophisticated ROM graphics functions that can be called from BASIC.

THE GRAPHICS SCREEN

Unlike the Apple II computers, the Macintosh contains only one screen format. The others offer separate screens for graphics and text. Also the Macintosh screen is monochrome or displays only one color against its background, usually black on a white background. The entire screen is composed of 512 vertical columns and 342 horizontal rows. This means that the screen can show 512 individual points from left to right and 342 points from top to bottom. However, we cannot use the entire screen in Microsoft BASIC, because the built-in screen formatting uses a portion of space to display various icons and windows. I have found that the usable viewing area of the graphics screen when operating under Microsoft BASIC is 496 columns by 294 rows (Fig. 7-1).

You may locate points at the viewable screen coordinates. If you specify a point greater than 495 or 293, this point will lie somewhere outside the user-viewing area. As is usually the case with computer screen specifications, we begin numbering rows and columns with 0. So the column positions are numbered 0 to 495 for a total of 496 points, and the rows are numbered 0 to 293 for a total of 294 points.

When specifying coordinates, notice that the column designations are viewed from left to right across the screen, whereas the row designations

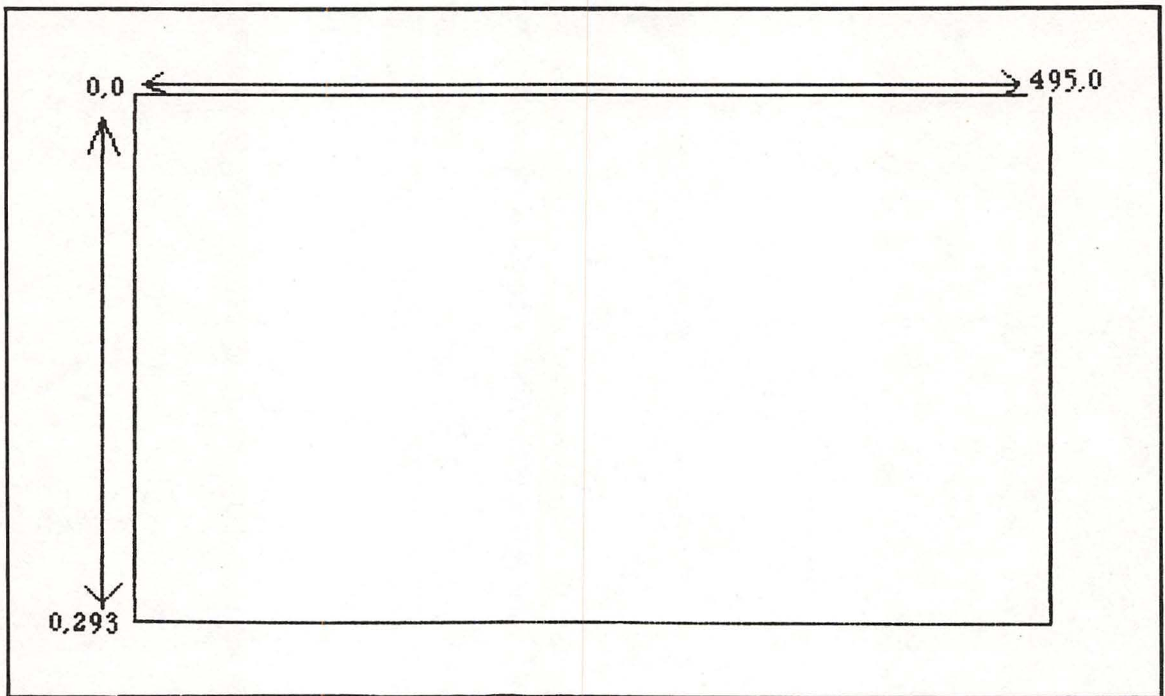


Fig. 7-1. The Viewable Macintosh Screen Consists of 496 Horizontal Points Numbered 0 to 495, and 294 Vertical Points Numbered 0 to 293.

are viewed from top to bottom down the screen. In a coordinate specification the column position is always named first, the row position second. A coordinate of 0,0 specifies a point on the screen that lies in the extreme upper left-hand corner. A column position of 0,293 specifies a screen point which lies at the extreme lower left-hand corner. By changing the row coordinate we can move the screen access point up or down. By changing the column coordinate we can move the access point left or right. The coordinates 495,0 access a point in the extreme right-hand corner of the screen, and 495,293 coordinates access a point in the extreme right-hand corner. The exact center of the viewable screen is arrived at by dividing the maximum coordinates by 2. Since thre are 496 columns and 294 rows, dividing each by 2 will give the coordinates 248,147. It is good to know where the center of the screen is, because many graphic writes will begin there.

It takes a while to get accustomed to graphics screen coordinates, but you will develop a "feel" for where coordinates are with a bit of practice.

The column coordinate will be referred to as the X coordinate; the row coordinate will be referred to as the Y coordinate. The naming of coordinates is extremely important when using most graphics statements in Microsoft BASIC. In the following discussions, you will see just how these coordinates work.

CIRCLE STATEMENTS

The CIRCLE statement in Microsoft BASIC allows us to draw circles on the screen using a single program line. The format for using this statement is shown here:

```
CIRCLE(X,Y),R
```

Here, X and Y are the coordinates, and R is the radius of the circle in screen points. The following program will demonstrate the use of the CIRCLE statement:

```
10  CLS
20  CIRCLE(248,147),60
```

This program will draw a circle with a radius of 60 points. These points are often referred to as *pixels* when describing the graphics screen. We can say that a monitor screen is composed of so many horizontal and vertical pixels. For the sake of this discussion, pixels and points are identical.

Here's how the program works. Line 10 uses the CLS statement to clear the screen. The CIRCLE statement in line 20 needs more explanation. In parentheses are the screen coordinates, which in this case, specify the center of the circle. This is where the graphic is to begin. No point is actually plotted at its center. This specifies a starting point for the plotting of the circle. The radius designation is 60. Therefore, the CIRCLE statement will draw a perfect circle on the screen 60 points from the X,Y coordinates. When you run the program, you will immediately see a large circle at the center of the screen. Now, change line 20 to:

```
20  CIRCLE(140,147),60
```

When you run this program, the same circle appears, but it has been shifted to the left. This is because of the changed X,Y coordinates. Instead of specifying the circle's center as the actual center of the screen, we have specified a point to the left of center. The horizontal position is now at X coordinate 140, which is less than, or to the left of, the previous horizontal coordinate. To move the circle to the right of center, you can change the coordinates in line 20 to:

```
20  CIRCLE(350,147),60
```

This will shift the circle to the right. To move the circle up or down, it is necessary to alter the Y coordinate specification. Change line 20 to:

```
20  CIRCLE(140,127),60
```

The Y coordinate of 127 is 20 points less than the previous Y coordinate, or 20 pixels higher on the screen. When the revised program is run, the circle will appear in the upper half of the screen. The center of the circle will still be at the horizontal center, but somewhere above the vertical center.

To see what the radius value does, change line 20 to:

```
20  CIRCLE(248,147),30
```

The X,Y coordinates again specify a point at the exact center of the usable screen. However, the circle radius has been cut in half and the circle will be half the size of the one drawn previously.

At this point, try changing the values in line 20. You will find that you can change the size of the circle by altering the radius value. You may also specify values that cause part of the circle to exceed the width of the screen. This can be done by changing line 20 to:

```
20  CIRCLE(450,147),60
```

This places the circle at the extreme right of the screen. The computer will attempt to display a uniform circle, but there is not enough screen space to do it, since if you add 60 points to the horizontal coordinate of 450, you arrive at a total of 510 points horizontal, and the screen can only display 495 points. No error message occurs here, but that portion of the circle beyond the coordinates of the screen is simply not shown.

The CIRCLE statement is a good one to begin with. It demonstrates how screen coordinates are used and also displays how this powerful graphics

statement is used. Those readers who have had experience with other types of computers that do not offer the CIRCLE statement will immediately realize its value. Drawing a circle by plotting points one at a time is an extremely difficult programming procedure. It can be done, but it's no fun.

LINE STATEMENTS

The LINE statement does just what its name implies—it draws a line on the screen. It can also do much more, as we will see later. The LINE statement can be used in many ways, but the basic format is:

```
LINE(X,Y)-(X2,Y2)
```

Hey! What's that X2, Y2? Don't panic! The explanation is quite simple. Obviously, a line must have a starting point and an ending point. Therefore, the LINE statement uses two sets of X, Y coordinates. The first indicates the starting point of the line, while the second set indicates the ending point. The following program demonstrates the use of the LINE statement in Microsoft BASIC:

```
10  CLS
20  LINE(10,147)-(250,147)
```

The LINE statement in line 20 simply tells the computer to draw a line from screen coordinates 10,147 to screen coordinates 250,147. Since the Y coordinate values are the same in both sets, this will be a horizontal line drawn from horizontal position 10 to horizontal position 250. The vertical position of the line will always be at Y coordinate 147. Now, let's draw a vertical line on the screen by changing line 20 to:

```
20  LINE(100,50)-(100,250)
```

Here, the first coordinate designation of 100 stays the same in both sets. Therefore, the horizontal position is fixed. Only the Y coordinate values change from set to set. When this program is run, a vertical line will be drawn at the horizontal center of the screen from vertical coordinate 50 to vertical coordinate 250. If the X coordinates change and the Y coordinate remains constant, a horizontal line is drawn.

We can also draw diagonal lines simply by changing both sets of coordinates. To demonstrate this, change line 20 to:

```
20  LINE(50,80)-(90,250)
```

This will draw a diagonal line slanting downward from left to right, since the start of the line is at screen coordinate 50,80 and the ending point is at 90,250.

You can now experiment a bit with the LINE statement to see what happens when coordinate values are switched around. You must be certain that X coordinate values range from 0 to 495 and Y coordinate values range from 0 to 293. If a coordinate is out of this visual range, the line will go off the screen.

The method of forming lines by specifying coordinates is known as the *absolute* method. The LINE statement examples shown thus far have specified the exact starting and ending points of the lines to be drawn. However, there's another way to draw lines as well, as the following program demonstrates.

```
19  CLS
20  LINE(50,10)-(50,100)
30  LINE-(150,100)
```

When this program is run, you will see the letter L drawn on the screen. You have no doubt noticed that line 30 contains another LINE statement, but this one has only one set of coordinates preceded by a

hyphen (-). The LINE statement in line 30 uses the relative coordinate method. Here's what happens. The computer draws the line programmed in line 20 first. When this first line has been drawn, the graphic coordinate position stops at 50,100. Line 30 tells the computer to draw another line from its present graphic position to coordinate 150,100. To visualize this best, one must imagine an invisible graphic cursor. When line 20 is executed, the graphic cursor is first positioned at coordinates 50,10 and then travels to coordinates 50,100, where the line ends. When line 30 is executed, the graphic cursor position of 50,100 is already locked into the computer by completing execution of the previous line. The computer assumes the line start point to be where the cursor is, at 50,100. To use the relative method of drawing a line, all you have to do is specify the line point the line should end at. You can run this program another way by removing line 20 altogether. Now, when the program is run, a line will be drawn starting at coordinates 248,147 and ending at the specified coordinates of 150,100. Unless otherwise specified, the graphic cursor is initially positioned at coordinate 248,147, the center of the screen.

We are often accustomed to thinking of everything as beginning at the left and ending at the right. The computer is not handicapped by such an assumption. In this case, a line was drawn from right to left rather than from left to right. Since we use the relative form of coordinate specification and the graphic cursor was automatically set to coordinates 248,147, the computer drew a line starting at coordinates 248,147 and moving left to coordinates 150,100. The same line could be drawn on the screen from left to right by the following:

```
LINE(150,100)-(248,147)
```

This line would begin at horizontal coordinate 150 and end at horizontal coordinate 248. Either way, the line looks the same, even though the computer looks at it in a completely different manner. For the purposes of our discussion, the lines

```
LINE(150,147)-(248,147)
```

and

```
LINE(248,147)-(150,147)
```

produce identical results. In the first example, the line is drawn from left to right, while in the second the same line is drawn from right to left. Either way, a line is displayed *between* coordinates 150,147 and 248,147.

USE OF COLOR

You will undoubtedly notice when leafing through the Microsoft BASIC reference manual that several graphic functions may be used with an optional color designation. The Macintosh screen is monochrome, but this does not mean that the Macintosh won't support color. It will, but you can't see it on the screen. I would imagine that before too long, Apple will be offering a color monitor package for the Macintosh. Microsoft, anticipating this, has provided color options for many graphics statements.

Any color on a computer is normally represented by a color number with a BASIC graphics statement. For now, there are only two color numbers that are applicable, and in most instances, these need not be used. The color number for black is 33, and the one for white is 30. Both the CIRCLE and LINE statements will accept a color number, but if none is provided, the computer automatically defaults to 33, producing a black foreground. For instance,

```
10 CIRCLE(248,147),60,33
```

would specify a circle with a radius of 60 points and a color of black, represented by 33. If the ",33"

designation is dropped, the computer automatically assumes 33. If you specified a color number of 30, the circle would be drawn in white, the same as the background, and it would not be visible. The LINE statement could be used as follows:

```
LINE(0,0)—(248,147),33
```

This specifies a line drawn from the first coordinate to the second in black, although if the color designation is omitted, 33 is still assumed. For the present, don't even worry about the color numbers, although we will get into a practical use for them a bit later in this chapter. Also, on the present Macintosh configuration, you can use an odd number such as 1, 3, or 5 to designate a black foreground or an even number, such as 2, 4, or 6 to specify a white foreground (invisible). When an odd number is inserted, the computer defaults to 33, whereas an even number causes the computer to default to 30. Again, in most instances, it is not presently necessary to use color designators, but it is important to know that they exist, especially if color eventually becomes available for the Macintosh.

PSET/PRESET STATEMENTS

The PSET statement is used to draw a pixel at a certain location on the screen specified by the standard X,Y coordinates. PRESET is almost identical to PSET, but when both are used without color designators, PSET sets the point and PRESET erases that same point. The format of PSET is

```
PSET(X,Y),C
```

where X and Y are the screen coordinates and C is the color number. In most instances, this number will not be used. To illustrate how these statements are used, clear any program currently in computer memory by typing NEW. You will be asked through prompts whether or not you wish to save the existing program. Select YES if you do and NO if you don't.

Now, in direct mode, type CLS to clear the screen. Then type:

```
PSET(248,147)
```

All of this is done in direct mode, so don't include a line number in front of the PSET statement. When you press RETURN, you will see a black point at the center of the screen. This pixel was written at screen coordinates 248,147. Now, type:

```
PRESET(248,147)
```

The point disappears. This is due to the fact that PRESET wrote a point at the same coordinates, but in a color of white so the point is invisible.

The following program describes a method of drawing a line on the screen by using PSET instead of the LINE statement. In most instances, we would use the LINE statement, since it takes less time to input and uses less memory. However, this program demonstrates how PSET might be used.

```
10  FOR X = 0 TO 495
20  PSET(X,147)
30  NEXT X
```

When this program is run, a line will be drawn from the left side of the screen to the right side at the vertical center. Here's how the program works. Line 10 sets X to count from 0 to 495. You will recall that these numbers make up the visible column coordinates on the Macintosh screen. Line 20 uses PSET to plot points on the screen. However, the X coordinate normally used in PSET is represented by variable X, which will be equal to 496 different numbers from 0 to 495 as the loop cycles. The Y coordinate is fixed at 147, which specifies the verti-

cal center of the screen. On the first pass of the loop, PSET plots a point at 0, 147. This is because X is equal to 0 on the first pass. On the next pass of the loop, a point is plotted at 1, 147, then 2, 147, and so on. The end result is a line that begins at coordinates 0, 147 and ends at coordinates 495, 147.

This program consumes three lines and is the exact equivalent of the following one-line program:

```
10  LINE (0,147)-(495,147)
```

Certainly, it is more efficient to use the LINE statement to produce the line, but we can also use PSET.

To demonstrate PRESET, add the following lines to the existing program:

```
40  FOR X = 495 TO 0 STEP -1
50  PRESET (X,147)
60  NEXT X
```

When you run the total program, you will see the line travel from left to right across the screen and then it will seem to spring back toward the left again. Here's what happens. PSET writes the line from 0, 147 to 495, 147. PRESET then begins erasing the line from 495, 147 to 0, 147 as the loop cycles. The line is drawn a point at a time and then erased a point at a time. To make the program even more interesting, add the following line:

```
70  GOTO 10
```

Now, when you run the program, the line will constantly spring out from the left side of the screen, travel the width of the screen, and then loop back to the left side of the screen. It will do this over and over again until you halt it. This is basically how animation works on the computer screen. A group of points is drawn at one location, erased from the original location, and then redrawn to another location.

Actually, all graphics are created by drawing a point at a time. Powerful statements like CIRCLE and LINE are written in a code that the computer can understand without any interpretation. Therefore, when CIRCLE or LINE is used, a subroutine that is contained in computer memory is run. It runs faster because it is not written in BASIC, but rather in machine language. This subroutine is called by BASIC using the specified coordinates.

As stated earlier, the CIRCLE statement was a tremendous aid to graphics programming in BASIC and that some dialects do not offer this capability. A circle can be programmed a point at a time using PSET, but it's a cumbersome chore. The following short program will draw a crude circle on the screen using PSET. The circle is not very good, but it illustrates how useful CIRCLE is. Also, it takes quite some time for the total circle to form. This program uses a mathematical formula that will plot each point. It won't take you very long to input this program and see the results.

```
10  CLS
20  FOR X = -100 TO 100
30  Y=(10000-X*X) ^ .575*.5
40  PSET(X+248,Y+130)
50  PSET(X+248,130-Y)
60  NEXT X
```

PSET is often used to draw irregular shapes on the computer screen. This includes those that cannot possibly be produced readily using CIRCLE or LINE. Technically, anything that can be drawn using PSET can also be drawn using LINE, but since the former works its magic a point at a time, it's sometimes easier to use PSET. PSET is often used for programming special text fonts. Often, the characters are worked out in PSET fashion and then committed to decimal code by a sophisticated conversion program. These methods lie outside the realm of this discussion, but chances are you will be

using PSET more and more as your exploration of Macintosh graphics continues.

ANIMATION TECHNIQUES

As was previously mentioned, animation on the computer involves drawing an object to one screen location, erasing it, and then drawing it to another location. If the draw, erase, and draw again routine is handled at the correct speed, the impression of continuous movement is attained. One example was provided earlier using PRESET and PSET. The following program will demonstrate animation using a FOR-NEXT loop and the PRINT statement. You've seen this animation time and again when using your computer.

```
10  CLS
20  FOR X = 0 to 1000
30  PRINT X
40  NEXT X
```

When this program is run, numbers will be printed vertically from top to bottom on the screen. However, when the bottom of the screen is reached, the appearance is given that the numbers are moving upward. Numbers seem to appear at the bottom of the screen and disappear at the top. This is called scrolling, but in effect, the numbers are simply being written in one location, erased, and written in another location. Let's slow down the program by adding the following line:

```
35  FOR DLAY = 1 TO 200:NEXT DLAY
```

This FOR-NEXT delay loop simply slows the computer down a bit so we can see this animation in slow motion. Now, the numbers are displayed much more slowly. Watch the bottom of the screen, and you will see new numbers appear while the older numbers seem to be pushed upward. This is not the true case at all. Assume at one instant that you have a 59 at the bottom of the screen. The number 58 lies above it. The next instant, you have a 60 at the bottom of the screen with a 59 above it and a 58 above the 59. What's really happening is the original 59 was overwritten by 60. The original 58 was then overwritten by 59, and so on. The entire screen changes with each number that is displayed. When 60 comes up, the 59 that appears above it is *not* the old 59 originally at the bottom, but a newly written 59. This applies to every other number that is visible on the screen. Now, change line 30 to:

```
30  PRINT 60
```

Now, the number 60 will be displayed, rather than the value of X. A column of 60s will appear at the left side of the screen, but when the bottom of the screen is reached, the numbers seem to freeze, except for the bottom 60, which seems to be flashing on and off. Actually, the screen is being rewritten *each time* a new number 60 comes up. We just can't see it because the old numbers are being written over by identical new numbers.

Serious animation on the Macintosh uses two sophisticated graphics statements, PUT and GET. Once an object is written to the screen, the GET statement is used to collect the screen information that allows this object to be displayed. The information is committed to an array that is large enough to hold the points that make up the image. The PUT statement is used to put the same image elsewhere on the screen at a point determined by coordinates. PUT and GET are very powerful, as they can move screen images with great rapidity.

When you use GET, the image is copied to an array. This does not remove the image from the screen, but simply provides a copy of the image that may be used elsewhere. The PUT statement allows the copy to be placed at another location on the screen. After you get it using GET, you can remove the original image from the screen with a CLS statement, but it is more often done using PUT. With PUT the image is placed in its original loca-

tion; that is, the copy of the image is put exactly over the original image. When used in this manner, PUT will make an image disappear from the screen. To clarify them, the steps that are used to perform animation are listed here.

1. Create the image on the screen using CIRCLE, LINE, or PSET.
2. Get the image from the screen using the GET statement.
3. Erase the original image from the screen using the PUT statement.
4. Put the image elsewhere on the screen, using the PUT statement again.
5. Erase the image from its new location by putting it on itself using PUT.
6. Put the image at another location on the screen using PUT.

Still confused? Let's go over an example.

Imagine that a circle is drawn on the screen. Before we can start the animation sequence, it is necessary to GET this circle using the GET statement. How this is done will be described later. Once you get the circle, the computer retains the image in its memory, whether it's on the screen or not. To erase the original image, we use PUT to place the computer recollection of the image on top of the actual screen image. It disappears from view. We then use PUT again to place the image on the screen at a new location. We now have to erase this newly located image to place it elsewhere, so we use PUT to place the computer recollection of that image onto the actual screen image again. We only have to GET an image once, but we always have to PUT it twice after the first sequence is over. Once you GET the initial image and then PUT to erase it, forever after, the first PUT will place the image back on the screen and the second PUT will erase it, assuming that the second PUT specifies the same coordinates as the first PUT.

When we use GET on an image, the computer remembers exactly what it looks like by placing its contents in an array. Fortunately, when using PUT and GET, we don't have to worry about any intricacies involved in making assignments to an array. GET does this automatically. The following program demonstrates PUT and GET:

```
10  CLS
20  DIM A(500)
30  CIRCLE(248,147),20
40  GET(228,127)-(268,167),A
50  PUT(100,100),A
```

When you run this program, you will immediately see two circles on the screen. The two are identical. The one at the center is the one that was drawn using the CIRCLE statement in line 30. The one at the left is the image of the original that was placed at the location on the screen by the PUT statement in line 50. We haven't accomplished visible animation yet, but be patient.

Now, the GET statement in line 40 has two sets of coordinates. These specify a box that is large enough to hold the circle. The first coordinates specify the upper left edge of the box. The second set of coordinates specify the lower right edge of the box. Figure 7-2 illustrates this.

To produce these two sets of coordinates, first, the circle was drawn in line 30 at the center of the screen. The circle was given a radius of 20 pixels. Therefore, to arrive at the box size needed to contain this circle, 20 was added to the circle coordinates provided in line 30 for one set, and 20 was subtracted from the same coordinates for the other set. Obviously, a circle with a radius of 20 points whose horizontal center is coordinate 248 will display its leftmost edge 20 points lower in count than 248, or 228. As for its rightmost edge, this will occur at 248 + 20, or 268. The same math was performed on the Y coordinate in line 30 to arrive at the top and bottom locations of the circle. These coordinates are used with the GET state-

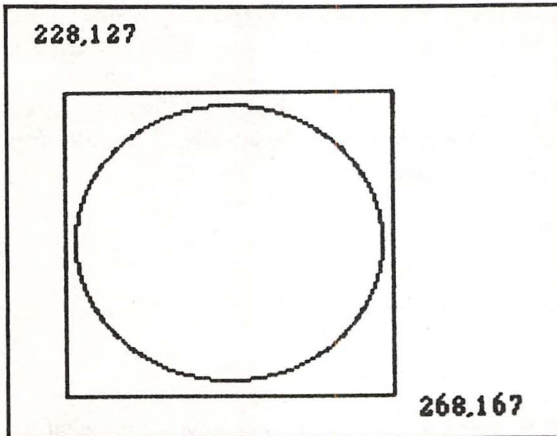


Fig. 7-2. When Using GET, the First Set of Coordinates Mark the Upper Left-hand Corner of an Invisible Box that Surrounds the Image to be Animated. The Second Set Mark the Lower Right Edge of the Same Box

ment in line 40, followed by ,A. The A simply specifies that the image is to be stored in array A. There is a fairly complex formula that can be used to determine the exact size of the array needed to hold an image, but you won't need it. The DIM statement does not have to tell the exact size of the array. An approximation is good enough. If your array is not large enough, you will get an error message; and then you can simply increase the size of your array.

Those of you who may eventually be writing extremely large programs that might tax the memory limits of your computer may wish to go through this formula in order not to DIM arrays that are larger than necessary. When memory is sparse, you can perform the math involved, following the explanation in the Microsoft manual.

All right, at this point we have written an image to the screen using the CIRCLE statement and stored it in array A using GET in line 40. To place the image elsewhere, the PUT statement uses a single set of coordinates that specify where the upper left-hand edge of the box is to appear. Again, the box is invisible, but it does contain the image of the circle. Coordinates 100,100 were chosen at random. Of course, this program leaves the original circle at the center of the screen, but this can be easily corrected by adding the following line between lines 40 and 50:

```
45  CLS
```

Now, when the program is run, the same set of events takes place, except that line 45 clears the original image from the screen before the PUT statement in line 50 places it in a new location. However, a more professional way of accomplishing the same result is to change line 45 to:

```
45  PUT(228,127),A
```

Note that array A is specified with both PUT and GET. The array must be specified or you will receive an error message. Line 45 now "PUTs" the copied image to the original image on the screen, because it uses coordinates 228,127, which mark the upper left edge of the box containing the circle. When an image is PUT to itself, it disappears from the screen. When you run this program, there will be a single circle on the screen.

Of course, you got the same result when you used CLS. So why go through the bother of using the more complex PUT line? A good question, but there's a good answer! The CLS statement clears the entire screen, which is fine in this particular program application, but PUT, when used to PUT an image to itself, erases only the image from the screen and leaves the background intact. Suppose you were trying to animate a bouncing ball across an elaborate background, such as a picket fence, for example. If you use CLS, your entire background will be cleared along with the image. If you use PUT, an image to itself, only the image disappears. The background remains the same. The following program demonstrates this.

```
10  CLS
20  DIM A(500)
30  CIRCLE(248,147),20
40  GET(228,127)-(268,167),A
50  LINE (0,147)-(495,147)
60  PUT(228,127),A
```

When this program is run, you will see a line across the center of the screen and, if you look quickly, a circle at the center of this line. However, the circle immediately disappears because the image has been PUT to itself with the PUT statement in line 60. Now, replace line 60 with CLS. Again, you will see the line and the circle, but just as quickly, the entire scene disappears. This shows how it is better to PUT an image to itself than clearing the screen when a background is involved.

Let's take our animation routine one step further and try to get the ball to thread the entire line from left to right. Change line 60 back to its original form (PUT instead of CLS). Now add the following line:

```
70  PUT(0,127),A
```

When you run this program, the circle will appear on the line, but at the far left of the screen. In order to make it move, we must PUT the image to itself again (at this new location), so we blot it out by adding another line:

```
80  PUT(0,127),A
```

Now, we can make it move across the line by simply increasing the value of the X coordinate and leaving the Y value at 127. Each time we use PUT to place the image on the screen, we must use an identical PUT to erase it *before* placing the image elsewhere. But this sure would be hard work!

Perhaps you've already gotten an idea of how we could do this a little easier. That's right! Why not allow a FOR-NEXT loop to specify the value of the X coordinate? As an experiment, erase lines 70 and 80. In direct mode, type 70 and press RETURN. Do the same for line 80. Now type LIST. You can see that lines 70 and 80 have been deleted from the program. Now, add the following:

```
70  FOR X = 0 TO 400
80  PUT(X,127),A
90  PUT(X,127),A
100 NEXT X
```

Now run the program. Eureka! Our circle travels from left to right across the plotted line. If you would like the circle to travel on top of the line, change lines 80 and 90 to a Y coordinate value of 105 instead of 127. Since we are decreasing the Y coordinate value, the image will move toward the top half of the screen. The value chosen here should place the ball at the very top of the line.

Now, how do we control the speed of animation? First, we must think about what is taking place in this simple program. The invisible box that holds the circle is being positioned on the screen horizontally by using the value of X supplied by the FOR-NEXT loop. During the first cycle of the loop, the upper left edge of the box is written at screen coordinates 0,127. On the next cycle, it's written at coordinates 1,127, then 2,127, and so on. (The Y coordinate will be 105 if you have not changed your program back to its original form.) This program is moving the circle to the right a single point at a time on each pass of the loop. To speed up the movement, all that's necessary is to increment our FOR-NEXT loop by a value greater than 1. Change line 70 to:

```
70  FOR X = 0 TO 400 STEP 2
```

Now, X will be incremented by 2 instead of 1, so the box will be written at coordinates 0,127 on the first pass, then 2,127; 4,127; and so on. Since the loop is skipping every other number, the image will be

written at every other pixel position. When you run the program, you will see that the image run approximately twice as fast as before.

Now, change line 70 to a step value of 4. The circle moves even faster. Try something wild, and use a step factor of 40. The circle will fly across the line so fast that you can hardly see it. This is much too fast for visible animation. I think you will find a step factor of 10 produces the fastest practical speed.

This program effectively demonstrates animation and the use of GET and PUT to not only manipulate images, but save the background as well. But we *can* go further.

Let's double our animation capability by adding two more lines. Change lines 80 and 90 to:

```
80  PUT(X,105),A
90  PUT(X,105),A
```

and add the following lines:

```
95  PUT(400-X,149),A
96  PUT(400-X,149),A
```

Now run the program. You will now see the same horizontal line, but one circle is rolling from left to right just above the line, while another rolls from right to left just below the line. Lines 95 and 96 animate the second circle. Here, the horizontal position represented by the X coordinate is arrived at by subtracting the value of X from 400. You will recall that this is the top value of the FOR-NEXT loop. Lines 80 and 90 animate the circle that begins on the left, while lines 95 and 96 animate the one that begins on the right. Each time X is incremented, it is subtracted from 400; so the bottom circle moves toward the left. We can even add a few sound effects by including one more line:

```
97  IF X = 200 THEN BEEP
```

The BEEP statement simply causes the internal speaker of the Macintosh to emit a 100-cycle tone for a short duration. I chose a value of 200 for line 97 because it is at this horizontal coordinate that the two circles are aligned. When you run the program, the circles travel as before, but as soon as they meet, the beep is heard, which might indicate a collision or anything else your imagination can conceive.

It is through the combination of multiple animation and sound effects that extremely interesting game programs and simulations are produced. You will also notice an increasing amount of flicker as the program discussed here becomes more complex. This flickering is caused by the fact that the computer has to work harder and is slowed down by the additional lines. Add to this the fact that any BASIC interpreter is slow when compared with machine language programs, and the real problem comes to light. While it is possible to write highly sophisticated computer animation routines in BASIC, by and large, these routines run so slowly that they are not practical. This is not the fault of the computer, but of the language.

All BASIC interpreters, because they run programs one line at a time, tend to be slow, and when speed is of the essense, as is the case with sophisticated animation programs, an interpreter is simply not adequate. Fortunately, a BASIC compiler should soon be out for the Macintosh. A compiler will pull the program into one unit and, once it is compiled, it will run three to six times faster.

If you're interested in animation programming, please don't be disheartened by the limitations of a BASIC interpreter. Many good animation programs can be written in this language. Programming under a BASIC compiler is almost identical to programming under an interpreter. That is, the same statements and functions are usually available. So write your programs now and, when the compiler is available, you'll be able to compile them and see them run faster.

MORE WAYS TO USE GRAPHICS STATEMENTS

Many Microsoft graphics statements have been discussed so far. However, many statements may be used differently to create even more useful results. Let's outline some of these modifications now.

The CIRCLE statement was used to create a perfect circle on the screen. It was used with the format

```
CIRCLE(X,Y),R
```

where X and Y are the coordinates of the circle's center, and R is the radius in points or pixels. However, the CIRCLE statement may also be used in this format:

```
CIRCLE(X,Y),R,C,start,end,aspect
```

Here, C represents the color of the circle, which is not necessary unless for some reason you wish to write a white circle against a black background. The start and end designators allow you to draw partial circles on the screen by inputting numeric values. The aspect number is normally 1, which produces a perfectly round circle. However, by increasing or decreasing this value, the circle can be turned into a horizontally or vertically configured ellipse.

It is not necessary to include all of these parameters, but it is necessary to insert commas if you wish to use one or two that lie to the right of others. For instance, if you wish to produce a fairly fat ellipse, you could do this with the following line:

```
CIRCLE(248,147),100,,,,.5
```

Here, the aspect ratio is ,5. We have not elected to use the color, start, or end specifications, often referred to as *parameters*. If they were used, they would be separated by commas; hence, the four sequential commas following the radius parameter of 100. To test this single-line program, simply type CLS in direct mode to clear the screen and then input the program line in direct mode without a line number. You will see a horizontally configured ellipse whose horizontal radius is the same as it would be with a normal circle. However, the vertical radius is one-half that of the horizontal radius because of the 0.5 aspect designator. Type CLS again in direct mode and input the following line:

```
CIRCLE(248,147),100,,,,2
```

Press RETURN and you will see the same ellipse, only this time it is configured in a vertical manner with the horizontal radius one-half the normal size and the vertical radius normal size.

Using the aspect ratio, we can produce many different types of elliptical forms. Remember, if you use an aspect ratio of 1 (the default aspect ratio), a perfect circle will be produced. The aspect ratio is a figure that denotes the ratio between the horizontal diameter or radius and the vertical diameter. If the aspect ratio number is less than 1, the ellipse will be configured horizontally. If it is more than 1, it will be configured vertically.

Choosing to program specific arcs of a circle is a little more complex. Here, it is necessary to determine a start and end point for the arc. Figure 7-3 shows a representation of a circle with four of its points specified. The maximum value of any arc beginning or ending segment will be two times PI, where PI is equal to 3.141593. At the right center of the circle is a value of 0 and two times PI. Either of these values can serve as a starting or ending point for an arc. If you use a starting point of 0, the arc will be drawn clockwise from that point. A starting point of two times PI will produce an arc that is drawn counterclockwise. To test this, type CLS in direct mode and input the following line:

```
CIRCLE(248,147),100,33,6.28,3.14
```

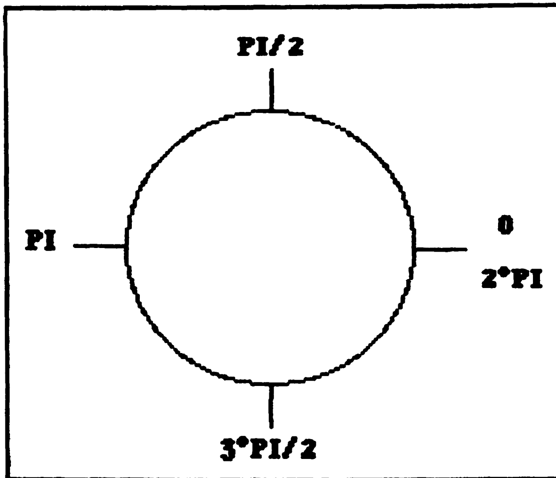


Fig. 7-3. Chart Used to Calculate Circle Arcs.

Here, I have inserted the number 33 for the color parameter, although I could have just as easily omitted this parameter by simply inserting a comma. When this program is run, a circle will be drawn counterclockwise from our right starting point of 2PI, represented by 6.28, to the opposite side of the circle, which is a point represented by PI, or 3.14 in this program line. The arc is drawn in a counterclockwise direction because 2PI was used as the starting value. Now, type CLS again and press RETURN. Then type:

```
CIRCLE(248,147),100,33,0,3.14
```

When you press RETURN, you will see a mirror image of the previous arc. This arc starts from the same starting point and proceeds to the same ending point, but the travel is clockwise because the starting point specified is 0. Clear the screen again, and type:

```
CIRCLE(248,147),100,33,3.14,0
```

When you press RETURN, another arc appears. This one looks just like the previous one, but it starts at the left side of the circle (PI). Since the ending point is specified as 0, the movement is counterclockwise. This may be a bit confusing, but just remember that if the starting point is larger than the ending point, movement will be counterclockwise. If the starting point is smaller, movement will be clockwise.

This one will take a bit of experimentation, but if you practice for a while, you will begin to get the hang of forming accurate arcs on the computer.

Another expandable graphics statement in Microsoft BASIC is LINE. So far, the format discussed has been:

```
LINE(X,Y)-(X1,Y1)
```

Here, X and Y mark the starting point of the line, while X1 and Y1 are the ending point. However, the LINE statement may also be set up in this format:

```
LINE(X,Y)-(X1,Y1),C,B(F)
```

Here, C is a color numeral, B indicates that, instead of a line, a box is to be drawn. Here, the upper left or lower left corners of the box are defined by the X,Y coordinates, while the upper right or lower right corners of the box are defined by the X1,Y1 coordinates. If Y is smaller than Y1, X,Y will define the upper left-hand corner of the box, when the original line travels from left to right. If Y is larger than Y1, X,Y will normally specify the lower left corner of the box. Let's experiment a bit. In direct mode, type CLS and press RETURN. Now, type the following:

```
LINE(10,50)-(100,60),,B
```

When you press RETURN, you will see a slender horizontally configured box in the upper left corner of the screen. The upper left corner of the box is at coordinates 10,50. The lower right corner of the box is at coordinates 100,60. You will notice that

the height of a horizontally configured box or rectangle is determined by the difference between the Y value and the Y1 value. Now type the following:

```
LINE(10,50)-(100,200),,B
```

When you press RETURN, you will see a larger box that is vertically configured at the left side of the screen. This is the same box as before, except its travel in the Y dimension is greatly increased due to the increased difference between Y and Y1. If the difference between Y and Y1 is greater than the difference between X and X1, the box will be vertically configured. If not, it will be horizontally configured. If there is no variance in the differences in X and X1 and in Y and Y1, the box will be a square. Try this line:

```
LINE(10,10)-(100,100),,B
```

When you press RETURN, you will see a perfect square in the upper left corner of the screen because the difference between X and X1 is the same as the difference between Y and Y1.

You may also use BF instead of B with the LINE statement. This works the same way only the box is completely filled in. The letter B is an alphabetic designator that stands for box, while BF stands for box fill. Clear the screen and type the following:

```
LINE(10,10)-(100,100),,BF
```

When you press RETURN, you will now see the square completely filled in black. The BF command is used exactly like the B command, only the rectangles produced are completely filled in. To fill in the entire screen with a black background, simply type:

```
LINE(0,0)-(495,293),,BF
```

This produces a box that starts at the upper left-hand corner of the screen and has as its lower right corner the lower right corner of the screen. When you press RETURN, the box fills the entire screen.

By using combinations of standard CIRCLE and LINE statements with the expanded parameters, a large number of graphic displays can be easily produced. You may need to draw a picture that uses some open boxes, filled boxes, and lines. You can draw all of them with the versatile LINE statement. Likewise, if you need to draw circles, arcs, or ellipses, the CIRCLE statement alone can help you draw them. As you become more familiar with Microsoft BASIC, the versatility of this very powerful, high-level language becomes more and more apparent.

THE POINT FUNCTION

The POINT function has one primary purpose. It is used to indicate what color any pixel on the screen is displayed in. POINT is the only graphics function resident in Microsoft BASIC. Its format is

```
C = POINT(X,Y)
```

where X and Y are any valid set of screen coordinates. C will be equal to 33 if the color of the pixel at coordinates X,Y is black. If it is white, C will be equal to 30. To illustrate the POINT function, type CLS in direct mode and press RETURN. Now, type:

```
PRINT POINT(248,147)
```

When you press RETURN, the number 30 will appear on the screen. This means that the point at the center of the screen (coordinates 248,147) is white, since 30 is the color number that represents white. Clear the screen again, and type:

```
PSET(248,147)
```

When you press RETURN, you will see a black point of light at the center of the screen. Now type:

PRINT POINT(248,147)

The number returned is now 33, which indicates that the point at coordinates 248,147 is now black.

To write a standard BASIC program that will do the same thing as our direct mode inputs, we might use the following:

```
10  CLS
20  PSET(248,147)
30  C = POINT(248,147)
40  PRINT C
```

Actually line 30 is not necessary at all. We could omit it and change line 40 to

```
40  PRINT POINT(248,147)
```

to print the point. When the program is run, line 20 sets a point at the center of the screen. Line 30 assigns the variable C to the value of the color at coordinates 248,147. Line 40 simply prints the value of C.

It might seem silly to go through all this to determine the color of a point at the center of the screen, especially when we know the color. However, POINT is an extremely useful function, especially for screen modifications. Using the POINT function and some simple programming routines, we can effectively enlarge or shrink the size of an image on the screen. The following program illustrates shrinking an existing image:

```
 10  CLS
 20  XX = 240
 30  YY = 140
 40  LINE(0,0)-(20,20),,BF
 50  FOR X = 0 TO 20 STEP 2
 60  FOR Y = 0 TO 20 STEP 2
 70  C = POINT(X,Y)
 80  PSET(XX,YY),C
 90  YY = YY+1
100  NEXT Y
110  YY = 140
120  XX = XX+1
130  NEXT X
```

Here's how the program works. Line 10 clears the screen, while lines 20 and 30 assign values to numeric variables XX and YY. Line 40 uses the LINE statement with a box fill (BF) command to produce a solid box in the upper left corner of the screen. This box is 21 pixels in length and 21 pixels in height. (Remember, we start counting at 0.) Line 50 sets up a FOR-NEXT loop that counts from 0 to 20 in steps of 2. Line 60 does the same thing, but uses Y as its variable instead of X. Each of these loops is designed to count through the screen coordinates occupied by the box. However, since the STEP 2 option is used with both FOR-NEXT loops, the scan will access every other pixel in the box instead of every pixel. The POINT function is used in line 70. It assigns to numeric variable C the color number of the pixel at the X,Y location determined by the two FOR-NEXT loops. Line 80 uses the PSET statement to set a point at coordinates XX,YY. The point that is set will be specified by the color number returned to C.

The values of X and Y are used to allow the POINT function to read every other point in our filled box. Variables XX and YY are used to determine a new set of screen coordinates at which the shrunken image of the box will be written. X and Y are counted in steps of 2, whereas XX and YY are counted in steps of 1 due to the count routines in lines 90 and 120. Each time a point is set, YY is incremented by 1. Each time the Y loop cycles out, XX is incremented by 1. Each time the X loop is incremented the Y loop cycles 20 times.

The end result is the POINT function returns

the color numeral of every other point in our screen box. The PSET statement sets points in steps of 1 at a different location on the screen. Since POINT samples only every other point, the new image is one-quarter the size of the original box. Most people would think it would be one-half the size rather than one-quarter, but my figure is correct, since the new image is both half the original height *and* half the original width. Try the program again, but omit the F portion of the BF parameter. Now, an open box will be drawn at the top left of the screen, and the shrunken image will appear to be identical, although smaller. This may not seem fantastic until you remember that in the first example, variable C was always equal to 33 some times and to 30 at others, depending on whether or not the POINT function is accessing one of the lines that make up the box or the white space in its interior. Now, add the following line:

```
45  CIRCLE(10,10),10
```

When the program is run with this revision, the target box in the upper left corner is now occupied by a circle, but the shrunken image at the center of the screen still copies it exactly. Regardless of what you have on the screen at the location sampled by POINT, you will have the same thing represented in shrunken form at the center of the screen.

Since every other dot in the original image is sampled, the shrunken image will not be as detailed. This is due to the fact that less dots are used to form the shrunken image than were used in the original. However, for simple shapes, this is usually of little consequence. The larger the image to be shrunk, the better the quality will be. For example, if you take out lines 40 and 45 and change line 40 to

```
40  PRINT "Q"
```

you will see what I'm talking about. Instead of a box in the upper left-hand corner, the letter Q is displayed. However, since this image is already quite small, it does not shrink well because the original is composed of only a few points and many of these are skipped over during the scanning process. Only a few points appear at the center of the screen. For solid images, this little program can work wonders, especially since it can be expanded to read a much larger screen area. If you want to experiment further, change line 40 to:

```
40  PRINT "W"
```

This letter is composed of a larger number of points and will reproduce slightly better at the center of the screen. If you look very closely, you will recognize the shrunken image as a W.

We can use the POINT function to enlarge an image as well. This involves a reversal of the shrinking routine, which sampled every other dot. An enlarging routine means the two loops must be set up to scan the image to be enlarged. In the shrink routine, every other dot was sampled; in the enlarging routine, every dot is sampled. When the retrieved information is displayed in enlarged form on the screen, the enlargement is accomplished by printing four dots for every one sampled. The program follows.

```
 10  CLS
 20  XX = 240
 30  YY = 100
 40  LINE(0,0)-(20,20),,B
 50  CIRCLE(10,10),10
 60  FOR X = 0 TO 20
 70  FOR Y = 0 TO 20
 80  C = POINT(X,Y)
 90  PSET(XX,YY),C
100  PSET(XX+1,YY),C
110  PSET(XX,YY+1),C
120  PSET(XX+1,YY+1),C
```

```
130  YY = YY + 2
140  NEXT Y
150  YY = 100
160  XX = XX + 2
170  NEXT X
```

When you run this program, you will see the original box in the upper left corner of the screen filled with a circle. At the same time, you will see the enlarged image begin to form at the center of the screen. Here's how the program works.

When the POINT function in line 80 is executed for the first time, X and Y are equal to 0. The color numeral is returned to numeric variable C. Lines 90 through 120 handle the enlarging process. In line 90, PSET places a point (same color as the original box) in screen location XX,YY. Line 100 then writes another point just to the right of that point at position XX+1,YY. However, we must enlarge both vertically and horizontally, so line 110 sets a point beneath the original one at coordinates XX,YY+1. Line 120 sets a point to the right of this one. We are using four dots in place of a single dot in the original figure to provide an enlargement factor of 2 both horizontally and vertically. You may substitute any image for the one programmed in lines 40 and 50 and have it enlarged. To test this, omit line 50 and change line 40 to:

```
40  PRINT "W"
```

When this program is run, an average-sized W will appear in the upper left corner of the screen and its enlarged version will appear at the center. This W is identical to the one displayed by the Macintosh character set, only it is larger. This is the method used to create the various font sizes in *MacWrite* and *MacPaint*, only the *MacPaint* and *MacWrite* enlargement routines are written in machine language so that they work faster. However, the enlarged characters produced by this BASIC program are just as accurate as those produced by the *MacPaint* and *MacWrite* routines.

The POINT function is indeed very useful in graphics programming. It can tell you what is currently on the screen on a point by point basis. This information can be placed anywhere on the screen in any size. Also, the information returned by the POINT function may be stored in an array and modified for later use. POINT also allows anything that can be displayed on the screen to be copied and/or modified.

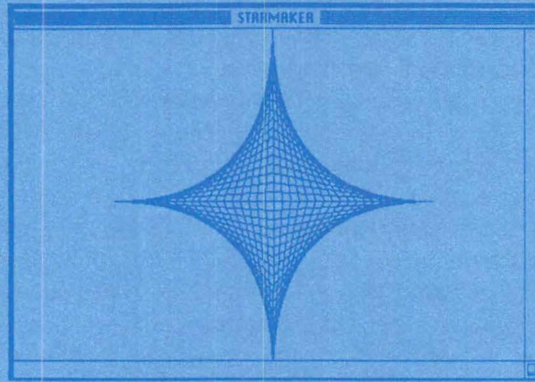
SUMMARY

Microsoft BASIC probably contains more graphics capabilities than any other version of BASIC currently available. While the graphics statements may be few, they are extremely powerful, each performing a multitude of different on-screen functions. When the Macintosh is coupled with Microsoft BASIC, just about anything imaginable can be displayed using proper programming technique.

Some readers may wonder why program graphics in BASIC when so much can be done with *MacPaint?* Certainly, *MacPaint* provides a very powerful graphics tool. Nine times out of ten, it is easier to draw an image with *MacPaint* than it is to program the same image in BASIC (or any other language, for that matter). However, *MacPaint* does not offer animation capabilities, and through programming you can set up an ever-changing graphic image on an endless loop with some randomizing built-in to display a moving kaleidoscope of pictures produced randomly by the computer. As a drawing tool, *MacPaint* is exceptional. It is not, however, a programming tool, and there are times when it is necessary to program to produce images.

More and more, graphics programs are coming to the front in the microcomputer industry. Don't ignore graphics programming. The old days of text-only programming are just about over, regardless of who you're programming for. Business programs that were once relegated to strictly text modes now offer a wealth of graphics displays because we use graphics every day of our lives, be it on billboards, business charts, or whatever. Graphics programming is easy with the advances made in computer languages, so why not show information with a single picture?

Chapter 8



Macintosh Filekeeping

In addition to its other excellent attributes, the Macintosh can be quickly and easily set up to perform filekeeping for many different applications. All files are stored on disk and once *written* there, can then be added to or *appended*. Many persons shy away from filekeeping, thinking it is a complex task better suited to systems level programming. However, nothing could be further from the truth, especially when you couple Microsoft BASIC with the Macintosh. Certainly, filing programs can be quite complex when it's necessary to sort information. However, as is always the case in computer programming, the complex programs are made up of a number of simple filing routines activated in logical order. This chapter will introduce you to Macintosh filekeeping. By starting with a few basic building blocks and proceeding from there, you will be amazed at how quickly you can be comfortable with writing such programs.

Filekeeping is one of the most useful attributes of all personal computers. Such programs are usually non-specific. This means that you can use a single filekeeping routine to store names, addresses, phone numbers, or any data you can imagine.

FILE READING PROGRAM

For this discussion, let's assume that a file of data has been written to disk. We will later learn how to set up such a file and write items to the file, but for now, assume it's already there. In order to access a file, you must first open it. In BASIC, this is done using the OPEN statement. This program shows the use of the OPEN statement in line 70. Here, the OPEN statement is followed by the name of a file. This name is enclosed in quotation marks, and in this example, is called "FILE.FIL". The .FIL designation is not necessary, but is often used to distinguish a data file from a BASIC program file. When INPUT is used with the OPEN statement, a file is opened specifically for reading. We must also specify a file number which in this case is represented by #1. Line 70 in this program tells the

Listing 8-1. File Reading.

```
10 REM FILE READING PROGRAM
20 REM PROGRAM READS SEQUENTIAL FILES
30 REM FILE MUST ALREADY BE CREATED
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 WIDTH 50
60 CLS
70 OPEN "FILE.FIL" FOR INPUT AS #1
80 INPUT #1,A$
90 PRINT A$
100 CLOSE #1
```

computer to open "FILE.FIL" as file #1 and that the file is opened for reading.

Line 80 uses the INPUT# statement to read data from the file. This statement is slightly different from the INPUT statement. The pound sign identifies it as the statement to access file information. This statement is followed by a comma and A$. A$ will be used to hold the first data item from the file. Line 90 then prints A$ to the screen. Line 100 uses the CLOSE statement to close the file. The file must be closed before moving on to other program lines that will be included in a standard filekeeping routine. The sequence to follow is delineated as follows:

1. Open the file for input.
2. Get an item from the file using INPUT#.
3. Print the item to the screen.
4. Close the file.

If you assume that FILE.FIL exists and contains one item, for example, the word COMPUTER, when this program is run, the file will be opened for input (reading) and the word COMPUTER will be assigned to A$ in line 80. Line 90 will then print COMPUTER to the screen, and line 100 will close the file.

ADVANCED FILE READING PROGRAM

This is the same program as the file reading program, only another line has been added. Line 120 branches to line 80 after the value of A$ has been printed to the screen. Line 80 contains the EOF function. This stands for END OF FILE. The number 1 is used in parentheses following EOF to indicate that this function is testing to see if the end of file #1 has been reached. The EOF function branches to line 130 when there are no more items in the file to be read. Since line 120 continues branching back to a portion of the program prior to the use of the INPUT# statement, the file items are each read in sequential order, and then each displayed on the screen, until line 80 detects the end of the file. The subsequent branch then closes the file. If there were no EOF functions in line 80, an error message would be displayed when there were no more items to read.

FILE WRITING PROGRAM

This simple program will help you create a data file. Line 60 assigns A$ the value FILE ITEM. For now, this is just a simple phrase to demonstrate file writing, but later, you will be able to assign any value you want to A$. Line 70 clears the screen.

Listing 8-2. Advanced File Reading.

```
10 REM  SECOND FILE READING PROGRAM
20 REM PROGRAM READS SEQUENTIAL FILES
30 REM FILE MUST ALREADY BE CREATED
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 WIDTH 50
60 CLS
70 OPEN "FILE.FIL" FOR INPUT AS #1
80 IF EOF(1) THEN 130
90 INPUT #1,A$
100 PRINT A$
120 GOTO 80
130 CLOSE #1
```

Line 80 uses the OPEN statement again to open a file called FILE.FIL. However, this time, the file is opened for output. This means that there will be output from the computer to the file; you are going to write information to a disk file. Again, the file is called #1.

At this point, the computer has opened the file and is ready to write information. The information we write to the file will be contained in A$. To accomplish the actual write, we use the PRINT# statement, which should not be confused with PRINT. Line 90 tells the computer to write the value contained in A$ to the file "FILE.FIL". Line 100 then closes the file. Congratulations! You have just written to a sequential file. If you run either of the two file reading programs, the phrase "FILE ITEM" will be displayed on the screen, because this is the information in FILE.FIL.

Listing 8-3. File Writing.

```
10 REM FILE WRITING PROGRAM
20 REM PROGRAM WRITES SINGLE ITEM TO FILE
30 REM COPYRIGHT FREDERICK HOLTZ 2/84
40 WIDTH 50
50 CLS
60 A$="FILE ITEM"
70 CLS
80 OPEN"FILE.FIL" FOR OUTPUT AS #1
90 PRINT#1,A$
100 CLOSE #1
```

ANOTHER FILE WRITING PROGRAM

There is a way to improve the file writing program. You can insert the name of the file you want to open and write to and have the program loop to continually accept items for the file. Here, you're not limited to a specific filename, nor to a single file item. You can open as many files as you like by running the program over and over again, and you can store as many items as you want in each file.

Line 70 prompts you to enter the name of the file you wish to open. This name is assigned to string variable FI$. The screen is then cleared and line 90 uses the OPEN statement to open the file named in FI$ for OUTPUT (writing) as #1. Line 100 then prompts you to type the file item and assigns the data to A$. Line 110 is an exit line. When you type END, this is an indication that you've finished entering items and wish to close the file. Any other input in line 100 will result in line 120 being executed. This line prints the value of A$ to the sequential file. When you type END, there is a branch to line 150, which closes the file.

Try running this program and insert ten or more items. Then run Listing 8-2 and read them all out to the screen again. You have now seen how sequential files are read and written.

Whenever you open a file to be read that doesn't exist, an error message will be displayed on the screen telling you that the file was not found. There is no real way you can get into trouble here. However, when you open a file for OUTPUT (writing), the computer will create that file on disk. Therefore, you must be careful because if you open a file for OUTPUT that already exists, the computer writes your new information over your old information, destroying the old file. It's like recording a new song on an old tape—you'll record the new song, but wipe out the old one.

Listing 8-4. More File Writing.

```
10 REM ADVANCED FILE WRITING PROGRAM
20 REM PROGRAM OPENS A USER NAMED FILE FOR WRITE
30 REM PROGRAM ALLOWS FOR INSERTION OF NUMEROUS ITEMS
40 REM TYPE "END" TO EXIT ITEM INPUT SEQUENCE
50 REM COPYRIGHT FREDERICK HOLTZ 2/84
60 CLS
70 INPUT"NAME OF FILE";FI$
80 CLS
90 OPEN FI$ FOR OUTPUT AS #1
100 INPUT "FILE ITEM";A$
110 IF A$="END" THEN 150
120 PRINT #1,A$
130 CLS
140 GOTO 100
150 CLOSE #1
160 CLS
170 END
```

FILE APPENDING PROGRAM

Suppose you open a file for OUTPUT and write several items to it. Suppose that you then wish to go back to the same file and add more items. Obviously, you can't open it for output again, as this will erase the items it currently contains.

The program in Listing 8-5 shows you how to add items. You simply open the file for APPEND. This is an indication that you wish to open a file that already exists in order to add items to the end of the list. This program helps you add one item to the end of a file named FILE.FIL, which already contains items. Line 60 contains the item to be written, which for this example, is named SECOND FILE ITEM. Line 80 opens the file in the APPEND mode, and line 90 uses the PRINT# statement to write the value to the end of the file. If you wish to be able to name your file and add many items to the end of the current list, simply go back to Listing 8-4 and change the word OUTPUT in line 90 to APPEND.

To summarize, the information discussed to this point opens a file for input and allows it to be read. Opening a file for output creates the file and allows you to write information to it. Opening it for output again erases the contents of the file. Opening a previously written file for append allows you to add information to the end of the current item list contained in the file. If you open a file that doesn't exist for APPEND, the file is created just like it would be if it were opened for output (writing).

COMPLETE FILING PROGRAM

This is a complete filing program that will allow you to read, write, or append items in a disk file. Lines 50 through 90 display a program menu on the screen. A menu is a selection of things you can ask this program to do. The four menu selections are:

1. Open file for write.
2. Open file for read.
3. Open file for append.
4. End file program.

Line 100 uses the INKEY$ variable to test your input from the keyboard. In lines 110 through 150, you can see the various branches that are designated for the different keyboard inputs.

If you select 1 from the menu, line 120 branches to line 170. Here, the screen is cleared and line 180 prompts you to input the name of the

Listing 8-5. File Appending.

```
10 REM FILE APPEND PROGRAM
20 REM PROGRAM ADDS ITEM TO THE END OF AN EXISTING FILE
30 REM IF FILE  DOES NOT ALREADY EXIST,  IT IS CREATED
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 WIDTH 50
60 A$="SECOND FILE ITEM"
70 CLS
80 OPEN "FILE.FIL" FOR APPEND AS #1
90 PRINT #1,A$
100 CLOSE #1
```

Listing 8-6. Complete Filing.

```
10  REM SEQUENTIAL FILING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 WIDTH 40
40 CLS
50 PRINT "FILE MENU"
60 PRINT "1. OPEN MENU FOR WRITE"
70 PRINT "2. OPEN FILE FOR READ"
80 PRINT "3. OPEN FILE FOR APPEND"
90 PRINT "4. END FILE PROGRAM"
100 K$=INKEY$
110 IF K$="" THEN 100
120 IF K$="1" THEN 170
130 IF K$="2" THEN 330
140 IF K$="3" THEN 420
150 IF K$="4" THEN 550
160 GOTO 100
170 CLS
180 INPUT "NAME OF FILE";FI$
190 CLS
200 PRINT"WARNING! IF ";FI$;" ALREADY EXISTS"
210 PRINT"IT WILL BE ERASED."
220 PRINT
230 INPUT"DO YOU WISH TO OPEN THIS FILE (Y/N)";W$
240 IF W$="Y" OR W$="y" THEN 250 ELSE 40
250 CLS
260 ON ERROR GOTO 580
270 OPEN FI$ FOR OUTPUT AS #1
280 INPUT"FILE ITEM";A$
290 IF A$="END" THEN 520
300 CLS
310 PRINT#1,A$
320 GOTO 280
330 CLS
340 INPUT"NAME OF FILE TO BE READ";FI$
350 CLS
360 ON ERROR GOTO 580
370 OPEN FI$ FOR INPUT AS #1
380 IF EOF(1) THEN 520
390 INPUT#1,A$
400 PRINT A$
```

```
410 GOTO 380
420 CLS
430 INPUT"NAME OF FILE TO BE APPENDED";FI$
440 CLS
450 ON ERROR GOTO 580
460 OPEN FI$ FOR APPEND AS #1
470 INPUT"FILE ITEM";A$
480 IF A$="END" THEN 520
490 CLS
500 PRINT#1,A$
510 GOTO 470
520 CLOSE #1
530 INPUT"PRESS RETURN FOR MAIN MENU";ER$
540 GOTO 40
550 CLS
560 PRINT"PROGRAM TERMINATED"
570 END
580 CLS
590 PRINT"ERROR:FILE CANNOT BE OPENED"
600 FOR DLAY=1 TO 1500:NEXT DLAY
610 CLEAR
620 GOTO 40
```

file to open or to create for writing. Line 200 prints a warning message stating that if the file you named already exists, it will be erased. You are then given the option to continue or return to the menu and make another selection. Assuming you wish to continue, the screen is cleared in line 250. Line 260 contains an ON ERROR statement. This is an error trapping routine. The ON ERROR statement detects a problem where the file could not be written, such as when the disk is full, and if there is a problem, there is a branch to line 580, which clears the screen to receive the error message in line 590. Once the message has been printed, there is a slight delay. Line 620 then branches back to the start of the menu print routine.

If there is no error, lines 270 through 320 are executed. These lines are nearly identical to those contained in Listing 8-4. This is the routine used to write information to the file.

If you selected menu option 2, there is a branch to line 330, where you input the name of the file to be read. This name is assigned to FI$. Lines 360 through 410 contain a close copy of the file reading program (Listing 8-2). If you select menu item 3 the program branches to line 420, which accesses the program to add items to a file. When you select menu option 4, the branch is to line 550, where the PROGRAM TERMINATED message is printed and the program ends.

There you have it—a complete filekeeping program that can be typed into your Macintosh in about 20 minutes and can be used over and over again to convert your disk into a filing cabinet.

FILE ITEM SEARCH PROGRAM

Assume that you have set up a file that contains

Listing 8-7. File Item Search.

```
10 REM FILE ITEM SEARCH PROGRAM
20 REM PROGRAM ALLOWS THE SEARCH
30 REM OF A FILE FOR A SPECIFIC ITEM
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 WIDTH 50
60 CLS
70 INPUT"INPUT NAME OF FILE TO BE SEARCHED";FI$
80 CLS
90 INPUT"INPUT THE SEARCH ITEM";SI$
100 CLS
110 OPEN FI$ FOR INPUT AS #1
120 IF EOF(1) THEN 160
130 INPUT #1,A$
140 IF A$=SI$ THEN PRINT A$:COUNT=COUNT +1
150 GOTO 120
160 IF COUNT = 0 THEN 170 ELSE 180
170 PRINT SI$;" WAS NOT FOUND IN ";FI$
180 CLOSE #1
```

several hundred items and you want to pull one of these items from the file for examination. It's very inefficient to display the entire contents of the file. It would be best if you could have the computer sort through all of the items and pull out the one you're looking for. The program in Listing 8-7 will do just that. In its present form, you have to type in the name of the item you're looking for exactly as it appears in the file. However, this program will be modified later to take partial items. For now, consider it a practicum in file searching.

Line 70 asks you to input the name of the file to be searched. You must type file name exactly as it appears on the disk listing. The screen is then cleared and line 90 asks you to input the item you are searching for. This must be typed exactly as it appeared when first entering it into the file. Again, the screen is cleared and line 110 opens the file you named for input (reading). Line 130 assigns the first item in the file to A$. Line 140 does the actual search. It compares the item you just read with the item you are looking for. If the two match, the item is printed to the screen. If the item you are looking for is contained in the file in five different locations, it will be printed to the screen five times.

On the other hand, if the item is never found COUNT will never be incremented and since it was not previously assigned a value, it will be equal to 0. When line 120 detects the end of file condition, the branch is to line 160, which tests the value of COUNT. If COUNT is equal to 0, line 170 is executed. It prints the item you are looking for and tells you that it wasn't found in the file you named. Line 180 is then executed, which closes the file. If the item was found, COUNT would be equal to a number larger than 0, so line 170 is branched over and the file is closed in line 180.

PARTIAL ITEM FILE SEARCH

This program is an enhanced version of the

previous one, but it is far more valuable because it will allow you to search for and display an item contained in a file without having to input that item in its entirety. For example, assume that a file contains the item

CHARLIE JONES OWES ME $50.00

but you don't know where. Wouldn't it be nice to be able to search that file by simply inputting Charlie Jones' name in order to see his record? This program will allow you to do just that. When the program is run, just type the name CHARLIE JONES and the above phrase will be found and displayed. You could also search for CHARLIE, JONES, or even CHA and the file item would still be displayed.

What this program does is take the search item and overlay it with the items it reads from the file. If you input CHA, the computer will overlay these characters with each part of every file item. When it obtains a match, the matching file is printed. If you simply input a search item of C, any file that contains a C will be displayed on the screen.

Line 90 prompts you to input the search item. This is assigned to SI$. SI is assigned the value of the length or the character count in SI$ with the LEN function. Line 120 opens the file you named for reading and line 140 grabs the first file item. Line 150 assigns to L the length of the file item just read. Line 160 begins a FOR-NEXT loop that counts from

Listing 8-8. Partial Item File Search.

```
10 REM ITEM PORTION SEARCH PROGRAM
20 REM PROGRAM WILL SEARCH FILE FOR
30 REM AN ITEM OR PORTION OF AN ITEM
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 WIDTH 50
60 CLS
70 INPUT"INPUT NAME OF FILE TO BE SEARCHED";FI$
80 CLS
90 INPUT"INPUT THE SEARCH ITEM";SI$
100 SI=LEN(SI$)
110 CLS
120 OPEN FI$ FOR INPUT AS #1
130 IF EOF(1) THEN 210
140 INPUT #1,A$
150 L=LEN(A$)
160 FOR X=1 TO L
170 T$=MID$(A$,X,SI)
180 IF T$=SI$ THEN PRINT A$:COUNT=COUNT+1
190 NEXT X
200 GOTO 130
210 IF COUNT=0 THEN 220 ELSE 230
220 PRINT SI$;" WAS NOT FOUND IN ";FI$
230 CLOSE #1
```

1 to the length of A$(L). Line 170 assigns to T$ the value of what is found in A$ at position X and for the length of SI. You will remember that SI is the character count of the item you are searching for. As the loop cycles, T$ simply grabs the required number of letters, starting with the first letter in the item, then starting with the second letter, the third, and so on, until a match is found or the characters in the item are exhausted. Say you were searching for the Charlie Jones item.

CHARLIE JONES OWES ME $50.00

Assume that for a search item, you simply typed in JONES. The number of characters in JONES is 5. Therefore, SI equals 5. When the computer comes to the file item you are looking for, here's how the sequence will go.

1. The word JONES is compared with the first five characters in A$.
2. There is no match here, since the first 5 characters in our example are CHARL.
3. When the loop cycles again, X is equal to 2. Therefore, T$ in line 170 is assigned the value of the 5 characters in A$ starting with the second character.
4. There is still no match, because the search item is JONES and the 5 characters that are compared with this starting at position 2 are HARLI.
5. The loop continues to advance until it finally gets to the 8th character position of A$. Here, the five characters are JONES, and match the search item.

When a match occurs, line 180 determines that T$ is equal to SI$ and the entire contents of A$ (the entire file item) are printed to the screen. The MID$ function is for manipulating and searching text. With it, we have been able to find a file item without having to type in the entire contents of the item.

SUMMARY

So, as you can see, file keeping is not at all complicated. A few simple routines combine to form a complete file keeping system that can be used for any purpose.

You will find that being able to store data in your computer and retrieve it as needed will open up new dimensions in the way you use your Macintosh. Now you can keep records of every type of disk.

In time, you will find yourself developing more sophisticated filekeeping programs, tailored to store certain information and display it in particular ways. Here, we've taken the first step—the rest is up to you.

Chapter 9

Programming the Mouse

A mouse is simply another type of input device or user interface, just like a keyboard, light pen, or joystick. The mouse most closely resembles the joystick. Almost everyone today is familiar with the joystick. It is the hand-controlled device included with most video games that allows the user to control on-screen object animation. A mouse is just a modified joystick that inputs information to the computer by its position on a table surface instead of by the position of a lever.

Unlike a joystick that moves an on-screen cursor in the direction the lever is pointing as long as the lever is held in that position, the mouse moves across the table in the direction the cursor should move. A small ball within the mouse tracks across the surface of the desk or tabletop and tells what direction to move the cursor. In other words, the cursor moves as long as the mouse moves in the same relative direction. For this reason, a mouse is generally easier to use than the joystick.

Although generally a joystick can provide a little more accuracy in pinpointing precise locations on the screen, the Apple Macintosh mouse is an extremely smooth performer. It tracks very well and quite precisely. The rubber ball that mounts in the bottom cover of the mouse can be easily removed for cleaning by rotating the circular disk clip that holds it in place. By rotating the disk counterclockwise, it will slip out of the mouse cover and the ball can be dropped out, so both the ball and the well that holds the ball can be cleaned. Dust can accumulate in its well, so it's a good idea to clean the mouse about once a week. The Macintosh manual provides complete instructions for performing this preventive maintenance.

Some persons think there is something magic about a mouse, but it's really far simpler than most other types of input devices. When running *MacPaint* or *MacWrite*, the mouse is easy-to-use when accessing different icon tools and positioning the cursor at different points on the screen. When the computer is first activated, programs in the

Macintosh ROM are automatically loaded into current memory, which allows the computer to read information returned by the position of the mouse. Fortunately, Microsoft BASIC allows us to write our own special programs that can make use of the mouse in different ways.

THE MOUSE FUNCTION

In Microsoft BASIC, we use the MOUSE function to perform seven different programming functions, depending on the argument used. Here, the MOUSE function is just like most other functions in Microsoft BASIC. It returns a specific value to a numeric variable when executed. Its format is

X = MOUSE(N)

where N is a value of 0 to 6. Depending on the number substituted for N, the MOUSE function may return different information about the status of the mouse button or the position of the mouse pointer on the screen.

MOUSE(0). When the MOUSE function is used with a 0 subscript or *argument,* it returns a value ranging from -3 to 3. Each of the seven possible values indicates the condition of the mouse button. The format is:

X = MOUSE(0)

When this line is executed, a value of from -3 to +3 is returned to X, depending on the status of the mouse button. If X is equal to 0 when this line is executed, this means that the mouse button is not currently being pressed and has not been pressed since the last time this function was executed. If the number 1 is returned to X, the mouse button is still not currently down, but it has been clicked once since the last time the function was executed.

When X is equal to 2, the button is not down, but has been clicked twice. A 3 indicates that it was clicked three times (as in a third level selection).

When X is equal to -1, this means the mouse has been clicked once and the button is still being held down. A -2 indicates two clicks with the button held down, and a -3 indicates three clicks with the button held down.

What is the purpose of this information? The purpose is to tell the computer the exact status of the mouse button and to have it execute certain subroutines depending on the status. The program shown in Listing 9-1 keys on a condition of the mouse being clicked once. Line 30 clears the screen, and the MOUSE(0) function is used in line 40 to assign to X the value of the mouse button condition. Don't be confused here. MOUSE(0) determines the condition of the mouse button, and the variable X is assigned that value. Line 60 uses an IF-THEN statement to test for a condition of X being equal to 1, which would indicate a single click of the mouse button. If MOUSE(0) returns any number other than 1, the ELSE portion of this statement simply branches back to line 40, where the condition of the mouse is read again. When X is

Listing 9-1. A Program That Prints a Message When the Mouse Is Clicked Once.

```
10 REM BASIC MOUSE BUTTON PROGRAM(ONE CLICK)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(0)
60 IF X=1 THEN PRINT"YOU PRESSED THE MOUSE BUTTON." ELSE 40
70 END
```

Listing 9-2. The Message Is Now Printed by Clicking the Mouse Twice.

```
10 REM BASIC MOUSE BUTTON PROGRAM(TWO CLICK)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(0)
60 IF X=2 THEN PRINT"YOU PRESSED THE MOUSE BUTTON TWO TIMES." ELSE 40
70 END
```

equal to 1, the phrase "YOU PRESSED THE MOUSE BUTTON" is displayed on the screen. Try this program and you will find that anytime you hit the mouse button, the message will be displayed on the screen. You must simply tap the mouse button and then release it in order to print the message. If you press once and hold, the message will not be displayed until you release the button. This is due to the fact that we are testing for a condition of X being equal to 1 and it will only be equal to 1 when the mouse button is clicked and then released.

The program in Listing 9-2 performs in the same manner as the one in Listing 9-1, only in this case, line 60 tests for a condition of X being equal to 2. This is the number returned to X when the button is clicked two times and then released. If you press the mouse button once and release, nothing happens. When it's clicked twice, the message is displayed.

Listing 9-3 shows the same type of program set up to respond to three clicks of the mouse. Note that all three programs are extremely simple and do not present any unique programming problems. Line 40 does it all by assigning to X the mouse button condition. As long as we know what numbers are returned based upon the various conditions the button can assume, programming is quite simple.

Listing 9-4 shows a program set up to test for a condition of X being equal to -1, which means that the mouse button has been clicked once and held. This program is a bit different from the previous three, as it contains an extra line. Line 50 is a delay loop. The delay is required because we're trying to set up a sequence whereby the message will not be displayed unless the mouse button is clicked once and held. Without this loop, line 40 is being executed so rapidly within the loop (ELSE 40) that it's almost impossible to click and release the mouse button before the MOUSE(0) condition is read. By slowing down the execution, line 40 is not executed as rapidly, because it takes longer for the branch at the end of the line 60 to be activated. Therefore, if you simply clicked and released the mouse button, a 1 would be returned. If line 40 were being executed

Listing 9-3. A Program Where When the Mouse Is Clicked Three Times, X is Equal to 3, and the Message is Printed.

```
10 REM BASIC MOUSE BUTTON PROGRAM(THREE CLICK)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(0)
60 IF X=3 THEN PRINT"YOU PRESSED THE MOUSE BUTTON THREE TIMES."ELSE 40
70 END
```

Listing 9-4. A Program That Displays the Message When the Mouse Button Is Clicked and Held.

```
10 REM BASIC MOUSE BUTTON PROGRAM(ONE CLICK HOLD)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(0)
50 FOR DLAY=1 TO 750:NEXT DLAY
60 IF X=-1 THEN PRINT"YOU ARE PRESSING THE MOUSE BUTTON." ELSE 40
70 END
```

faster than you could click and release, everytime you pressed the mouse button, a -1 would be returned. This happens because you cannot click and release faster than the computer can read the value of X. The time delay loop gives the user time to click and hold (-1) or to click and release (1). Since we're looking for -1, the message will only be printed when you click and hold the mouse button. Try running this program as it appears and then remove the time delay loop and try it again. You will find that when the time delay loop is removed, every time you press the button, whether you hold or not, the message will be printed.

The program in Listing 9-5 allows you to see the numbers that are returned to X for the number of times you click the mouse button and whether or not it is currently up or down. This is just like the previous program, except there is no IF-THEN-ELSE statement. Line 40 assigns to X the value of the condition of the mouse button, while line 41 prints the value of X on the screen. Line 45 is a time delay loop, while line 60 simply branches to line 40 to again test for the condition of the mouse button.

When you run this program, you will immediately see a column of zeros being printed on the left side of the screen. This means that the mouse button is not being pressed. Therefore, X is equal to 0. If you click the button once, the number 1 will be displayed. Two or three clicks will return the numbers 2 and 3. If you click once and hold, a -1 will be printed on the screen. This program demonstrates the many conditions possible when monitoring the mouse button.

MOUSE(1). The mouse programs shown thus far address only the condition of the mouse button. The MOUSE function can also tell the current, starting, or ending X,Y coordinates, but cannot do this until the MOUSE(0) function is executed first. Therefore, in every program that uses any MOUSE function, you will always see at least one reference to MOUSE(0).

The MOUSE(1) function determines the cur-

Listing 9-5. A Program That Prints the Number That Represents the Mouse Button Condition.

```
10 REM BASIC MOUSE BUTTON PROGRAM(VARIABLE CLICKS AND HOLDS)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(0)
41 PRINT X
45 FOR DLAY=1 TO 750:NEXT DLAY
60 GOTO 40
```

Listing 9-6. A Program That Prints the X Coordinate Value of the Mouse Pointer Location.

```
10 REM DEMONSTRATION OF MOUSE(1)
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 X=MOUSE(1)
60 PRINT X
70 GOTO 40
```

rent X coordinate. This is the value of the horizontal coordinate of the screen location of the mouse pointer at the time the MOUSE(0) function was executed. The program in Listing 9-6 provides an explanation.

Line 40 assigns to B the condition of the mouse button. Line 50 assigns to X the horizontal coordinate of the mouse pointer. Line 60 prints the value of the horizontal coordinate on the screen. Line 70 then branches back to line 40, where MOUSE(0) is executed again, followed by MOUSE(1).

We had to use MOUSE(0) in this program to get MOUSE(1) to work. In this particular example, although the value of B is never used, this line must be executed before MOUSE(1) is. You don't have to press the mouse button when using this program, since which state the mouse is in makes no difference whatsoever.

When you run the program, a column of numbers appears on the left side of the screen. Move the mouse across the desk and you will see the numbers change. The last number in the column represents the current horizontal position of the mouse pointer on the Macintosh screen. If you place the pointer toward the center of the screen, you will get a value of about 240. Its vertical position on the screen makes no difference, since MOUSE(1) only returns the value of the horizontal coordinate.

This program is a bit awkward to use since it continuously prints a series of coordinates. The program shown in Listing 9-7 is a little more practical. The program is very similar to the one in Listing 9-6, but here the value of B which indicates the mouse button state *is* used. This program will print the X coordinate of the mouse pointer anytime you press the mouse button, which eliminates the never-ending column. Line 60 tests for a condition of B being equal to any number other than 0. Remember, when MOUSE(0) returns 0, this means

Listing 9-7. A Program That Displays the X Coordinate Value When the Mouse Button Is Clicked.

```
10 REM BASIC MOUSE X-COORDINATE PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 X=MOUSE(1)
60 IF B<>0 THEN PRINT"X-COORDINATE =";X
70 FOR DLAY=1 TO 1000:NEXT DLAY
80 GOTO 40
```

Listing 9-8. A Program That Prints the Y Coordinate Value When the Mouse Button Is Clicked.

```
10 REM BASIC MOUSE Y-COORDINATE PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 Y=MOUSE(2)
60 IF B<>0 THEN PRINT"Y-COORDINATE =";Y
70 FOR DLAY=1 TO 1000:NEXT DLAY
80 GOTO 40
```

the mouse button has not been activated. Run this program and place the mouse pointer at any location on the screen. Click the mouse button and the X coordinate value will appear on the screen. Line 70 is the time delay loop again, which prevents two or three coordinate values from being displayed for each click of the mouse.

MOUSE(2). The MOUSE(2) function is very similar to MOUSE(1), except here, a value will be returned that indicates the mouse pointer position in the Y or vertical plane. The program shown in Listing 9-8 is nearly identical to the program in Listing 9-7, but here, line 50 assigns to the Y variable the number returned by the MOUSE(2) function.

The program shown in Listing 9-9 combines the last two programs into one. Here, the values of both the X and Y coordinates will be displayed anytime you click the mouse button. To achieve this, line 45 was added to Listing 9-8 to assign to X coordinate value, and line 60 has been changed to print the values returned by both MOUSE(1) and MOUSE(2).

Now, what's the value of all that? You can now use the mouse button and determine its X,Y coordinates. With this in mind, you now have the capability of supplying needed information to other programs that can draw objects on the screen, display text, or perform many other operations at a point on the screen specified by the mouse pointer. The program shown in Listing 9-10 uses the same principals to allow you to draw on the screen using the pencil icon just as you would with *MacPaint*. When you run this program, you can position the mouse pointer on the screen, click, hold, and move the mouse while drawing a line. In this example, J

Listing 9-9. A Program That Returns Both the X and Y Coordinate Values When the Mouse Button is Clicked.

```
10 REM BASIC MOUSE X,Y-COORDINATE PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
45 X=MOUSE(1)
50 Y=MOUSE(2)
60 IF B<>0 THEN PRINT"X =";X;"Y =";Y
70 FOR DLAY=1 TO 1000:NEXT DLAY
80 GOTO 40
```

Listing 9-10. A Simple Drawing Program. When the Mouse Button is Held, a Line is Formed.

```
10 REM MOUSE DRAWING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 X=MOUSE(1)
50 Y=MOUSE(2)
60 J=MOUSE(0)
70 IF J<0 THEN CALL LINETO(X,Y):GOTO 40
80 CALL MOVETO(X,Y):GOTO 40
```

represents the button value. As long as it is less than 0, this means the mouse button is being held down. As long as the button is held down, a line is drawn dot by dot at the coordinates specified by X and Y. These values are returned in lines 40 and 50.

CALLING A ROM SUBROUTINE

The program in Listing 9-10 uses the LINETO call, which is a subroutine contained in the Macintosh ROM. These subroutines are listed in the back of the Microsoft BASIC manual and address the many built-in functions called by the *MacPaint* program. These subroutines are always preceded by CALL in Microsoft BASIC.

CALL is a statement that does just what its name implies. It calls on assembly language subroutines, or, in this particular case, an internal Macintosh subroutine. LINETO-CALL works very much like the LINE statement discussed previously, but it works faster because it is run by the Macintosh ROM. Line 80 uses another Macintosh ROM routine called MOVETO. This simply places the graphic cursor at the point determined by the values of X and Y. MOVETO does not draw a line; it simply positions the graphic cursor. LINETO draws a line from the current position of the graphic cursor to the position determined by X and Y. The subroutines are aptly named, since LINETO draws a line, while MOVETO moves the cursor but does not draw a line.

MOVETO can also be used to position text at a specific point on the screen.

PUTTING THE MOUSE TO WORK

While there are other MOUSE functions to discuss, let's put what we've already learned to practical use. Listing 9-11 shows the MOUSE functions used to draw circles of a fixed size at any

Listing 9-11. A Basic Circle Drawing Program That Employs the Mouse.

```
10 REM MOUSE CIRCLE DRAWING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 X=MOUSE(1)
60 Y=MOUSE(2)
70 IF B<>0 THEN CIRCLE(X,Y),30
80 GOTO 40
```

point on the screen. Lines 40, 50, and 60 assign to B, X, and Y the mouse button condition, and horizontal and vertical coordinates, respectively. Line 70 tests for a condition of the mouse button being activated. When it is clicked or clicked and held, the CIRCLE statement is executed and draws a circle whose center is at the coordinates returned to X and Y. These coordinates are determined by the location of the mouse pointer at the time the mouse button is clicked. The CIRCLE statement is used with a radius of 30, so all of the circles produced will be that size.

Click and hold the mouse button and draw a series of circles up and down the screen. You must move the mouse slowly, however, to allow time for the computer to read new X,Y coordinates.

Figure 9-1 shows the circles displayed by this program when you click the mouse at various locations on the screen. Here, individual circles are displayed at the screen positions you specify. Figure 9-2 shows what happens when you press and hold the mouse button and then move the pointer across the screen. You get a "slinky" effect. If you pull slowly enough, the various circles are closely spaced. If you pull rapidly, you get an almost random spacing. There is a problem with this program, in that if you don't like what you draw, you have to manually halt the program using COMMAND/C and then run it again. This erases the screen and you have to start from scratch.

The program in Listing 9-12 uses the mouse button to clear the screen, so you don't have to manually halt the program. Line 70 is changed to include the logical operator OR. This simply tells the computer to draw a circle on the screen as long as B is not equal to 0 or 2. Line 75 tells the computer to clear the screen if B is equal to 2, so you can draw to your heart's content. When you want to clear the screen, simply click the mouse button twice and release. This will return a value of 2 to B, and line 75 will be executed.

These simple examples show how the MOUSE function can be put to practical use to display graphics. However, the three functions discussed thus far are only the beginning.

MOUSE(3) AND MOUSE(4). MOUSE(1) and MOUSE(2) return the current X and Y coordinates of the mouse pointer. MOUSE(3) and MOUSE(4) also return the X and Y coordinates, but these are not the current coordinates, but rather the starting coordinates. If you set your program up properly, MOUSE(3) and MOUSE(4) can be used to write programs that can replicate some of the programming tools found in *MacPaint.* When the mouse button is first pressed, MOUSE(3) and MOUSE(4) return to their variables the X and Y coordinates of the pointer. However, as long as the button is held down, the coordinates will remain the same. MOUSE(1) and MOUSE(2) will still return the coordinates of the mouse's current position. Using these together allows us to write effective move programs much like the circle and rectangle tools found in *MacPaint.* Listing 9-13 is a rectangle drawing routine that closely simulates the routine found in *MacPaint.*

Line 40 assigns to B the condition of the mouse button. For now, let's skip lines 50 and 60. Lines 70 and 80 use MOUSE(1) and MOUSE(2) to return the *current* X,Y coordinates. Lines 90 and 100 use MOUSE(3) and MOUSE(4) to return to XX and YY, the *starting* coordinates. Line 110 tests for a condition of B being equal to -1, which would indicate that the mouse button has been pressed and is still down. When this occurs, the Microsoft BASIC LINE statement draws a line from the starting coordinates (XX,YY) to the current coordinates (X,Y). The B (box) option is used, so an open box is drawn instead of a straight line. Line 120 branches to line 40, and another check is made to see if the mouse button is still down. If it's not, B becomes equal to 0. Line 50 detects this and branches to line 130, which resets all variables to 0. The image on the screen remains intact.

Line 60 uses the LINE statement again to draw

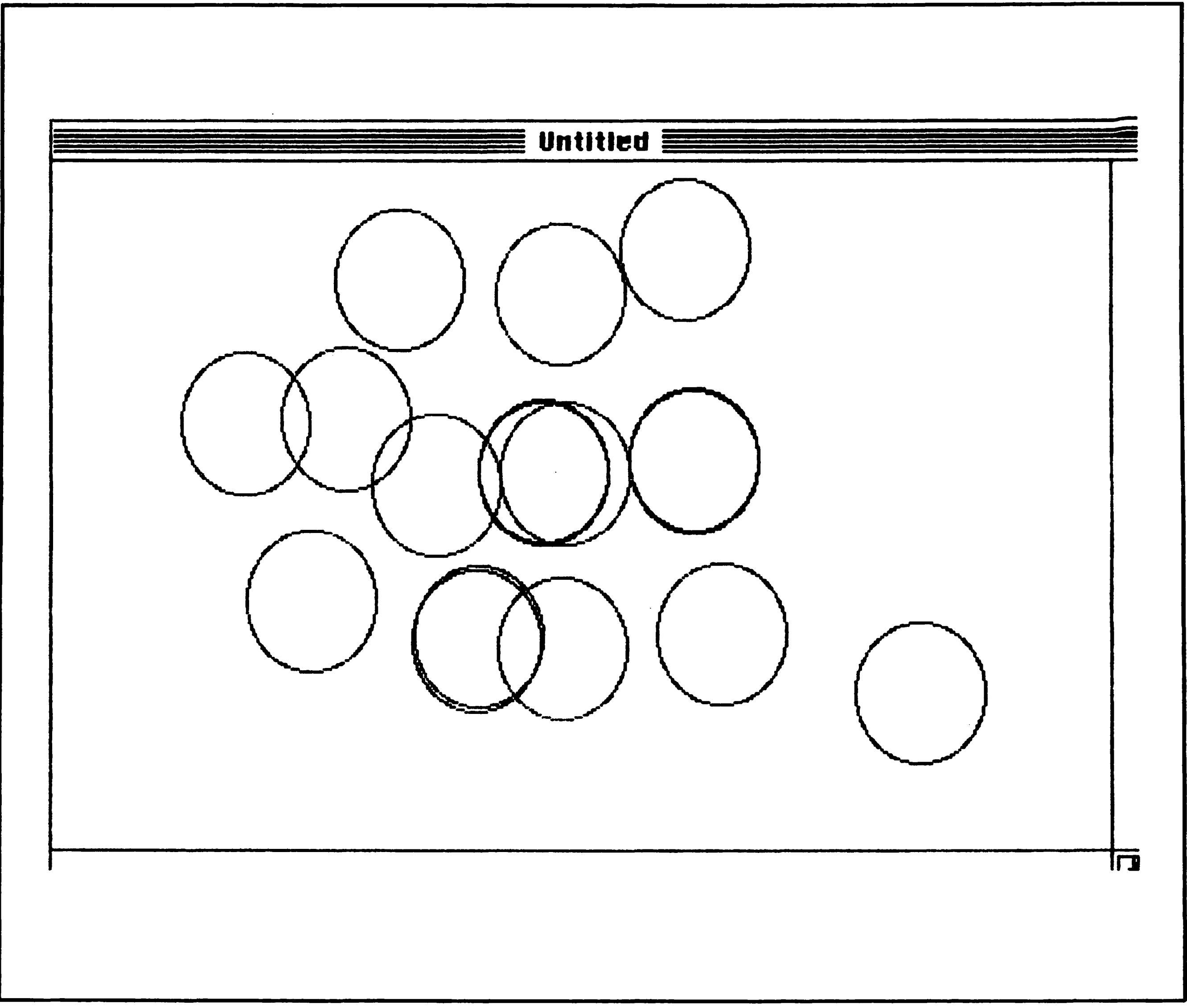


Fig. 9-1. A sample screen write done by the circle drawing program.

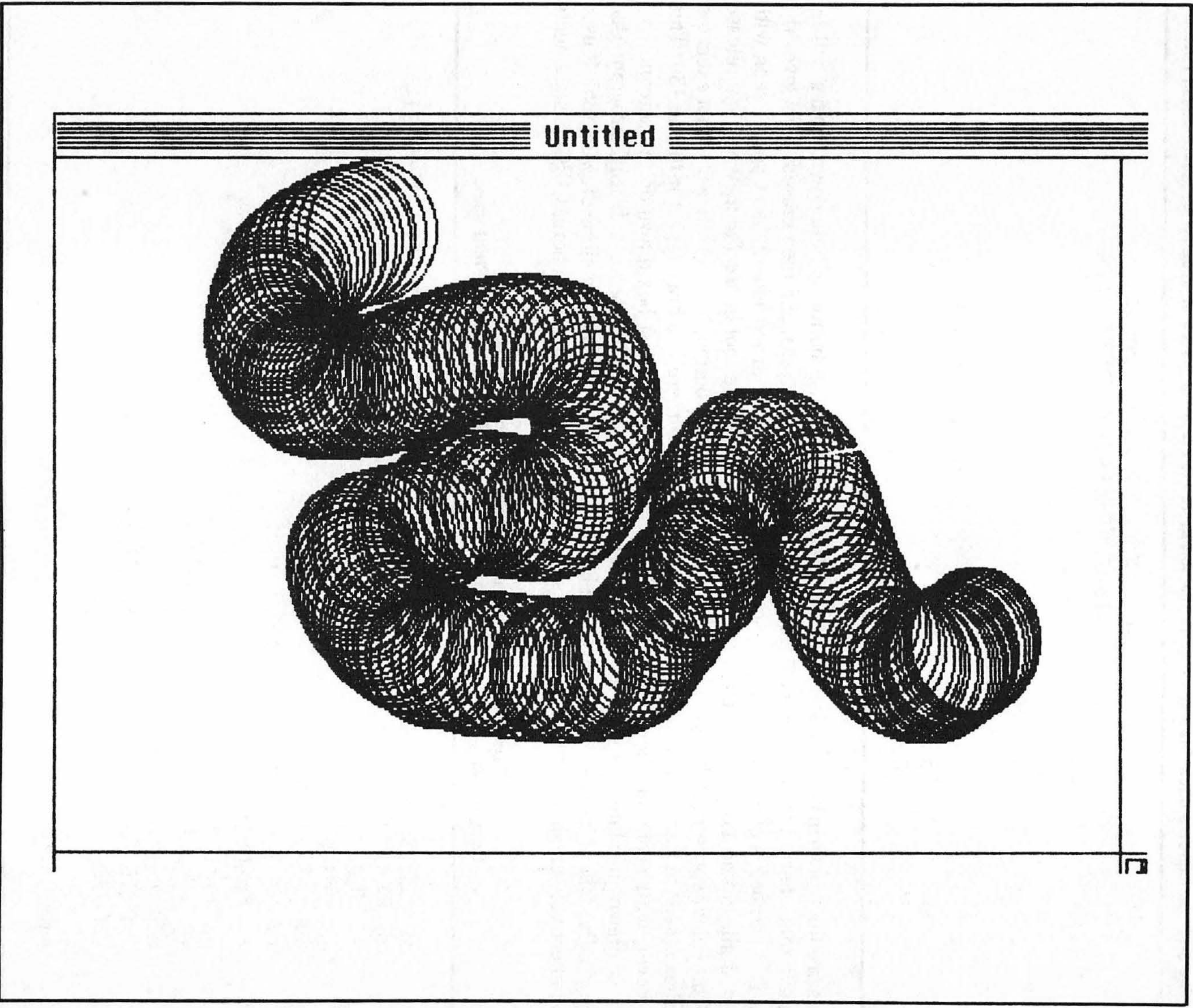


Fig. 9-2. Another example of how the basic circle drawing program can produce unusual screen images.

Listing 9-12. An Improved Circle Drawing Program That Allows You to Erase the Screen by Clicking the Mouse Button Twice.

```
10 REM MOUSE CIRCLE DRAWING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 X=MOUSE(1)
60 Y=MOUSE(2)
70 IF B<>0 OP B<>2 THEN CIRCLE(X,Y),30
75 IF B=2 THEN CLS
80 GOTO 40
```

the same line drawn in line 110. However, there's a big difference here. Line 60 uses a color number of 30. This number is optional in line 110, so it's not used at all. In line 110, the computer goes to the default color represented by 33. This is black, so the box is seen in black against a white background. However, with a color number of 30 in line 60, the box is drawn in white, so it is invisible. More importantly, this white box is drawn over the black one, effectively erasing it from the screen. As long as the mouse button is held down, boxes will be repeatedly drawn and then erased. This gives the rubber band or expanding effect you've seen with the rectangle tool in *MacPaint*. When you release the mouse button, however, line 60 is not executed because there is a branch past it to line 130. This preserves the last box drawn on the screen.

Listing 9-14 shows a BASIC program that emulates the *MacPaint* circle drawing tool. It uses the same principals, but here, CIRCLE statements

Listing 9-13. A Program That Draws Squares and Rectangles Using the Mouse. It Operates Much as One of the Graphics Tools in *MacPaint* Does.

```
10 REM SAMPLE MACPAINT RECTANGLE DRAWING ROUTINE IN MS-BASIC
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 IF B=0 THEN 130
60 IF B=-1 THEN LINE(XX,YY)-(X,Y),30,B
70 X=MOUSE(1)
80 Y=MOUSE(2)
90 XX=MOUSE(3)
100 YY=MOUSE(4)
110 IF B=-1 THEN LINE(XX,YY)-(X,Y),,B
120 GOTO 40
130 X=0:Y=0:XX=0:YY=0
140 GOTO 40
```

Listing 9-14. A Program That Emulates the *MacPaint* Circle Tool, but is Written Entirely in MS-BASIC.

```
10 REM SAMPLE MACPAINT CIRCLE DRAWING ROUTINE IN MS-BASIC
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 XX=MOUSE(3)
60 YY=MOUSE(4)
70 IF B=-1 THEN 80 ELSE 40
80 B=MOUSE(0)
90 IF B=-1 THEN CIRCLE(X,Y),H,30
100 X=MOUSE(1)
110 Y=MOUSE(2)
120 IF B=0 THEN 160
130 H=ABS(YY-Y)
140 IF B=-1 THEN CIRCLE(X,Y),H,33
150 GOTO 80
160 X=0:Y=0:XX=0:YY=0
170 GOTO 40
```

are substituted for the LINE statements. The radius of the circle is represented by H and is determined by subtracting the current Y coordinate from the starting Y coordinate represented by Y and YY, respectively. The ABS function (absolute value) is used in line 130 to always return a positive value for the radius. For instance, if YY is equal to 10 and Y is equal to 50, let's say, the value would be -40. However, the ABS function converts -40 to +40. Therefore, we always have a legal value to use for the circle radius.

Listing 9-15 shows a program listing that allows the keyboard to interface with the mouse. Here, you determine the point on the screen where text is to be written by placing the mouse pointer at that location and clicking once. You then type the word, phrase, or sentence via the keyboard and press RETURN. You can then move the pointer to another location and do the same. The MOVETO subroutine is used in lines 80 and 100 to position the cursor at the coordinates returned by MOUSE(1) and MOUSE(2). Line 110 uses the INKEY$ function to allow text to be entered. If nothing is entered via the keyboard, a loop is set up between lines 110 and 120. However, when any key is pressed, it is assigned to A$ and printed by line 150. Line 140 tests for the RETURN key being pressed. In ASCII code, this key is represented by decimal number 13, so CHR$ (13) represents the RETURN key. The CHR$ function returns the character equivalent of an ASCII number of the key pressed. When RETURN is pressed (A$ = CHR$(13)), line 140 branches to line 40 and the routine begins again, allowing you to position the mouse pointer at another location on the screen.

MOUSE(5) AND MOUSE(6). MOUSE(5) and MOUSE(6) work very much like MOUSE(3) and MOUSE(4), except these functions return the ending X and Y coordinates. If the button was down the last time MOUSE(0) was called, MOUSE(5) returns the horizontal coordinate at the time of this call. In this manner, it works very much like

Listing 9-15. A Program for Placing Text at Set Points on the Screen by Clicking the Mouse.

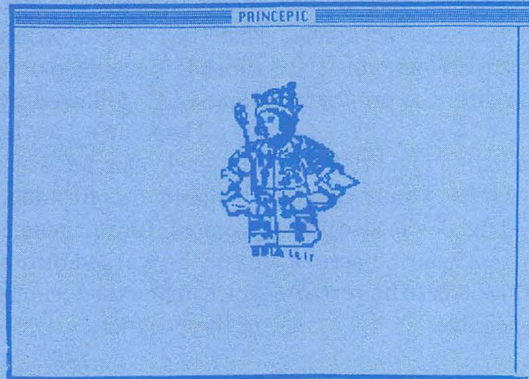
```
10 REM BASIC TEXT SETTING MOUSE ROUTINE
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 B=MOUSE(0)
50 X=MOUSE(1)
60 Y=MOUSE(2)
70 IF B<>0 THEN 80 ELSE 40
80 CALL MOVETO(X,Y)
90 PRINT CHR$(91)
100 CALL MOVETO(X,Y)
110 A$=INKEY$
120 IF A$="" THEN 110
130 B=MOUSE(0)
140 IF A$=CHR$(13)  THEN 40
150 PRINT A$;
160 GOTO 110
```

MOUSE(1), but if the button was up, MOUSE(5) returns the horizontal coordinates of the mouse when the button was released. MOUSE(6) does the same thing, for the vertical coordinate. These functions are often used to determine the ending point when the cursor has been moved.

SUMMARY

The mouse and the Macintosh go hand in hand. So do the mouse and the Microsoft BASIC. Programming the mouse makes it possible to set up programs that use the mouse rather than the keyboard. With the mouse, you can set up some drawing functions that are not currently available in *MacPaint.* Fortunately, the lowered speed of an interpreter as compared to machine language subroutines is not usually a major problem when the computer is used as a drawing board. Using the MOUSE function, you can emulate much of what is already available in other programs provided for the Macintosh, but you can also set up your own functions to better suit your particular needs.

Chapter 10



Ready-to-Run Programs

No book about a computer that describes its general operation and attributes is complete without some ready-to-run programs that the reader may input directly to his/her computer. Therefore, this chapter is devoted completely to such programs. These programs have been written specifically for the Macintosh.

It's also a fairly simple matter to convert programs written in other Microsoft BASICs to conform to the Macintosh's Microsoft BASIC. Therefore, any programs for other systems that you have access to can probably be run on your Macintosh, provided you adhere to the language the Macintosh understands. While it is sometimes difficult to convert from one dialect of BASIC to another, it is not so difficult to convert from one Microsoft BASIC to another. Still other programs have been written for the first time on the Macintosh. This allows us to take full advantage of the excellent operating characteristics of this computer when operated under Microsoft BASIC. All programs have been fully tested on the Macintosh, so if you input the programs in this chapter exactly as they appear, they should run successful every time.

Some of the programs included are for instructional purposes only, but most are designed to provide the user with a practical program that will perform a service. In the following pages, you will find programs that allow you to play games, perform mathematical operations, write unusual displays to the screen, and even one that serves as a practical alarm clock. If you study each program carefully, you will undoubtedly find ways to modify them. You may even want to use these programs as subroutines to perform a specific function in another program whose end purpose is completely different from the program here.

Before modifying any program, input it exactly as it appears in these pages. When the program is running properly, then go through each line and perform the desired modifications. This is a good practice, because it's very easy to make a typing

error while inputting any program listing. If you make modifications during the input and the program does not run, it's sometimes difficult to determine whether your modifications created the problem or a typing error did. To assure accuracy, the program listings in this book were output to the Macintosh ImageWriter printer after the programs ran successfully.

Along with each program listing is a brief explanation of how the program works. Where applicable, a picture of the on-screen display is also provided. This way, you can compare your on-screen display to the one pictured in this book to be certain the program is running successfully.

OHM'S LAW

This is a simple computational program (Listing 10-1) that allows you to input variables into the Ohm's Law formula of E = IR, where E represents voltage, I current in amperes, and R resistance in ohms. This formula states that circuit voltage (in volts) is equal to current times resistance. This is probably the easiest type of program to write.

Line 50 clears the screen, and line 60 prompts the user to input the value of current in amperes which is assigned to I. Line 70 asks for the resistance in ohms, which is assigned to R. Line 80 clears the screen again, and line 90 uses the variable values in the formula. It is necessary for us to insert an asterisk between the I and R to indicate that they are to be multiplied. The product of I times R is assigned to E. Line 100 then displays the value of E, followed by the units, volts.

The Ohm's Law formula has many variations, but each can be worked with this program by changing the formula in line 90 and the input prompts. Any formula, no matter how complex, can easily be worked through a program such as this. If you can find the formula, then you can always put it in the program.

MORTGAGE PAYMENTS

Here is another program (Listing 10-2) that depends on mathematical formulas to perform calculations on the user's input. This program will allow you to input a monetary figure that represents a loan. It will then output the monthly payment to satisfy this figure based upon the number of years and interest rate. Several formulas are used in this program, but the program is almost identical to the Ohm's Law one in most other ways.

Line 50 instructs you to enter the amount to be

Listing 10-1. Ohm's Law.

```
10 REM OHMS LAW BASIC FORMULA
20 REM E=I*R WHERE I IS CURRENT IN AMPERES
30 REM AND WHERE R IS RESISTANCE IN OHMS. E IS GIVEN IN VOLTS
40 REM COPYRIGHT FREDERICK HOLTZ 2/84
50 CLS
60 INPUT"INPUT CURRENT IN AMPERES";I
70 INPUT"RESISTANCE IN OHMS";R
80 CLS
90 E=I*R
100 PRINT E;"VOLTS"
```

Listing 10-2. Mortgage Payments.

```
10 REM MORTGAGE PAYMENTS
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 WIDTH  40
50 INPUT"ENTER THE AMOUNT TO BE FINANCED";A
60 CLS
70 INPUT"HOW MANY YEARS";Y
80 CLS
90 INPUT"ENTER THE INTEREST RATE";I
100 Y=Y*12
110 I=(I/100)/12
120 MORT=(I/((I+1)^(Y)-1)+I)*A
130 CLS
140 PRINT"MONTHLY PAYMENT =";CSNG(MORT)
150 CALL MOVETO(0,200)
160 BEEP
170 INPUT"DO YOU WISH TO ENTER ANOTHER VALUE(Y/N)";A$
180 IF A$="Y" OR A$="y" THEN 30
190 CLS
200 PRINT"PROGRAM TERMINATED"
```

financed. The input is stored to variable A. Line 70 asks for the number of years over which the loan is to be amortized. This is assigned to Y. The interest rate is input in line 90 and assigned to I. The interest rate must be input as a percent and not as a decimal. If the loan is to be calculated at 13 percent, you should input 13 in response to this prompt. If it's 13½ percent, you should input 13.5.

Our formulas begin in line 100. Here, the number of years input in line 70 is converted to months by multiplying Y times 12. The interest rate is converted to interest for each month in line 110. These values are worked through another formula in line 120, which calculates the monthly payment.

In Microsoft BASIC for the Macintosh, all numeric variables are assumed to be double precision numbers. This simply means that all mathematical operations are carried out to many decimal places. While double precision numbers are great for complex mathematical operations that require a high degree of accuracy, this program does not require that the last fraction of a penny be calculated to the *n*th decimal place. Therefore, line 140 uses the CSNG function to convert numeric variable MORT to a single precision number. This simply cuts down on the number of decimal places that are carried out, giving you a more practical value to work with. For example, if you input a value of $55,000 for the loan and amortize it at 13 percent over 30 years, you will see that the mortgage payment would be $608.41 per month. However, if MORT was not converted to a single precision number, the double precision equivalent would be $608.4097355968. The computer, when

Listing 10-3. Leap Year Calculator.

```
10 REM LEAP YEAR CALCULATOR
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 WIDTH 40
50 INPUT"YEAR";Y
60 CLS
70 IF Y/4=INT(Y/4) AND Y/100<>INT(Y/100) OR Y/400=INT(Y/400)
   THEN 80 ELSE 110
80 CLS
90 PRINT Y;"IS A LEAP YEAR
100 END
110 CLS
120 PRINT Y;"IS NOT A LEAP YEAR"
130 END
```

operating under single precision, simply rounds this long string of numbers to the more practical answer of 608.41.

This one is an eminently practical program, as it will allow you to calculate amortization for any value, any number of years, and any interest rate. Very few amortization charts or booklets will give you this amount of flexibility.

LEAP YEAR CALCULATOR

This is one of my favorite programs (Listing 10-3), not because it's complex, educational, or even necessary. The main reason I like to describe such a program in my books is because it lets me expound upon just exactly what Leap Year is and more importantly, when it occurs. Leap Years are necessary in order to adjust our calendars with the rotation of the earth around the sun. Most of us consider the period of time for a full solar orbit of the earth to be 365 days. Actually, it's not exactly 365 days, so after a set number of years, it's necessary to add one day on the calendar to make all the other dates balance out. If we didn't have Leap Year, after a thousand years or so, the calendar would be so out of whack that we would have temperatures in the nineties when the calendar indicated it was December and we would be ice skating in the middle of August.

Now, most people believe that Leap Year occurs every four years. This is not correct. A mathematical formula is used to calculate what years are deemed Leap Years. Indeed, Leap Year usually occurs every four years, but every hundred years or so, eight years may pass between Leap Years. A Leap Year is any calendar year that is evenly divisible by 4 and *not* evenly divisible by 100 *or* any calandar year that is evenly divisible by 400. Therefore, the year 2000 is a Leap Year. This is because it is evenly divisible by 400. It is also divisible by 4, but this is not enough. Any year that is evenly divisible by 4 is not necessarily a Leap Year unless it is not evenly divisible by 100. However, if it is evenly divisible by 100, it may still be a Leap Year if it's evenly divisible by 400. Here's an example. The year 1896 was a Leap Year because it is evenly divisible by 4 and not evenly divisible by 100. However, the year 1900 (four years later) was not a Leap Year. 1900 is evenly divisible by 4, but it

is also evenly divisible by 100. This means that 1900 could be a Leap Year unless it is evenly divisible by 400, which it is not. Therefore, 1896 was a Leap Year,but the next Leap Year did not occur until 1904. An eight-year period spanned the time between these two Leap Years.

I've gone through this long, boring explanation for two reasons. First, it demonstrates that things are not always as simple as they seem. Second, it indicates that by inserting the proper mathematical formula into a computer program, we can easily determine which years fall under the Leap Year classification and which years don't. The Leap Year program uses all of this information in line 70 to determine whether the year input is a Leap Year. In line 50, you are asked to input the year. Line 70 uses the IF-THEN-ELSE sequence with the logical operators AND and OR. Line 70 says if the year (Y) divided by 4 is equal to the integer of the year divided by 4 (has no fraction), AND if the year (Y) divided by 400 is an even number, then go to line 80 and print LEAP YEAR. The ELSE portion of the line is activated if none of the other conditions prove true. The branch here is to line 110, where the screen displays the year you have input followed by IS NOT A LEAP YEAR. Again, the formula is the key here. Without the logical operator capability of Microsoft BASIC, the single-line formula in line 70 might require three separate formulas and many IF-THEN branches. So one program line can do the job on the Macintosh.

BINARY TO DECIMAL CONVERSION

When dealing with computers, we also deal with three types of numbers in the main. These are decimal, hexadecimal, and binary. We can convert a decimal number to a hexadecimal base by using the HEX$ function. However, the conversion of binary numbers to decimal numbers is a little more complex and no single-word function is usually found in BASIC to allow such a conversion. This program (Listing 10-4) will allow you to input the eight place binary number and will then convert it to its decimal number equivalent. Remember, binary numbers are comprised of ones and zeros only and there must be eight of them in any combination to fit the

Listing 10-4. Binary to Decimal Conversion.

```
10 REM BINARY TO DECIMAL CONVERSION
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 INPUT"TYPE THE BINARY NUMBER";B$
50 IF LEN(B$)<>8 THEN 40
60 FOR X=1 TO 8
70 X$=MID$(B$,X,1)
80 IF X$="0" THEN COUNT=COUNT +1:GOTO 100
90 DEC=2^(8-X)+DEC
100 NEXT X
110 CLS
120 PRINT"THEN DECIMAL EQUIVALENT OF";B$;" IS";DEC
130 CLEAR
140 PRINT:PRINT:PRINT:PRINT
150 INPUT"DO YOU WISH TO CONVERT ANOTHER NUMBER(Y/N)";A$
160 IF A$="Y" OR A$="y" THEN 30 ELSE CLS:END
```

number in the computer equivalent of binary 00000000 is 0, while the decimal equivalent of binary 11111111 is 255. Therefore, any binary number in the computer will always yield a decimal equivalent of from 0 to 255.

Line 40 asks that you input the binary number. Line 50 uses the LEN function to make certain the number you input (represented by B$) contains eight characters. If not, there is a branch back to line 40, where you are prompted to input the binary number again. To convert a binary number to a decimal, it is necessary to read each character position of your binary number, it is said to be in the seventh binary column. In this case, the number 2 is raised to the seventh power. If a 0 is found in this position, the value is still 0. Now, 2 raised to the seventh power is equal to 128. When performing binary to decimal conversions on paper, the number 128 would be jotted down. Then you go to the second character position in the binary number, which is in the sixth binary column. If a 1 appears here, you raise 2 to the sixth power, which is a value of 64. If a 0 is found here, the value is 0. When you've gone all the way through all right binary characters, you add the column equivalent and arrive at the decimal equivalent for the entire binary string.

Admittedly, this is a complex operation, but this program does it for you almost instantly. Line 70 uses the MID$ function in a FOR-NEXT loop to read the value of the binary character at each of the eight character positions. Line 80 branches to line 100 if the character found at any position is 0. If not, line 90 raises 2 to the power of 8 minus X. This value is added to the previous value of DEC. When the loop is exited, all binary character positions containing 1 have been used to raise 2 to the power of that column. The numeric variable DEC now contains the decimal equivalent of the binary number.

CHECKBOOK BALANCER

This is a practical program (Listing 10-5) that can be used by anyone to avoid all the longhand

Listing 10-5. Checkbook Balancer.

```
10 REM CHECKBOOK BALANCE PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 INPUT"CURRENT BALANCE";B
50 Y=B
60 CLS
70 CALL MOVETO(10,10)
80 PRINT "AMOUNT ";
90 INPUT X
100 CLS
110 Y=Y-X
120 CALL MOVETO(200,100)
130 IF Y<=0 THEN 150
140 PRINT "$";Y;"BALANCE":GOTO 70
150 PRINT"$";ABS(Y);"DEFICIT":BEEP
160 GOTO 70
```

math involved when writing checks and trying to balance your checkbook. This program allows you to input your starting checkbook balance and then input each check value. Each time you type in the amount of a check and press RETURN, the amount is deducted from your balance and displayed on the screen. You can also input deposits to your account via this program and have them reflected in the balance as well.

Line 40 prompts you to input the starting balance, which is committed to B. In line 50, Y is assigned to the same value as B. The screen is cleared in line 60, and line 70 uses the MOVETO subroutine called by Microsoft BASIC from Macintosh ROM. Line 80 prints the word AMOUNT on the screen at location 10,10 (specified in line 70). Line 90 uses the INPUT statement to accept your check amount. Line 100 clears the screen again and numeric variable Y, which is originally equal to the starting balance, is reassigned the value of Y minus X (balance minus check amount). Line 130 checks to see if your balance has reached 0 or gone below 0. If so, there is a branch to line 150. If not, line 140 is executed, which prints the dollar sign ($), the value of Y, and the word BALANCE. There is then a branch to line 70, where you are again prompted to input another check amount. If Y is indeed less than or equal to 0, line 130 branches to line 150. Here, the ABS function is used to print the absolute value of Y. If Y is equal to −1000, the ABS value is 1000. The ABS function simply removes the minus sign. However, line 150 prints a different prompt. The word "BALANCE" no longer appears next to the figure. It is replaced with the word "DEFICIT". This indicates that your account is overdrawn. To alert you to this fact, the BEEP statement is also included in line 150. Anytime your balance goes to 0 or below, the Macintosh speaker will emit a beep.

To add to your balance, you must enter deposits as a negative number. If you wish to input a deposit of 2000, enter it as -2000 in response to the AMOUNT prompt. With this simple program, you can easily keep track of the amount of money you currently have in your account. It is most useful to persons who must write a large number of checks and who, like me, do not figure the balance after each check, but at the end of the check writing sequence. It certainly rules out a great deal of the possible errors.

MACINTOSH TYPEWRITER

This program (Listings 10-6 and 10-7) is presented as two programs, both of which accomplish basically the same thing. Each allows you to use the Macintosh and ImageWriter printer just like a typewriter. When you run this program, the screen is cleared and you begin typing as you would on a typewriter. Each time you press RETURN, the line on the screen is then sent to the ImageWriter. Listing 10-6 uses the INKEY$ function in line 50, whereas Listing 10-7 uses INPUT A$.

In Listing 10-6, the screen is cleared in line 40 and line 50 assigns to A$ the value of the keyboard input (INKEY$). Line 60 branches to line 50 as long as there is no keyboard input, but when you hit a key, lines 70 through 110 are executed. Line 70 uses the POS function to determine the cursor position on the screen. Line 90 displays the character you typed. Line 100 uses the LPRINT statement to send the character to the ImageWriter printer. The character is not immediately printed; it's simply held in a special place in memory called a buffer. However, when you press RETURN, the buffer is dumped to the printer and the entire line is printed out. Line 80 dumps the printer buffer whenever the text cursor reaches the far right of the screen. This occurs when CT is equal to 60 (returned by the POS function).

The second program is simpler, but not quite as efficient. Line 50 assigns your keyboard input to A$. When you press RETURN, line 60 is executed. Here, the LPRINT statement sends the entire contents of A$ to the printer.

Listing 10-6. Macintosh Typewriter Version 1.

```
10 REM MACINTOSH TYPEWRITER VERSION 1
20 REM USES THE IMAGEWRITER PRINT AND THE MACINTOSH
30 REM KEYBOARD TO PROVIDE TYPEWRITER-LIKE OPERATION
40 CLS
50 A$=INKEY$
60 IF A$="" THEN 50
70 CT=POS(0)
80 IF A$=CHR$(13) OR CT=60 THEN PRINT:LPRINT:GOTO 50
90 PRINT A$;
100 LPRINT A$;
110 GOTO 50
```

ALPHABETIZER

Here is a very practical program (Listing 10-8) that will allow you to input up to 100 words to be alphabetized. The computer will then sort these words in alphabetical order and display them on the screen. Line 30 sets up a string array (A$) to hold the words that will later be input. Line 50 sets up 1 as an initial value for CT. This will be the count routine that will determine where the input words fall in the array. The array can only hold 100 elements with the routine used. Although it was DIMmed to a value of 100, which technically means that there are 101 element positions available (0-100), our count routine starts at 1 rather than 0, so A$(0) is never used.

Line 60 prompts you to input an item. It is assigned to array position A$(CT). Line 70 is the exit routine. It tests to see if A$(CT) is equal to either "END" or "end". If so, there is a branch to line 80, which begins the sort routine. If not, there is a branch to line 40, which again clears the screen. CT is stepped by 1, and you are again asked to input an item.

When you have finished inputting all items, type END in response to the prompt. This brings about a branch to line 80, where the screen is cleared. Lines 90 and 100 print the word PROCESSING at the center of the screen. If you input a long list of words, it can take thirty seconds or so for the computer to properly sort them in alphabetical order. The word PROCESSING is displayed to let the user know that the computer hasn't locked up.

Listing 10-7. Macintosh Typewriter Version 2.

```
10 REM MACINTOSH TYPEWRITER VERSION 2
20 REM USES THE IMAGEWRITER PRINT AND THE MACINTOSH
30 REM KEYBOARD TO PROVIDE TYPEWRITER-LIKE OPERATION
40 CLS
50 INPUT A$
60 LPRINT A$
70 GOTO 50
```

Listing 10-8. Alphabetizer.

```
10 REM ALPHABETIZING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 DIM A$(100)
40 CLS
50 CT=CT+1
60 INPUT"ITEM";A$(CT)
70 IF A$(CT)="END" OR A$(CT)="end" THEN 80 ELSE 40
80 CLS
90 CALL MOVETO(220,140)
100 PRINT"PROCESSING"
110 FOR XX=1 TO CT-1
120 FOR YY=XX+1 TO CT-1
130 IF A$(XX) > A$(YY) THEN SWAP A$(XX),A$(YY)
140 NEXT YY
150 NEXT XX
160 CLS
170 FOR X=1 TO CT-1
180 PRINT A$(X);" ";
190 LPRINT A$(X)
200 NEXT
```

The sort routine begins in line 110. Here, a loop is formed followed by a *nested* loop (loop inside a loop) in line 120. The value of XX is used to point to the first word held in the array, while YY points to the word immediately following it. Line 130 tests to see if the value of the first word is larger than that of the second. If so, the SWAP statement is used to replace A$(XX) with A$(YY) and vice versa. In other words, the array is sorted two words at a time, putting the one with the "lower" letter in the alphabet ahead of the one with the "higher" letter in the alphabet. This continues for every element in the array until all words are sorted. Remember, the nested loop goes through all of the words in the array following the one indicated by the outer loop. Let's assume we have just entered the sort routine. Therefore, XX represents the first word in the A$ array. The nested loop in line 120 compares this first word with all other words. This means that the lowest word in the entire array is put into the number 1 position. On the next pass of the XX loop, the second word in the array is compared with all others. If you input 10 words to be alphabetized (not including the word "END"), the YY loop nested within the XX loop will cycle 10 times for each cycle of the XX loop.

When the sorting has been accomplished, the screen is cleared in line 160. The loop in lines 170 through 200 prints the words in the array (now in alphabetical order) to the screen and to the ImageWriter printer. If you enter a long list of words, processing time can be considerable. Therefore, this program does not prove practical for alphabetizing lists of thousands of words. However, if you would like to alphabetize a list of 100 or less words, I think you will find it very valuable.

ALARM CLOCK

Some readers might wonder why I have included an alarm clock program (Listing 10-9) in this book since the Macintosh already comes equipped with one that can be accessed via the Apple menu. There are several reasons. First, this program is a good exercise in programming, and, second, I don't think the alarm clock routine found in the Macintosh is adequate, since when the alarm time is reached, the speaker only beeps once and then goes about the process of quietly keeping time again. If you want to simulate a true alarm clock, the thing's got to keep beeping until you turn it off. That's what this program does.

When this program is run, you are asked to input an alarm time. The TIME$ function is then used to display the current time at the center of the screen. When the alarm time is reached, the speaker starts beeping and won't quit until you turn it off. To turn it off, simply press any character or number key and the computer reverts to quietly keeping time again. You can ignore the built-in ROM alarm clock, but you can't ignore this one—it will drive you crazy.

Line 40 asks you to input the alarm time. In the built-in version, you have to input this in hours, minutes, and seconds. But who really cares about seconds? In this version, all you input are the hours and minutes, such as 12:18. Remember, the internal clock keeps time by military standards, so 1:00 PM is expressed as 13:00, whereas 1:00 AM is 01:00. You must make your alarm time conform to this standard. When you press RETURN, the alarm time is displayed in the upper left corner of the screen, while the current time in hours, minutes, and seconds is displayed in the center. Line 60

Listing 10-9. Alarm Clock.

```
10 REM ALARM CLOCK
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 INPUT"ALARM TIME";AT$
50 CLS
60 PRINT"ALARM SET FOR";AT$
70 CALL MOVETO(220,145)
80 R$=LEFT$(TIME$,5)
90 PRINT TIME$
100 IF R$=AT$ THEN 130
110 FOR DLAY=1 TO 1000:NEXT DLAY
120 GOTO 70
130 BEEP
140 FOR DLAY=1 TO 100:NEXT DLAY
150 BEEP
160 FOR DLAY=1 TO 100:NEXT DLAY
170 A$=INKEY$
180 IF A$="" THEN 130
190 AT$=""
200 GOTO 70
```

prints the alarm time, which is represented in the program by AT$. Line 70 sets the graphics cursor near the center of the screen, and line 90 prints the TIME$ value. Line 110 is a delay loop to avoid screen flicker. The delay is just enough to refresh the TIME$ display every second.

Line 80 assigns to R$ the value of the first five characters in TIME$. This includes the hours and minutes portions and the colon that separates them. This effectively assigns R$ the value of TIME$ without the seconds portion. Line 100 compares R$ with AT$, the latter being the alarm time specified. When the two are equal, there is a branch to line 130, where two beeps are committed to a routine with delays between each. Line 170 uses the INKEY$ function to test the keyboard for a key strike. If there is none, line 180 branches back to line 130, so the beep sequence is repeated over and over again. When you hit a key to stop the alarm, AT$ is reassigned to a value of nothing and there is a branch to line 70, where the time display sequence is reentered. The computer now quietly displays time.

This program allows the computer to better simulate a practical alarm clock like the ones found in most homes. It can be set to go off at a certain time. When the alarm sounds, it must be turned off. The alarm can be reset by running the program again.

ASCII CHARACTER DISPLAY

Throughout this book, you've seen references to the ASCII character set contained in the Macintosh. An ASCII character code is a standard for microcomputers that assigns certain numbers to the characters that microcomputers can display. Usually, the first 128 characters (0-127) are standard for all machines. The remaining 128 (128-255) are special characters and may vary from machine to machine. Those first 128 characters are the ones that allow different types of computers to communicate with each other via a modem or some other serial device.

In the Macintosh character set, character 65 is the capital letter A. You will find that ASCII character 65 represents the capital letter A on all other types of modern microcomputers. Therefore, if we want to send the letter A from our Macintosh to someone with another type of computer in a faraway city, all we have to do is connect the two computers via phone line and modem and set our machine so that when an A is typed, its ASCII character code is sent to the other computer. The receiving computer will convert ASCII number 65 to its display equivalent, which is a capital A. This is about the only single link that is available to all microcomputers.

This program (Listing 10-10) is designed to show you the printable characters in the Macintosh ASCII character set (Fig. 10-1). Some of the characters are non-printable and are displayed on the screen in the form of a hollow square. Some of the characters are control characters. For instance, ASCII character 12 is a control character that clears the screen. It is not printable; it simply controls a machine function. The display is shown as it should appear on your screen.

Line 30 clears the screen, while line 40 sets the screen width to a maximum of 50 positions. Line 50 begins a FOR-NEXT loop that counts from 0 to 255. This spans the entire ASCII character range. Line 60 uses the CHR$ function coupled with the value of X. You will remember that CHR$ returns the character represented by the ASCII number in parentheses. The loop cycles until all 255 characters have been displayed on the screen.

When this program is first run, you will see a number of open boxes on the screen, and suddenly, the screen is completely cleared, because the loop value of X is equal to 12. The value of CHR$(12) is a control function that clears the screen. This will be followed by more boxes and the entire printable ASCII character set, which includes numbers,

Listing 10-10. ASCII Character Display.

```
10 REM PROGRAM TO PRINT THE ASCII CHARACTER SET
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 WIDTH 50
50 FOR X=0 TO 255
60 PRINT CHR$(X);
70 NEXT X
```

File Edit Control

ASCII SET

```
□□□□□□□□□□□□□□□□□□□□□ !"#$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
uvwxyz{|}~ÄÅÇÉÑÖÜáàâäãåçéèêëíìîïñóòôöõúùûü†°¢£§•¶ß®©™´¨□Æ
Ø□□□□¥□□□□□□□ªº□æø¿¡□□□□□□«»… ÀÃÕŒœ–—“”‘’□◊ÿ
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□
```

Fig. 10-1. The ASCII character set, printed on the Macintosh screen.

punctuation marks, upper- and lowercase letters, and special stylized letters and symbols. Anytime you see an open box on the screen, this means that the character is not printable.

MATHEMATICAL CIRCLE

As we enter the section of this chapter on graphics programs, I think this one will clearly demonstrate the power behind such graphics statements as CIRCLE, LINE, and PSET. As you can see from the picture on the screen, this program (Listing 10-11) displays a circle (Fig. 10-2). Admittedly, it's not a very good circle, but then again, it was generated by a mathematical formula rather than by using the CIRCLE statement. This image could have been drawn a lot better using the CIRCLE statement with only one program line instead of several. This circle is done on a point-by-point basis using PSET. Actually, PSET is a graphics function, but it can also be replaced with a mathematical formula that PEEKs and POKES the Macintosh screen memory, resetting the decimal byte numbers with those that will display dots. This, however, is not within the scope of this book.

A FOR-NEXT loop is entered in line 30 to set the vertical height of the circle. The circle spans a distance of 200 points, because the starting value is -100 and the top value is 100. Line 40 contains the circle formula. It subtracts the value of -X times X from 10,000 and raises it to a power of .575 times .495. The latter value establishes our ratio, which is slightly less than one-half. Lines 50 and 60 use PSET statements to set the points determined by the values of X and Y.

The mathematicians in the readership will understand the formulas, but for the rest of you, suffice it to say that the formula, when run through 200 times, sets 200 different dots on the screen in a circular pattern. Again, you can begin to comprehend the power of the CIRCLE statement in Microsoft BASIC, which works on the same principle.

RANDOM LINES

This is a fun program (Listing 10-12) that simply draws lines at random on the Macintosh screen (Fig. 10-3). Lines 40 and 50 are executed over and over again until you stop the program. The RND function is used in line 40 to draw a line from the current graphics cursor position (initially at coordinates 248,147) to a point on the screen determined by the value of the random numbers multiplied by the integer values. I did not elect to use the INT function here, and in every case, the coordinate values returned by the RND function are not integers. However, the computer will simply locate a starting and ending point based upon the nearest integer value of the non-integer numbers. The screen picture included here depicts the screen after about ten seconds of run time. If you let this program go long enough, the entire screen will be filled to a solid black.

GRAPHIC FUNNEL

This program (Listing 10-13) depicts a graphic funnel on the Macintosh screen (Fig. 10-4). It does this by aligning ever-expanding circles. The CIRCLE statement is found in line 50 within a FOR-NEXT loop. The loop counts from 51 to 140 in steps of 4. The first circle is displayed at coordinates 51,189 (240-X). The radius of the first circle will be 11 pixels, since 51 minus 40 is 11. The next circle will be displayed at coordinates 55,185, since the next value of X will be 55. The second circle will have a radius of 15 pixels. This process continues until the loop times out, each successive circle having a slightly larger radius and being located above and to the right of the previous one.

STEER HORN

The Steer Horn program (Listing 10-14) is a favorite one. It combines the CIRCLE statement and the COS function (cosine) to plot a cosine curve of ever-expanding circles on the screen (Fig. 10-5).

Listing 10-11. Mathematical Circle.

```
10 REM POINT BY POINT CIRCLE
20 CLS
30 FOR X=-100 TO 100
40 Y=(10000!-X*X)^.575*.495
50 PSET(X+248,Y+130)
60 PSET(X+248,130-Y)
70 NEXT
```

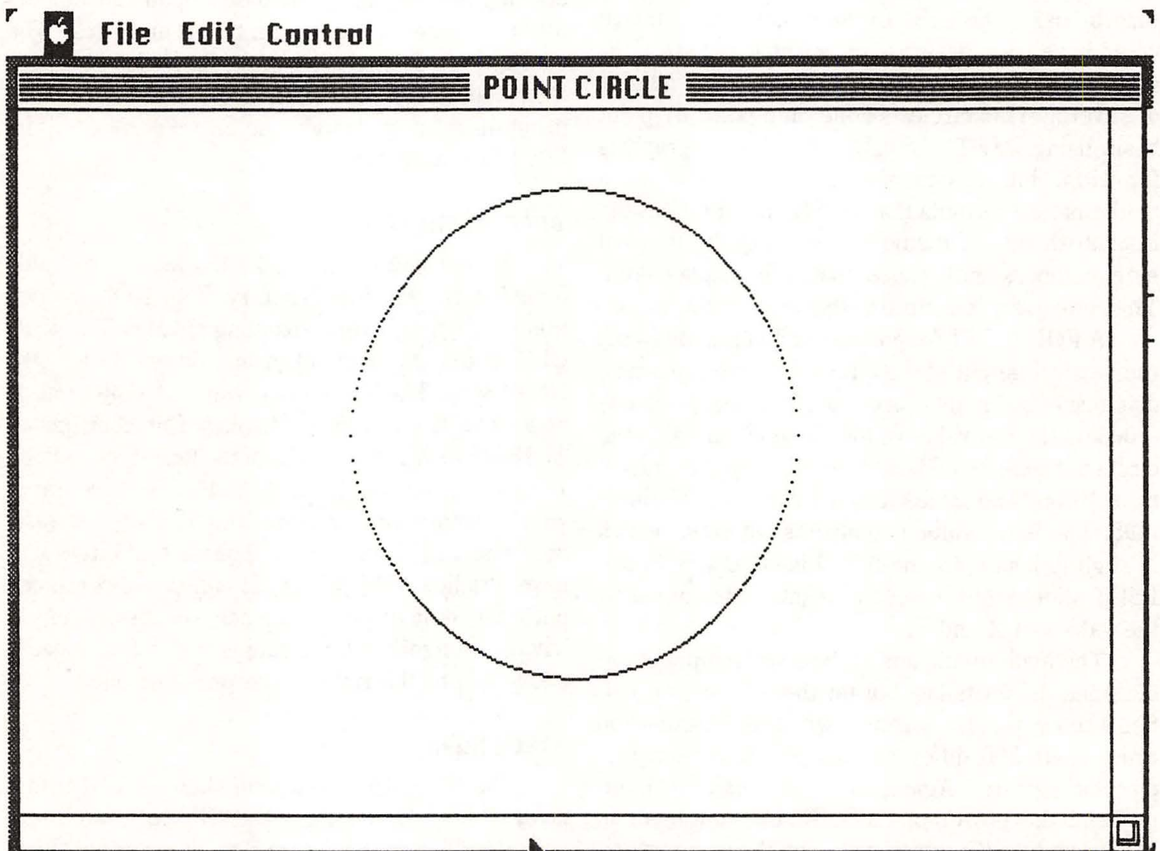


Fig. 10-2. A mathematically drawn circle.

Listing 10-12. Random Lines.

```
10 REM RANDOM LINES
20 REM COPYRIGHT FREDERICK HOLTZ 2/84 RUN
30 CLS
40  LINE -(RND*500,RND*300)
50 GOTO 40
```

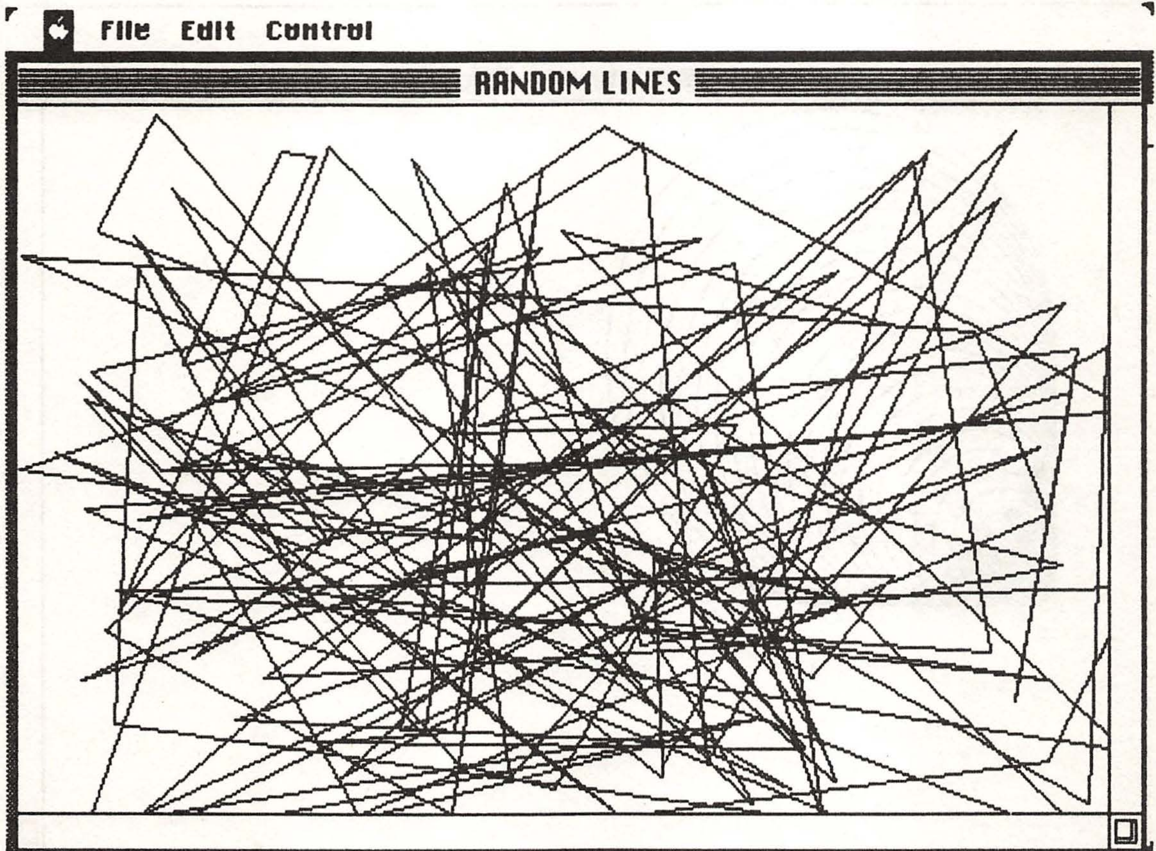


Fig. 10-3. Some random lines displayed by the random lines program.

Listing 10-13. Graphic Funnel.

```
10 REM FUNNEL DESIGN
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 FOR X=51 TO 140 STEP 4
50 CIRCLE(X,240-X),X-40
60 NEXT X
70 GOTO 70
```

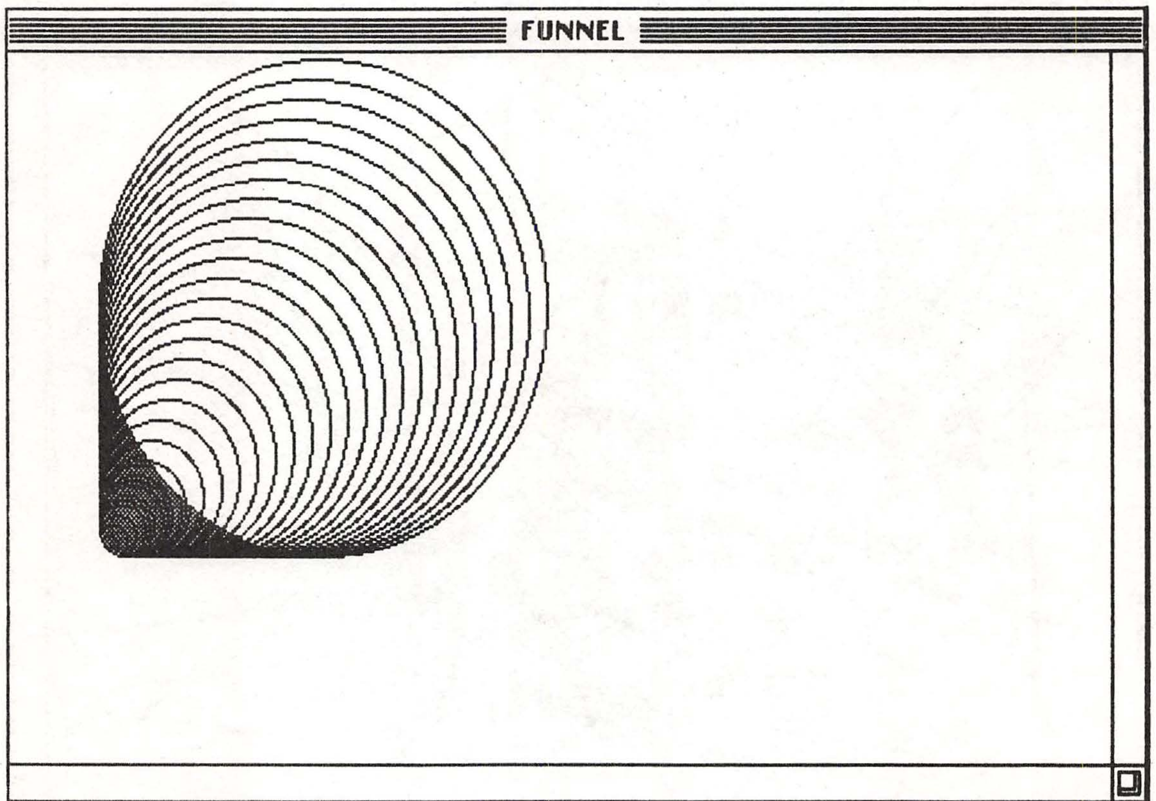


Fig. 10-4. The funnel, drawn using successively larger circles.

Listing 10-14. Steer Horn.

```
10 REM STEER HORN
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 FOR X=1 TO 340
50 IF A=30 THEN A=33 ELSE A=30
60 Y=50*COS(X/50)
70 CIRCLE(45+X,Y+140),1+X/5,A
80 NEXT X
90 GOTO 90
```

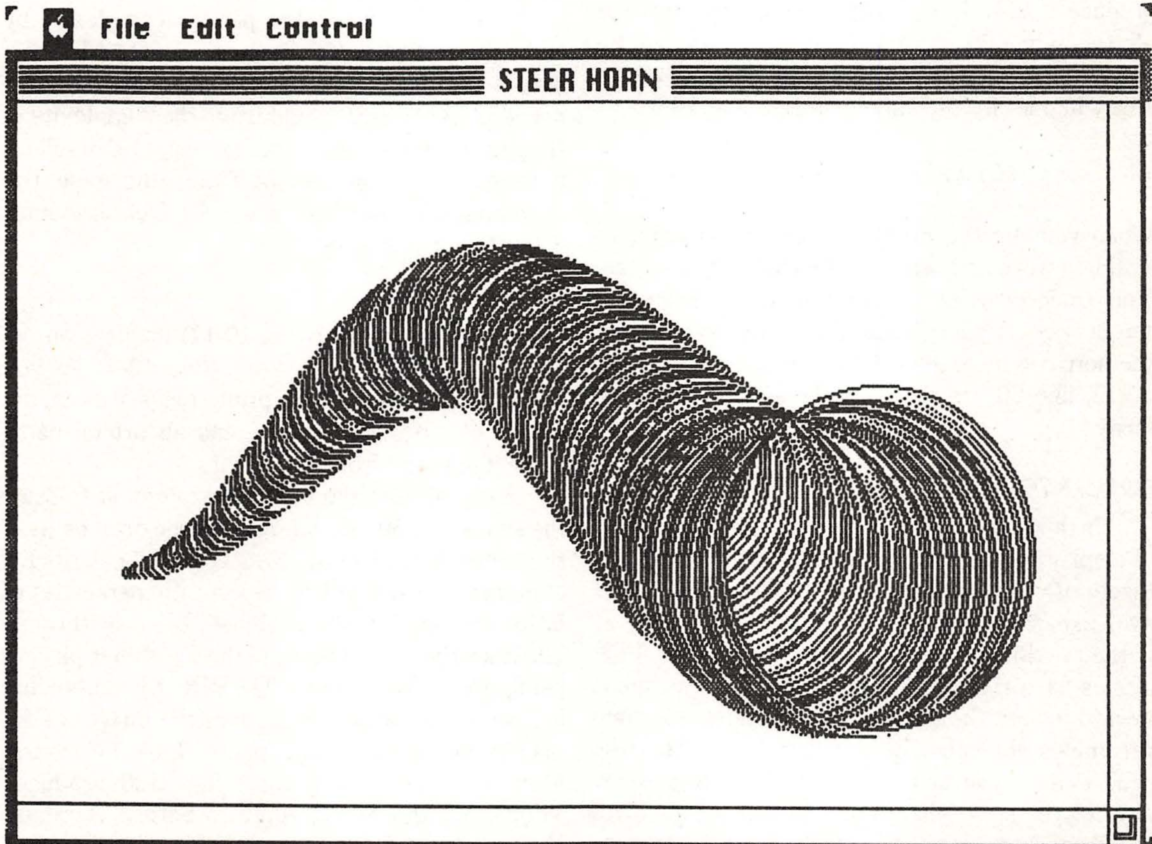


Fig. 10-5. The steer horn, drawn using a FOR-NEXT loop to change the value of the radius.

This works just the way the previous program did, except the location of each circle is plotted along a cosine wave that makes the image look like one-half of the chapeau sported by many Texas steers.

Line 50 assigns to A a value that is 30 on one pass of the loop, 33 on the next, and then goes back to 30 again. Each time the loop cycles, the number is first 30 and then 33. This rotation takes place for the entire cycling of the loop. You will remember that 30 and 33, when coupled with graphics statements, denote screen writes in white (30) and black (33), respectively. This means that every other circle is written in white and cannot be seen on the screen. This gives us the rough effect needed to produce a steer horn image. The cosine curve is plotted in line 60. Y is used as the Y coordinate for the CIRCLE statement. To change this program to plot without circles, simply change line 70 to:

```
70  PSET (45+X,Y+140)
```

When you run the program with this revision, a standard waveform will be displayed on the screen. This basic program can be modified in many different ways. By increasing the count of your loop, the horn can be extended. By fooling with the values in line 60, its shape can be altered tremendously.

CIRCLE PATTERN

In this program (Listing 10-15) circles are used to display a geometric pattern at the center of the Macintosh screen (Fig. 10-6). The principle of ever-expanding circles is incorporated here just as in the previous program, but in reverse. All of the circles have the same center. The circles expand inward toward the center, and the pattern formed resembles what might be seen on a sea shell like the sand dollar. You can play with this program by altering the step value in line 40. Larger numbers will leave more spacing between circle perimeters and produce a different pattern.

MULTICIRCLE PATTERN

This program (Listing 10-16) is almost a repeat of the circle pattern program, except two clusters of circles are used. Both overlap to produce an interesting pattern (Fig. 10-7). An additional CIRCLE statement is found in line 70, and note that in line 50, the step value is committed to S. This is assigned a random number from 1 to 10 by the RND function in line 40.

The program is set up on a continuous loop, so it will run over and over again, producing many different circle patterns. The delay loop in line 90 allows viewers time to see the display before it is cleared and another one begins.

You can increase the pattern complexity by adding yet another CIRCLE statement at line 75. The next circle might be located at screen coordinates 325,135. This would triple the complexity of the pattern. If the image does not establish itself for a long enough period of time, increase the maximum value of DLAY in line 90. Decrease it for a shorter display time.

SOLAR SYSTEM

This program (Listing 10-17) displays an orbital chart of our solar system (Fig. 10-8). As you can see from the screen print, the sun is at the center of the solar system, and all orbital paths expand outward from this point.

Line 40 uses the CIRCLE statement to draw the sun. Lines 50 through 70 draw the orbit paths of the outer five planets. Notice that the CIRCLE statement is used with an aspect ratio parameter of 5/18. This produces a fat ellipse. Lines 80 through 100 draw the orbital paths of the four inner planets using smaller step values. The CIRCLE statements in lines 110 through 190 produce the images of the planets along the orbital paths. Line 110 draws Mercury (nearest the sun); line 120 produces Venus. Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto are produced by the remaining CIRCLE statements.

Listing 10-15. Circle Pattern.

```
10 REM CIRCLE PATTERN
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 FOR X=1 TO 100 STEP 2
50 CIRCLE(248,135),100-X
60 NEXT X
```

Fig. 10-6. A circle pattern created using a FOR-NEXT loop and the CIRCLE command.

Listing 10-16. Multicircle Pattern.

```
10 REM MULTICIRCLE PATTERN
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 S=INT(RND*10)+1
50 FOR X=1 TO 100 STEP S
60 CIRCLE(175,135),100-X
70 CIRCLE(250,135),100-X
80 NEXT X
90 FOR DLAY=1 TO 2500:NEXT DLAY
100 CLS
110 GOTO 40
```

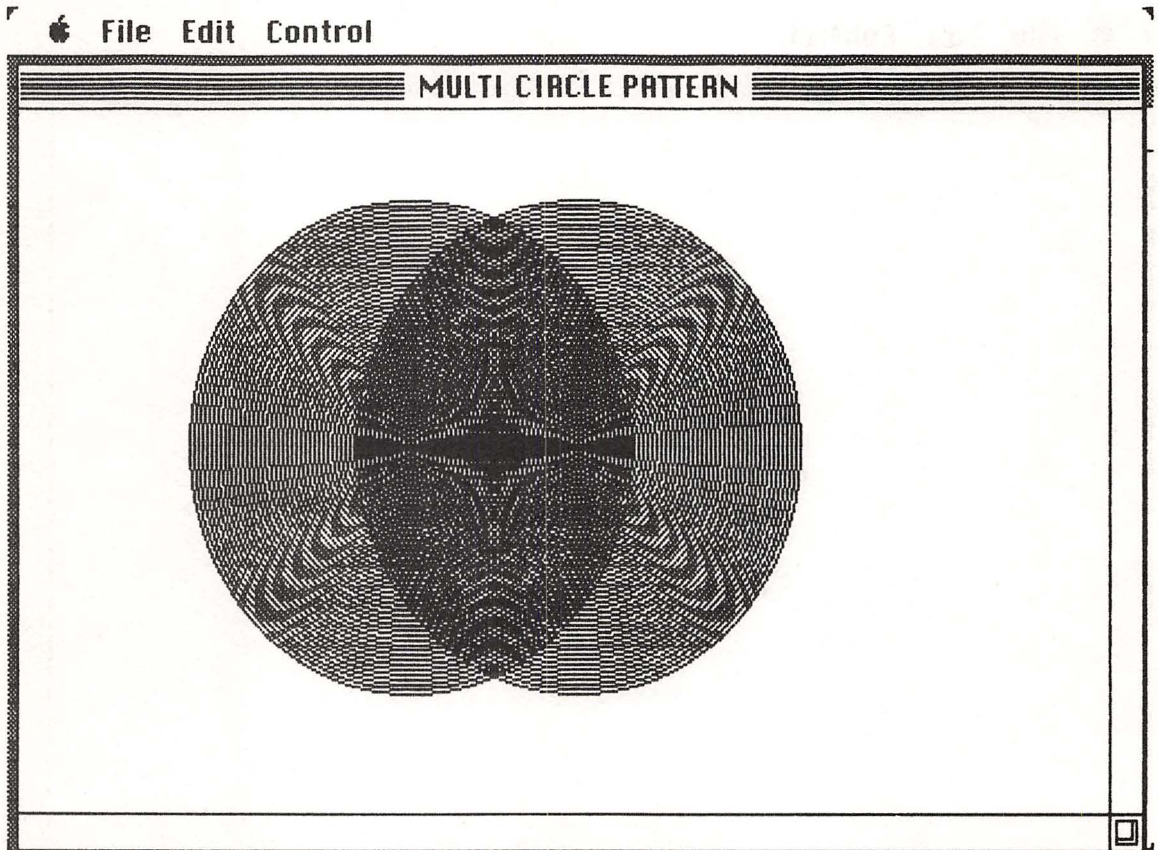


Fig. 10-7. A multicircle pattern.

Listing 10-17. Solar System.

```
10 REM SOLAR SYSTEM
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 CIRCLE(45,120),10
50 FOR X=150 TO 360 STEP 50
60 CIRCLE(45,120),X,,,,5/18
70 NEXT X
80 FOR X=30 TO 100 STEP 20
90 CIRCLE(45,120),X,,,,5/18
100 NEXT X
110 CIRCLE(15,120),3
120 CIRCLE(90,125),4
130 CIRCLE(100,107),5
140 CIRCLE(135,118),4
150 CIRCLE(150,150),7
160 CIRCLE(249,120),6
170 CIRCLE(245,78),5
180 CIRCLE(200,49),5
190 CIRCLE(160,26),4
200 GOTO 200
```

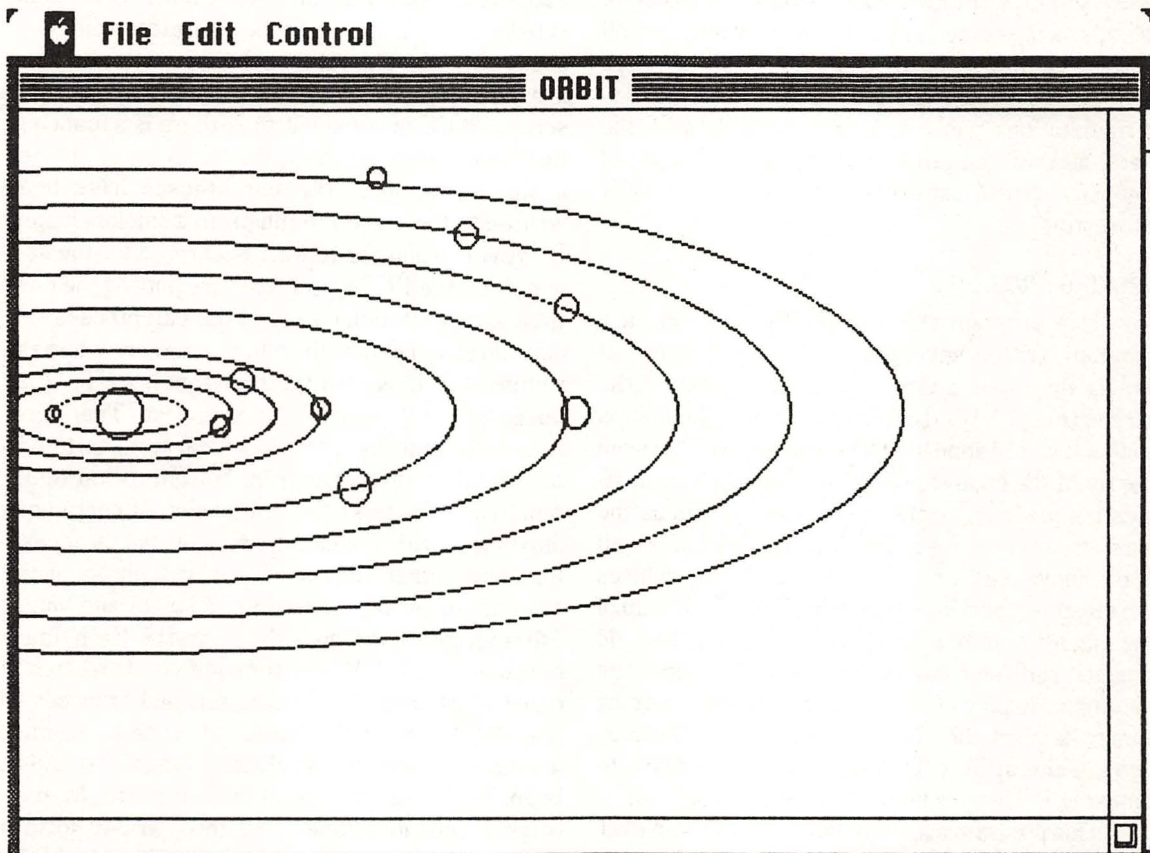


Fig. 10-8. A rendition of the solar system.

FOUR-POINTED STAR

Here (Listing 10-18) is another continuous loop graphics program that will display many different four-pointed star patterns on the screen (Figs. 10-9, 10-10, and 10-11). The patterns are determined at random. Line 120 uses the LINE statement to place a horizontal line at the center of the screen. The remaining three LINE statements expand outward from this point to a peak at the top and bottom of the screen. These lines intermesh vertically and horizontally to form the intricate patterns displayed in the three accompanying screen prints. This is a symmetrical image, so every screen write that appears on the left side of the star is duplicated at the top and bottom as well as the right side. Try rearranging the values here and see what you come up with. With a little experimentation, you can round the points of the stars, and you can produce many different types of non-symmetrical pattern weaves that are equally interesting.

SMOKING CIGARETTE

This program (Listing 10-19) is based on a program written several years ago. Originally, it simply displayed a cigarette at the center of the screen (Fig. 10-12). Later, I added some animation so that it would appear as if smoke were rising from the tip of the cigarette (Fig. 10-13). Another modification made cigarette ash appear to form as the cigarette was smoked. This last rendition does all of the above plus it has a new routine that produces the effect of a portion of the ash falling off the end of the cigarette after a short period of time. I would imagine the next modification would involve the automatic display of the Surgeon General's warning somewhere on the screen. WARNING: The Surgeon General Has Determined That Cigarette Smoking is Dangerous to Your Microprocessor.

This program uses a melange of GET and PUT statements for the animation routine. The cigarette itself is produced by the LINE statements in lines 120 and 130. The smoke image moves and is really a combination of parentheses that are printed to the screen in lines 50 and 90. Each of the parentheses is committed to an array using GET in lines 60 and 100. Lines 70 and 110 erase these initial images from the screen as soon as they are drawn.

At this point, the cigarette image is drawn and our animation loops now begin in line 140. Lines 140 through 190 put a stream of parentheses vertically on the screen from the tip of the cigarette toward the top of the screen. A count routine is then entered at line 200. Each time CT assumes a value of 2, it is reset to 0 and another count routine in line 220 is entered. CT determines when a vertical line is to be drawn at the tip of the cigarette to simulate ashes forming. The value of R determines when the ash cluster is to fall toward the bottom of the screen. If CT is not equal to 2, there is a branch to line 140, where the animation routine is entered again. Remember, the parentheses have been written to the screen to illustrate a smoke cluster. However, when the routine is entered for the second time, the PUT statements are putting the ones their arrays contain on top of the parentheses that have already been written to the screen (when the routine was executed the first time). PUTting an image to itself causes it to disappear. Therefore, the smoke pattern seems to appear from bottom to top and then disappear from bottom to top (quite rapidly). This gives the impression of continuous movement with smoke billowing up and disappearing and another cluster of smoke then doing the same thing. At the same time, a longer and longer ash is formed at the tip of the cigarette. Each time a new line is added, R is incremented by 4. When R is equal to 24, line 230 detects this and branches to line 260. Here, a GET statement is used to capture a large portion of the ash cluster. Line 270 sounds a beep, while line 280 erases the ash cluster from its original position. Lines 290 through 340 form a separate animation loop that causes the ash cluster to fall toward the bottom of the screen. The two

Listing 10-18. Four-pointed Star.

```
10 REM 4 POINTED STAR
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 RANDOMIZE VAL(MID$(TIME$,7,2))
50 CLS
60 C=INT(RND*10)+2
70 X=400
80 XX=96
90 Y=147
100 YY=147
110 T=33
120 LINE(X,147)-(247,Y),T
130 LINE -(XX,147),T
140 LINE -(247,YY),T
150 LINE -(X,147),T
160 X=X-C
170 XX=XX+C
180 Y=Y+C
190 YY=YY-C
200 IF X>=247 THEN 120 ELSE 210
210 FOR DLAY=1 TO 1500:NEXT DLAY
220 GOTO 50
```

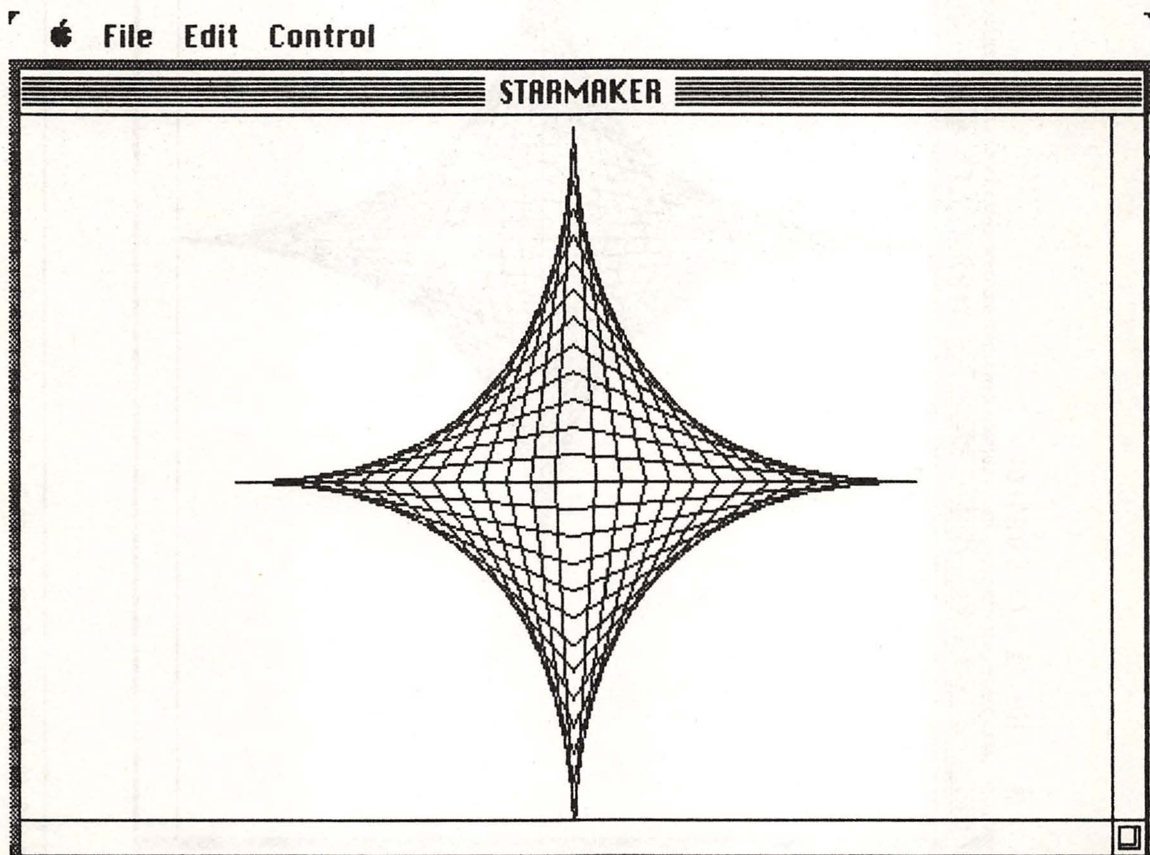


Fig. 10-9. The four-pointed star, version 1.

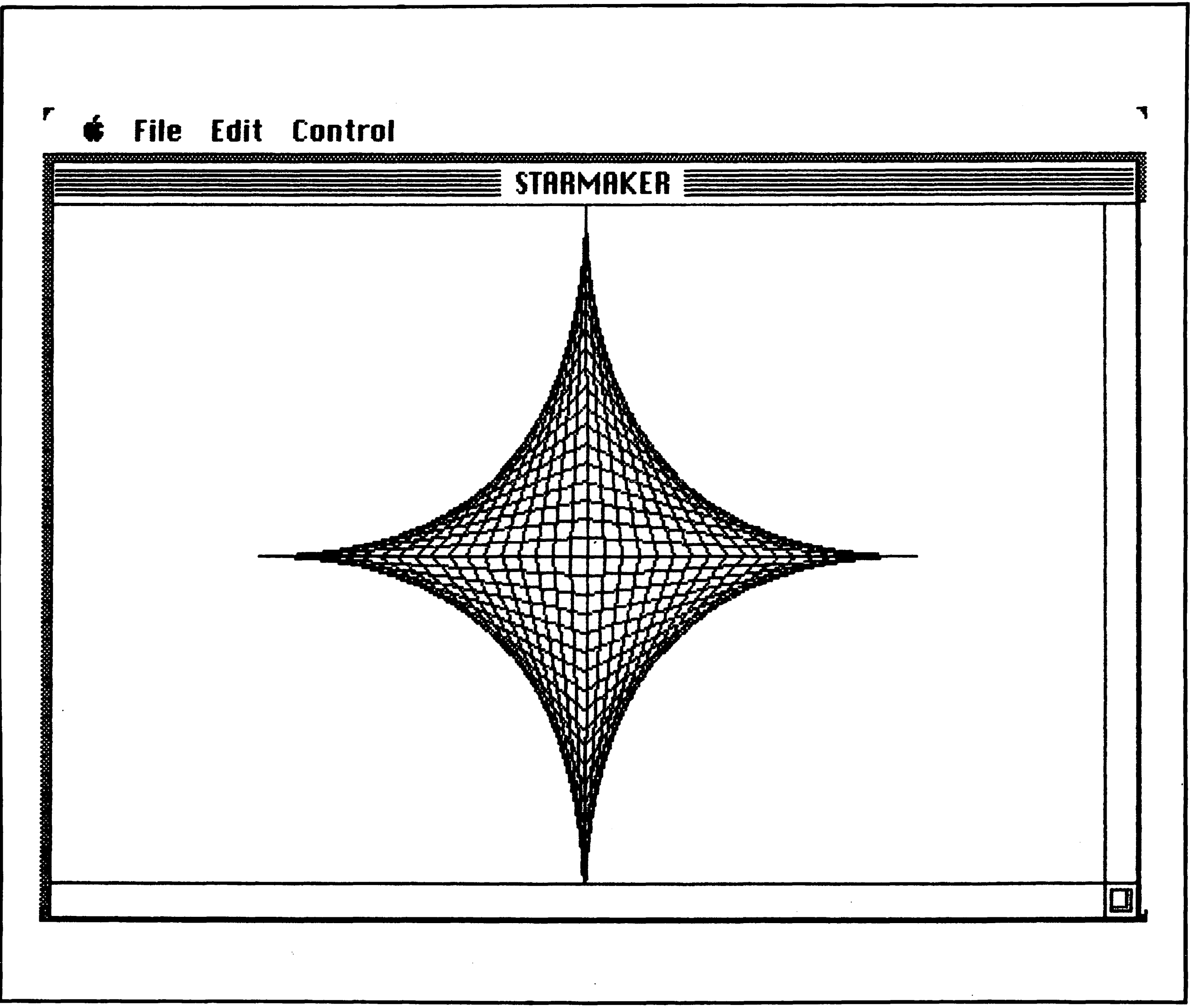


Fig. 10-10. The four-pointed star, version 2.

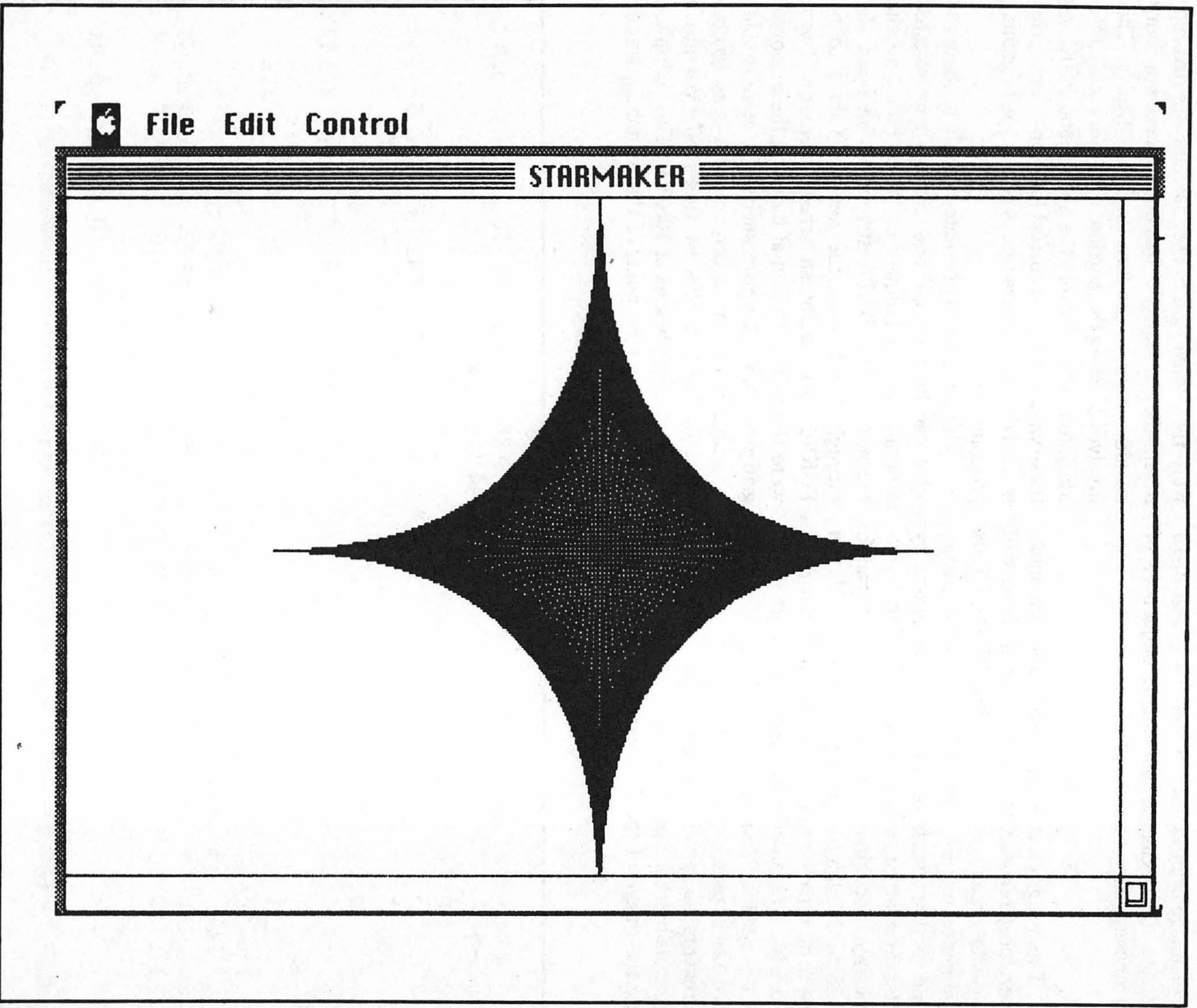


Fig. 10-11. The four-pointed star, version 3.

screen image prints shown in Figs. 10-12 and 10-13 illustrate the beginning and ending sequences of this amusing animation routine.

ORIENTAL PRINCE PICTURE

This program (Listing 10-20) is for the more experienced readers. It displays a detailed image of an oriental prince on the screen (Fig. 10-14). This is a computer picture, and it looks very realistic. While pictures such as these may be drawn by professional artists using a program like *MacPaint*, any computer hobbyist with no artistic ability can do the same through computer programming. All you have to do is have a high-resolution line drawing to start with. I obtained mine from a local art studio and enlarged it to about 10 times its normal size. Place a clear plastic grid over the picture. The grid represents the pixel positions on the Macintosh screen as blocks. The blocks were then filled in to copy the image of the drawing. The filled blocks were then converted to decimal numbers that represent the numbers the Macintosh screen memory should contain to produce the same display. This involves locating the portion of memory within the Macintosh that controls the screen and POKEing these values into their correct locations. Again, this is an elaborate process that is not for the beginning programmer.

This program is presented simply to show you what can be accomplished through sophisticated programming techniques on a machine like the Macintosh. The DATA statements in lines 150 through 880 contain the decimal values to be POKEd into Macintosh screen memory. These were the values arrived at by using the elaborate grid procedure. The program itself is quite simple.

Line 30 clears the screen and A is assigned a value of 108,300 plus 64 times 60. Macintosh screen memory begins at 108,300. The multiplied numbers place the point of the beginning screen

Listing 10-19. Smoking Cigarette.

```
10 REM SMOKING CIGARETTE
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 DIM A(100), B(100), C(400)
40 CLS
50 PRINT"("
60 GET(0,0)-(8,14),A
70 PUT(0,0),A
80 CLS
90 PRINT")"
100 GET(0,0)-(8,14),B
110 PUT(0,0),B
120 LINE(167,147)-(287,167),33,B
130 LINE (287,147)-(317,167),33,B
140 FOR X=127 TO 20 STEP -5
150 PUT(160,X),A
160 FOR DLAY=1 TO 10:NEXT DLAY
170 PUT(165,X),B
180 FOR DLAY=1 TO 10:NEXT DLAY
190 NEXT X
200 CT=CT+1
210 IF CT=2 THEN CT=0 ELSE 250
220 R=R+4
230 IF R=24 THEN 260
240 LINE(167+R,147)-(167+R,167)
250 GOTO 140
260 GET(167,147)-(180,167),C
270 BEEP
280 PUT(167,147),C
290 FOR DLAY=1 TO 10:NEXT DLAY
300 FOR X=148 TO 400 STEP 4
310 PUT(167,X),C
320 FOR DLAY=1 TO 10:NEXT DLAY
330 PUT(167,X),C
340 NEXT X
```

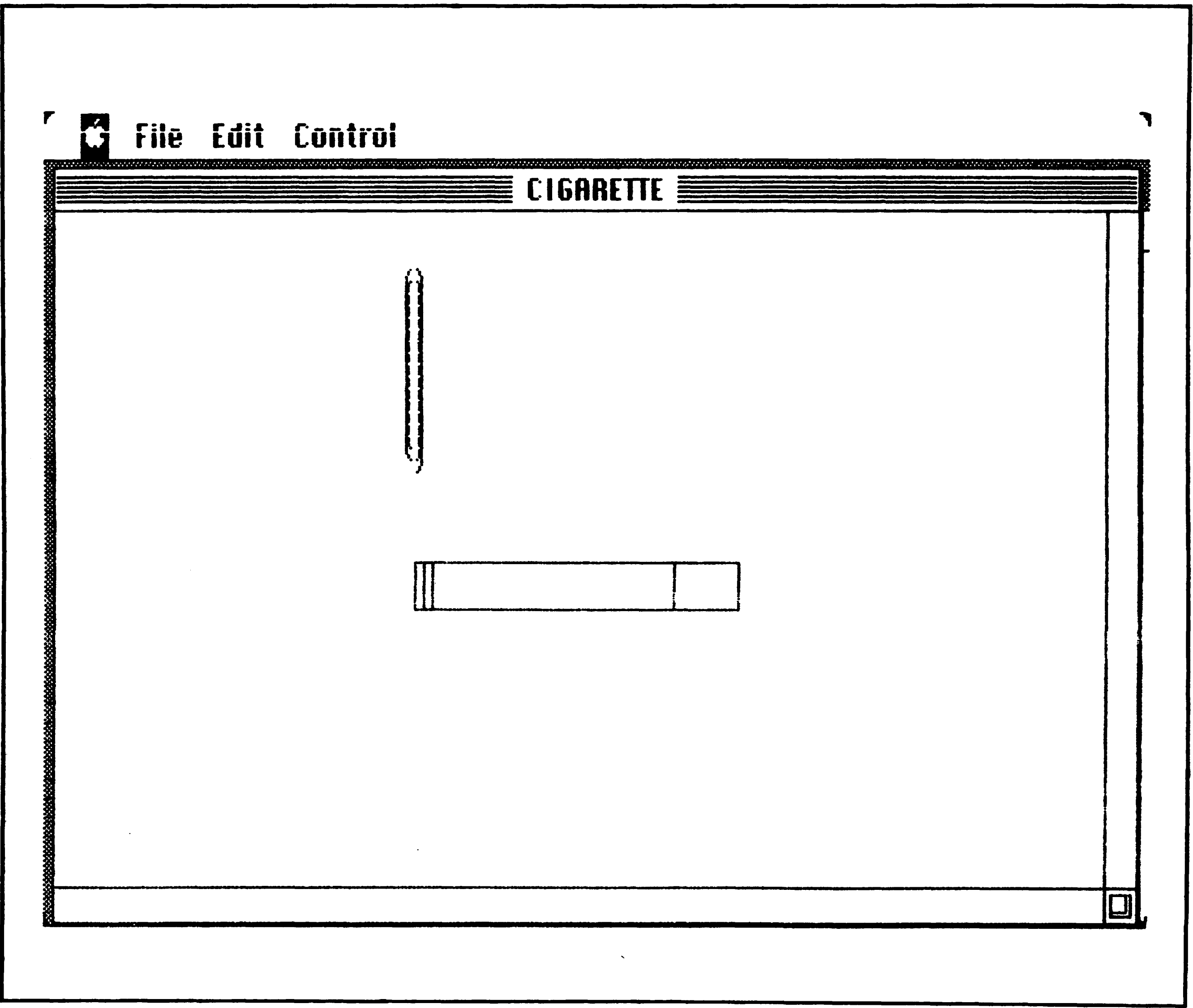


Fig. 10-12. The cigarette burning.

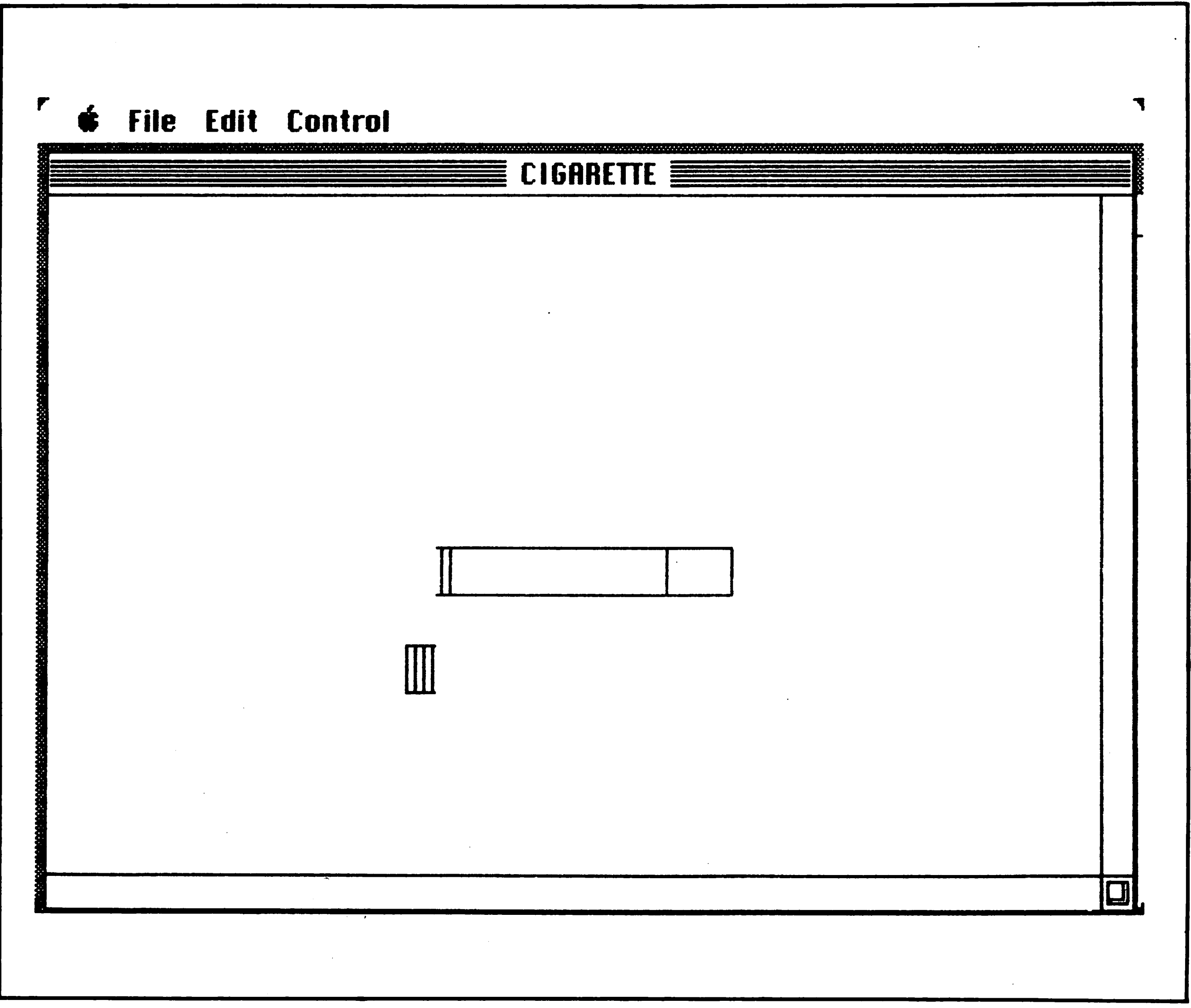


Fig. 10-13. The cigarette with falling ashes.

Listing 10-20. Oriental Prince Picture.

```
10 REM PICTURE OF AN ORIENTAL PRINCE
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 A=108300!+(64*60)
50 FOR X=A TO A+511 STEP 64
60 READ Q
70 IF Q=500 THEN END
80 IF Q=1000 THEN FOR X=A TO A+511 STEP 64:POKE X,0:NEXT X:GOTO 110
90 POKE X,Q
100 NEXT X
110 A=A+1
120 G=G+1
130 IF G=11 THEN G=0:A=A+501
140 GOTO 50
150 DATA 1000,1000,0,0,0,0,0,1,251,255
160 DATA 0,0,0,31,115,255,241,147,15,31,223,126,254,223,143,255
170 DATA 1000,1000,1000,1000,1000,1000
180 DATA 0,0,0,0,1,1,1,1,0,0,192,240,184,56,248,228
190 DATA 55,55,23,28,16,32,39,40,157,157,255,1,0,254,126,15
200 DATA 158,142,158,254,63,14,198,227
210 DATA 128,128,128,240,48,224,192,128
220 DATA 1000,1000,1000,1000,1000
230 DATA 1,1,0,0,0,0,0,0,164,180,252,126,126,56,24,12
240 DATA 56,28,30,31,30,26,63,59,3,121,113,113,1,1,129,129
250 DATA 251,255,255,255,255,255,255,255
260 DATA 128,128,192,128,128,0,0,0
270 DATA 1000,1000,1000,1000,1000,1000
280 DATA 12,12,14,6,6,7,7,31,61,31,1,0,31,251,118,228
290 DATA 128,194,141,255,255,255,127,127
300 DATA 127,254,255,252,252,246,226,195,0,0,252,231,119,55,227,227
310 DATA 0,0,0,0,192,240,254,63
320 DATA 1000,1000,1000,1000
330 DATA 0,0,1,2,4,15,15,15,7,199,95,127,63,193,193,128
340 DATA 232,252,30,31,159,174,238,238
350 DATA 15,108,109,249,253,255,121,121
360 DATA 17,177,207,63,7,255,255,223
370 DATA 239,123,255,215,223,229,227,211,255,239,254,127,253,239,159,
    255
380 DATA 192,176,216,216,240,240,248,255
390 DATA 1000,1000,1000
```

```
400 DATA 15,28,238,131,129,240,128,12
410 DATA 128,64,96,185,255,126,48,48
420 DATA 238,225,227,231,231,227,227,243
430 DATA 248,232,136,201,201,200,136,136
440 DATA 253,195,223,227,232,27,127,88
450 DATA 217,235,109,14,207,204,228,244
460 DATA 243,239,254,124,152,17,14,14
470 DATA 255,31,57,49,192,64,224,64
480 DATA 192,192,192,192,224,28,7,1
490 DATA 0,0,0,0,0,0,0,192,1000
500 DATA 2,7,15,15,15,152,241,7,224,192,192,224,96,192,224,240
510 DATA 243,227,253,255,252,191,191,56
520 DATA 137,137,8,254,13,136,232,56
530 DATA 252,248,32,255,0,64,64,115
540 DATA 228,30,230,2,3,243,255,252
550 DATA 57,48,12,14,15,127,129,0
560 DATA 192,103,39,31,128,192,249,127
570 DATA 1,15,252,224,224,224,192,0
580 DATA 224,128,0,0,0,0,0,0,1000
590 DATA 15,252,96,98,48,16,24,15
600 DATA 8,6,1,128,120,3,255,255
610 DATA 29,253,31,11,9,24,24,145
620 DATA 185,253,173,140,204,236,252,252
630 DATA 255,254,252,244,102,231,127,127
640 DATA 2,3,3,1,3,15,255,255
650 DATA 0,0,255,128,243,255,237,224
660 DATA 62,124,252,120,240,192,0,0
670 DATA 1000,1000,1000
680 DATA 15,0,0,0,0,0,0,0
690 DATA 199,1,1,1,1,1,1,1
700 DATA 255,126,14,254,255,255,255,249
710 DATA 252,36,4,196,230,230,230,230
720 DATA 253,124,0,0,3,6,7,0,
730 DATA 255,231,227,248,252,252,252,240
740 DATA 224,240,240,240,48,16,24,8
750 DATA 1000,1000,1000,1000,1000
760 DATA 1,1,1,1,1,1,0,0,
770 DATA 255,99,1,0,0,0,255,127
780 DATA 246,242,146,2,2,254,255,255
790 DATA 0,0,0,0,0,0,255,191,240,240,248,248,32,15,255,228
800 DATA 12,12,12,56,112,192,192,192
```

```
810 DATA 1000,1000,1000,1000,1000,1000
820 DATA 123,123,123,123,0,0,0,0
830 DATA 211,211,215,159,0,0,0,0,
840 DATA 132,196,196,198,0,0,0,0,
850 DATA 196,196,196,100,0,0,0,0,
860 DATA 192,128,128,128,0,0,0,0
870 DATA 1000,1000,1000,1000
880 DATA 500
```

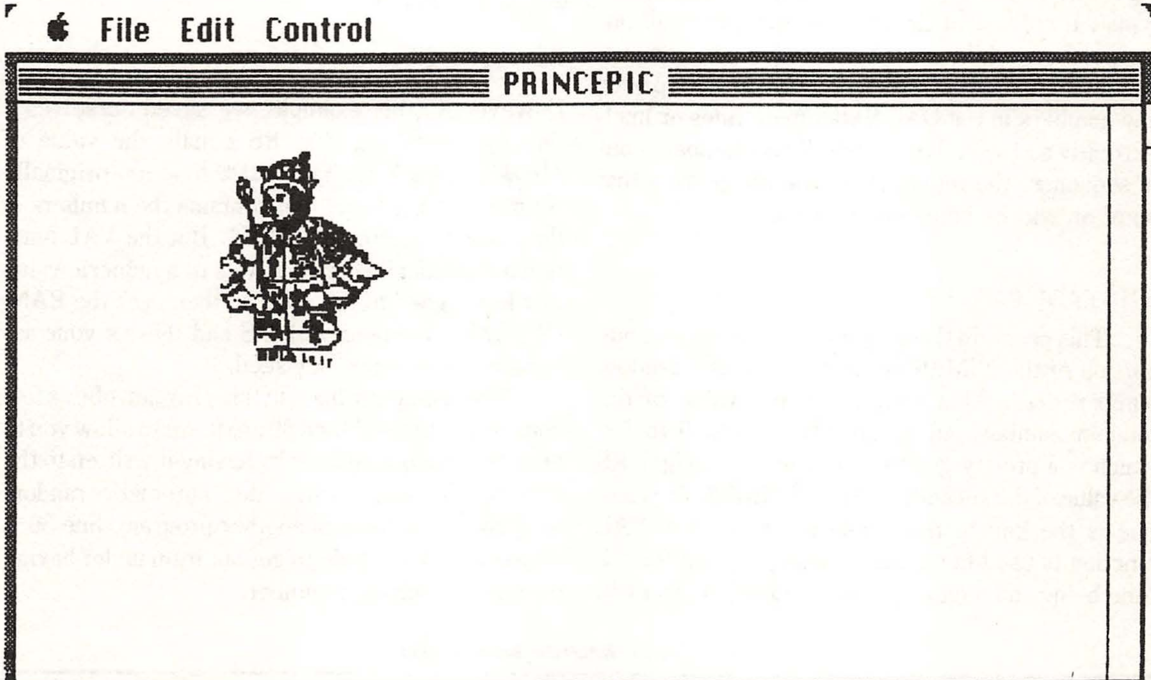


Fig. 10-14. A drawing of an oriental prince done using POKE and DATA statements.

write in the upper mid-left screen position. A loop is then entered in line 50 to step A from its initial value (assigned in line 40) to that value plus 511. This represents the width of the entire drawing. Line 60 uses the READ statement to access the first DATA element. Line 70 checks for the end of the DATA elements, which is represented by 500. Note that in line 880, the last DATA statement line, contains the number 500. Line 80 tests for a DATA element value of 1000. This means that a string of eight zeros are to be printed. This is easier from a programming standpoint than typing in 0,0,0,0, 0,0,0,0. If the value of Q is anything other than 1000 or 500, its value is POKEd into screen memory in

line 90. The loop then recycles.

When the first horizontal row is printed to the screen, the loop times out and A is incremented by 1. G is incremented by 1 as well. The value of G determines when the next row is to be started and its initial position on the screen. Line 140 then branches to line 150, which starts the POKEing process all over again for the next row.

This is a nice demonstration program, and it won't take you more than about 15 or 20 minutes to input it to your computer, providing you are a touch typist. If not, enlist the aid of a partner to call out the numbers while you type them into the computer. Make absolutely certain that you don't skip over any numbers in the DATA statement lines or inadvertently add a number or two. If one number is out of sequence, the remainder of the image from that point on will be completely wrong.

AUTOMATIC RANDOM SEED

This program (Listing 10-21) uses the seconds portion of the TIME$ function to act as a random number seed. This means that the value of the random number can by anywhere from 0 to 59, which is a prettty good range. Line 30 assigns RS the value of the seconds portion of TIME$. This one line is the key to the whole program. The VAL function is used here, but let's skip over it for the time being and look at the MID$ function. You will recall that MID$ was used in an earlier program in this chapter to set up an alarm clock. Here, MID$ is used to sample the last two characters in TIME$. This represents the seconds portion of TIME$. The normal format when using MID$ is something like:

RS$ = MID$(TIME$,7,2)

However, the RANDOMIZE statement can only work with numeric variables, so it is necessary to convert the string value (RS$) to a numeric value with a line like:

RS = VAL(RS$)

However, in this example, we saved ourselves a line by specifying that RS equals the value of MID$(TIME$,7,2). The MID$ function originally returns a string value that contains the numbers in the seconds portion of TIME$. But the VAL function automatically converts this to a numeric value that is assigned to RS. Line 40 then uses the RANDOMIZE statement with RS and there's your automatic random number seed.

The remaining lines in this program offer a few comments. Lines 70 and 80 are there to allow you to view the random number by having it written to the screen. However, when this automatic random seed routine is used in another program, line 30 is all you need to include to get out from under having to type in your own number.

Listing 10-21. Automatic Random Seed.

```
10 REM AUTOMATIC RANDOM SEED
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 RS=VAL(MID$(TIME$,7,2))
40 RANDOMIZE RS
50 REM RS IS NOW THE RANDOM SEED NUMBER
60 REM THE REMAINDER OF THE PROGRAM SIMPLY DISPLAYS THE VALUE OF RS
70 PRINT RS
80 GOTO 30
```

WORD MAZE

This program (Listing 10-22) randomly prints the 26 letters of the alphabet to the screen to form a word puzzle much like those seen in popular magazines (Fig. 10-15). The idea is to find the hidden words. Now, there are no words purposely programmed into this routine, but I have found that if you print a large block of letters at random, several small words will crop up. The idea of the game is to find them. The player who finds the most words (check a dictionary if necessary) is the winner. Note that the automatic random seed routine described previously is not included here. However, you can take line 30 from the previous program and substitute it for line 40 in this program and you're all set. As a matter of fact, anywhere you see a RANDOMIZE statement, you can always substitute this line.

The screen print shown here are generated at random. You can easily pick out several two-letter words, such as BY, IS, IT. If you look in the fourth row, tenth column, you will see the word OUT spelled from bottom to top. Traditionally, these puzzles display words from left to right, right to left, and diagonally. If you look closely, you will be able to pull many words out of such a puzzle. The chances of two puzzles being exactly alike are very slim. Due to the spacing of letters on the Macintosh, word puzzles such as these do not look as good as those generated by some other computers, but the Macintosh form of proportional spacing is highly advantageous for all other text processing functions.

NUMBERS GUESS GAME

Here is a program (Listing 10-23) that is certainly not unique among computer games, but is an interesting exercise and lots of fun to play. The computer randomly arrives at a number from 1 to 100. It is the player's job to guess that number. Each guess is typed in via the keyboard and if you guess wrong, the computer tells you whether the number is too high or too low. By following the computer's clues, you will eventually arrive at the correct number. When the correct number is guessed, the computer tells you how many tries it took to come up with the answer. The player with the lowest score is the winner.

The automatic random number seed routine is used with this program. Here, RN is assigned the value of the seconds portion of TIME$. RN is used with the RANDOMIZE statement in line 50. Lines 60 through 90 print the instructions on the screen. When you press RETURN, line 130 assigns a random number from 1 to 100 to numeric variable CN. You are then instructed to guess the number. Your guess is input at line 180 and assigned to X. Line 190 sets up a simple count routine that increments CT by 1 each time you input a guess. Lines 200, 210, and 230 compares your guess (X) with the actual number (CN). In line 200, if X is equal to CN, you've guessed the number correctly and a prompt is displayed telling you your guess is correct. There is then a branch to line 240, which arranges the display of the number of guesses it took you to arrive at the answer. In line 210, a test is made to see if your guess is less than the actual number. This sets up the printing of the TOO LOW prompt. Line 220 is then executed, which delays execution for a long period to allow the user to read the computer clue. Line 230 checks to see if X is larger than CN, which causes the TOO HIGH prompt to be displayed. Lines 270 through 300 allow you to play the game again as soon as one round has been completed. Type in Y to play again. If you type N or any other character, there is a branch to line 310, where the screen is cleared and PROGRAM TERMINATED is displayed.

Playing this game with the computer is just like playing it with a human being, except you can always be sure the computer will be truthful regarding clues. You can't always say this is true of a human player.

Listing 10-22. Word Maze.

```
10 REM WORD MAZE
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 RANDOMIZE
50 CLS
60 WIDTH 50
70 DIM A$(25)
80 FOR X=65 TO 90
90 A$(G)=CHR$(X)
100 G=G+1
110 NEXT X
120 FOR X=0 TO 400 STEP 14
130 FOR Y=0 TO 200 STEP 19
140 CALL MOVETO(X,Y)
150 PRINT A$(RND*25)
160 NEXT Y
170 NEXT X
```

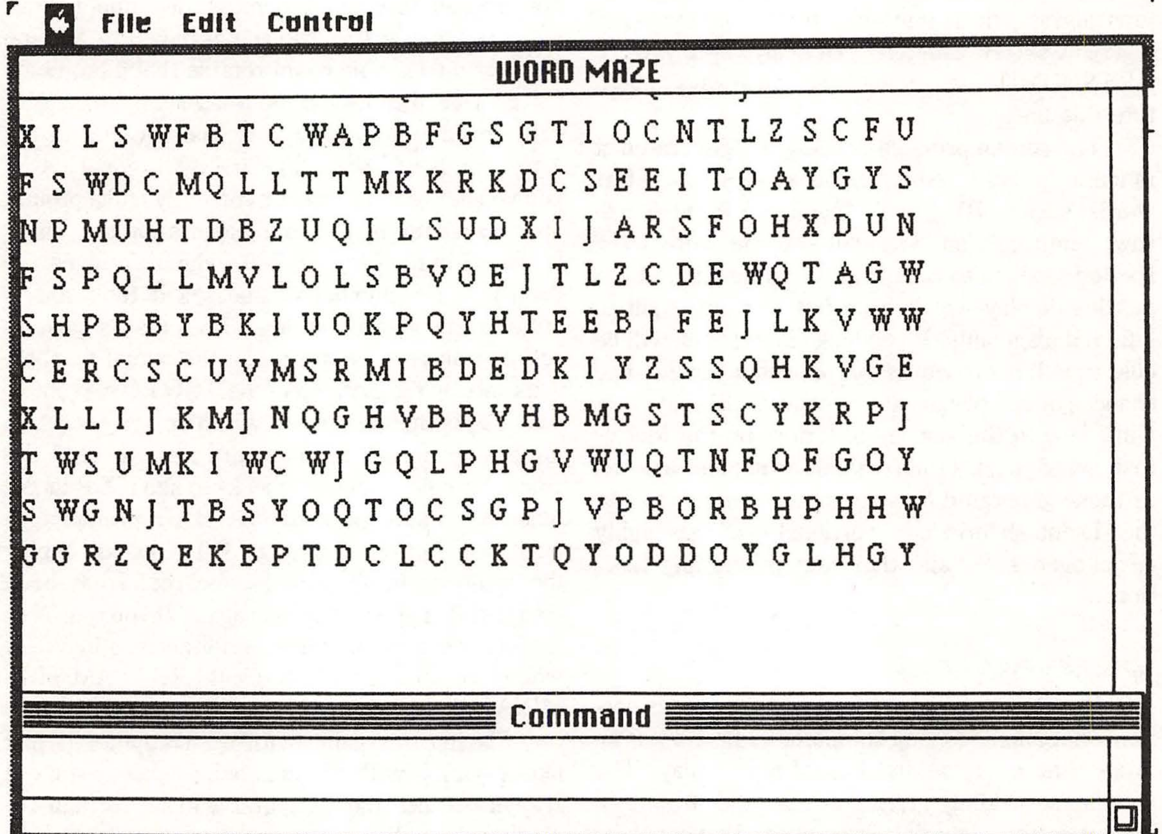


Fig. 10-15. A word maze created using the RND function.

Listing 10-23. Numbers Guess Game.

```
10 REM NUMBERS GUESS GAME
20 REM COPYRIGHT FREDERICK HOLTZ 2/84
30 CLS
40 RN=VAL(MID$(TIME$,7,2))
50 RANDOMIZE RN
60 PRINT"THE COMPUTER WILL CHOOSE A NUMBER OF FROM 1 TO 100."
70 PRINT"YOUR JOB IS TO GUESS THE NUMBER. YOUR SCORE WILL BE"
80 PRINT"KEPT BY THE COMPUTER AND REGISTERED WHEN YOU ARRIVE"
90 PRINT"AT THE CORRECT NUMBER. GOOD LUCK!!!!"
100 PRINT:PRINT
110 INPUT"PRESS <RETURN> TO CONTINUE>";A$
120 CLS
130 CN=INT(RND*100)+1
140 CLS
150 PRINT"GUESS THE NUMBER I HAVE CHOSEN. I'LL GIVE CLUES"
160 PRINT"AFTER EVERY GUESS."
170 CALL MOVETO(0,200)
180 INPUT X
190 CT=CT+1
200 IF X=CN THEN CLS:PRINT X;"IS CORRECT!!!":GOTO 240
210 IF X<CN THEN CLS:PRINT"TOO LOW. TRY AGAIN."  ELSE 230
220 FOR DLAY=1 TO 1000:NEXT DLAY:GOTO 140
230 IF X>CN THEN CLS:PRINT"TOO HIGH. TRY AGAIN.":GOTO 220
240 CALL MOVETO(0,200)
250 PRINT"IT TOOK YOU";CT;"GUESSES"
260 PRINT
270 INPUT"WOULD YOU LIKE TO PLAY AGAIN(Y/N)";PA$
280 IF PA$="Y" OR PA$="y" THEN 290 ELSE 310
290 CT=0
300 GOTO 120
310 CLS
320 CALL MOVETO(150,100)
330 PRINT "PROGRAM TERMINATED"
340 END
```

COMPUTER NUMBERS GUESS

As I mentioned previously, I have written many numbers guess programs, all of which were very similar to the one just presented. These programs are seen often in computer programming circles. However, the version shown here (Listing 10-24) is quite different. There are not as many of these. This one works the opposite of what the

Listing 10-24. Computer Numbers Guess.

```
10 REM COMPUTER NUMBERS GUESS GAME
20 REM THE COMPUTER TRYS TO GUESS THE NUMBER YOU CHOOSE
30 REM COPYRIGHT FREDERICK HOLTZ 2/84
40 HN=100
50 LN=0
60 CLS
70 INPUT"TYPE IN A NUMBER BETWEEN 1 AND 100";N
80 CLS
90 PRINT"I HAVE COMMITTED THIS NUMBER TO A MEMORY"
100 PRINT"LOCATION WHICH I CANNOT ACCESS"
110 PRINT
120 PRINT"NOW, I AM GOING TO TRY AND GUESS THE NUMBER"
130 PRINT"IF MY GUESS IS HIGH, TYPE 'H'. TYPE 'L' IF IT'S LOW."
140 PRINT"IF MY GUESS IS CORRECT, TYPE 'CORRECT'.
150 PRINT"PLEASE PLAY FAIRLY. I TRUST YOU COMPLETELY."
160 FOR DLAY=1 TO 5000:NEXT DLAY
170 X=50
180 CLS
190 CT=CT+1
200 PRINT"MY GUESS IS";X
210 PRINT
220 PRINT
230 INPUT"H,L,C";Y$
240 IF Y$="C" THEN 330
250 IF Y$="H" THEN 300
260 IF Y$="L" THEN 270
270 IF X>LN THEN LN=X
280 X=HN-INT((HN-LN)/2)
290 GOTO 180
300 IF X<HN THEN HN=X
310 X=HN-INT((HN-LN)/2)
320 GOTO 180
330 CLS
340 PRINT"IT TOOK ME";CT;"GUESSES. CAN YOU DO BETTER?"
```

former one did, at least as far as the players are concerned. This program allows you to set a number. The computer must then guess what that number is. Each time the computer displays a wrong guess, you must indicate whether it is high or low. When the computer arrives at the correct number, it displays the number of turns it took. By using the previous program along with this one, you

can play the computer on one round and let it play you on the next.

Lines 40 and 50 assign initial values to HN and LN. These variables stand for high number and low number and will be used by the computer to begin making a guess. Line 70 prompts you to type in a number between 1 and 100. This line can be excluded altogether, since the number you type in is never used by the computer. In other words, the computer has no way of knowing what number it's trying to guess based upon this input. The line is included here to make the program seem more realistic. Lines 90 through 150 allow the computer to communicate with the human player and also instruct the player what responses are expected when there is a wrong guess. In line 170, X is initially assigned a value of 50. This is the first number the computer will always guess, as it cuts its search area in half. When 50 is output as the first guess, you must tell the computer whether this number is high or low. If it's high, the computer is programmed to know that it should not come up with another number equal to or higher than 50. This means it must be a number between 1 and 49.

For the purposes of discussing this program, assume you have chosen 34 as your number. Line 200 tells the computer to output its first guess to the screen. Its guess is represented by X, which was initially assigned a value of 50. Since your number is 34, you will respond to this guess by typing H and pressing RETURN, indicating the number is too high (50 is higher than 34). Your clue is committed to Y$. Line 250 branches to line 300 because your response was H. Line 300 reassigns HN to the value of 50, since X (50) is less than the initial value of HN (100). Forever after, the computer will never output a number that is equal to or higher than 50. In line 310, the new guess number is arrived at. Here, X is assigned a value of HN (now equal to 50) minus the integer value of HN – LN divided by 2. LN was originally assigned a value of 0, so X is not equal to 50 minus the integer value of 50 minus one divided by 2, or 50 divided by 2. The integer value 50 divided by 2 is 25 and 50 minus 25 is 25. Therefore, the computer's next guess will be 25. The branch in line 320 causes the second guess to be displayed.

Now, 25 is too low, so you will type L and press RETURN. Line 260 branches to line 270 because of your choice. Here, LN (low number) is reassigned the value of 25. Its original value was 0. However, since you responded with L, the computer now knows that the number cannot be equal to or less than 25. The Macintosh is in a good position now, because it knows the number you input lies somewhere between 26 and 49. Its search area has diminished quite a bit. Line 280 makes the next guess. It assigns X in the same way it was assigned previously in line 310. This time, however, HN and LN are not separated by as many numbers. The computer's next guess will be 38. You type H because the number is too high, and again, there's a branch to line 300. HN is reassigned a new high value of 38. The next guess will be 32 and then 35, and finally, 34. When the computer displays this number, you input a C for correct, which makes line 240 to branch to line 330, where the screen is cleared and the computer prints the number of guesses it took. Here, CT is the count routine that is incremented after each guess by the computer.

The computer goes through a very structured search routine that is as efficient as possible. Therefore, it will never take the computer more than 7 guesses to arrive at the correct answer. Sometimes, it will get the answer in fewer than 7 guesses, but never more than 7. This may take some of the fun out of it, but you can learn from the computer. If you use the same search routine when running the previous program discussed, you will never require more than 7 guesses either. Be sure you input your clues properly. If you give the computer a wrong clue at any time, it will never guess the correct number.

It would be possible to build in a random

routine that would make the computer guess all numbers at random and never guess a previous selection twice, but this is not the way you play the game, so why should the computer? The original game is based on clues and you mentally throw away any numbers that lie above or below the range established by your first two guesses. Try this game. I think you'll like it.

SUMMARY

You will find the program listings included in this chapter informative, educational, fun, and practical. They pretty much run the gamut of what can be expected in the wide sampling of programs available. Certainly, no one program is terribly complex, but remember, highly complex programs are built from simple routines such as these. If you are a beginner, go over each program after it is input in an attempt to understand what each line does within the rest of the programming structure. Once you understand the program fully, you are then in an excellent position to modify it to suit your own particular requirements. These experiences will provide a wealth of knowledge that you simply can't get in any other way. Happy programming!

Appendix A
ASCII Character Codes

Dec	*Hex*	*CHR*	*Dec*	*Hex*	*CHR*	*Dec*	*Hex*	*CHR*	*Dec*	*Hex*	*CHR*
000	00H	NUL	032	20H	SPACE	064	40H	@	096	60H	'
001	01H	SOH	033	21H	!	065	41H	A	097	61H	a
002	02H	STX	034	22H	"	066	42H	B	098	62H	b
003	03H	ETX	035	23H	#	067	43H	C	099	63H	c
004	04H	EOT	036	24H	$	068	44H	D	100	64H	d
005	05H	ENQ	037	25H	%	069	45H	E	101	65H	e
006	06H	ACK	038	26H	&	070	46H	F	102	66H	f
007	07H	BEL	039	27H	'	071	47H	G	103	67H	g
008	08H	BS	040	28H	(	072	48H	H	104	68H	h
009	09H	HT	041	29H	)	073	49H	I	105	69H	i
010	0AH	LF	042	2AH	*	074	4AH	J	106	6AH	j
011	0BH	VT	043	2BH	+	075	4BH	K	107	6BH	k
012	0CH	FF	044	2CH	,	076	4CH	L	108	6CH	l
013	0DH	CR	045	2DH	–	077	4DH	M	109	6DH	m
014	0EH	SO	046	2EH	.	078	4EH	N	110	6EH	n
015	0FH	SI	047	2FH	/	079	4FH	O	111	6FH	o
016	10H	DLE	048	30H	0	080	50H	P	112	70H	p
017	11H	DC1	049	31H	1	081	51H	Q	113	71H	q
018	12H	DC2	050	32H	2	082	52H	R	114	72H	r
019	13H	DC3	051	33H	3	083	53H	S	115	73H	s
020	14H	DC4	052	34H	4	084	54H	T	116	74H	t
021	15H	NAK	053	35H	5	085	55H	U	117	75H	u
022	16H	SYN	054	36H	6	086	56H	V	118	76H	v
023	17H	ETB	055	37H	7	087	57H	W	119	77H	w
024	18H	CAN	056	38H	8	088	58H	X	120	78H	x
025	19H	EM	057	39H	9	089	59H	Y	121	79H	y
026	1AH	SUB	058	3AH	:	090	5AH	Z	122	7AH	z
027	1BH	ESCAPE	059	3BH	;	091	5BH	[	123	7BH	{
028	1CH	FS	060	3CH	<	092	5CH	\	124	7CH	|
029	1DH	GS	061	3DH	=	093	5DH	]	125	7DH	}
030	1EH	RS	062	3EH	>	094	5EH	^	126	7EH	~
031	1FH	US	063	3FH	?	095	5FH	_	127	7FH	DEL

Dec = decimal, Hex = hexadecimal(H), CHR = character,
LF = Line Feed, FF = Form Feed, CR = Carriage Return,
DEL = Rubout

Appendix B

Non-ASCII Character Codes

Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
128	80	Ä	158	9E	û	188	BC	º
129	81	Å	159	9F	ü	189	BD	Ω
130	82	Ç	160	A0	†	190	BE	æ
131	83	É	161	A1	°	191	BF	ø
132	84	Ñ	162	A2	¢	192	C0	¿
133	85	Ö	163	A3	£	193	C1	¡
134	86	Ü	164	A4	§	194	C2	¬
135	87	á	165	A5	•	195	C3	√
136	88	à	166	A6	¶	196	C4	ƒ
137	89	â	167	A7	ß	197	C5	≈
138	8A	ä	168	A8	®	198	C6	∆
139	8B	ã	169	A9	©	199	C7	«
140	8C	å	170	AA	™	200	C8	»
141	8D	ç	171	AB	´	201	C9	…
142	8E	é	172	AC	¨	202	CA	
143	8F	è	173	AD	≠	203	CB	À
144	90	ê	174	AE	Æ	204	CC	Ã
145	91	ë	175	AF	Ø	205	CD	Õ
146	92	í	176	B0	∞	206	CE	Œ
147	93	ì	177	B1	±	207	CF	œ
148	94	î	178	B2	≤	208	D0	–
149	95	ï	179	B3	≥	209	D1	—
150	96	ñ	180	B4	¥	210	D2	“
151	97	ó	181	B5	µ	211	D3	”
152	98	ò	182	B6	∂	212	D4	‘
153	99	ô	183	B7	Σ	213	D5	’
154	9A	ö	184	B8	Π	214	D6	÷
155	9B	õ	185	B9	π	215	D7	◊
156	9C	ú	186	BA	∫	216	D8	ÿ
157	9D	ù	187	BB	ª			

Appendix C
ImageWriter Printer Specifications

Print Method:	Dot matrix, logic seek (line by line)
Printing Speed:	At 10 characters per inch: 120 characters per second 72 lines per minute
Character Format:	Standard characters: Up to 7 dots wide by 8 dots high Custom (down-loaded) characters: Up to 16 dots wide by 8 dots high
Standard Characters:	96 ASCII (alphanumeric and symbols) 25 European language characters
Vertical Dot Spacing:	1/72 of an inch
Line Length:	8 inches maximum

Horizontal Pitches:

Characters per Inch	Characters per Line	Dots per Inch (Approx.)
17	136	136
15	120	120
13.4	107	107
12	96	96
10	80	80
9	72	72
8.5	68	136
7.5	60	120
6.7	53.5	107
6	48	96
5	40	80
4.5	36	72
variable	variable	160
variable	variable	144

Paper Feed Direction:	Forward and reverse
Line Spacing:	1/144 to 99/144 of an inch, selectable in increments of 1/144 of an inch
Line Feed Method:	Stepper motor drive
Line Feed Speed:	Maximum 10 per second at 6 lines per inch

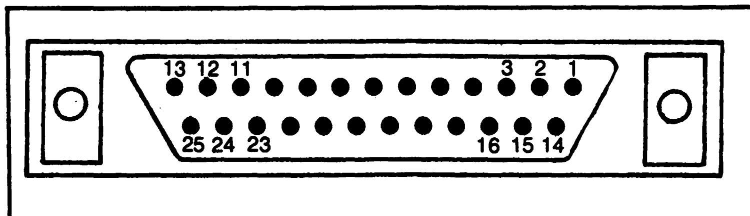
Paper Width:	3 to 10 inches
Paper Thickness:	0.05-0.28 millimeter (0.002-0.011 inch) Original + 3 copies maximum
Paper Feed Method:	Selectable, friction or sprocket/pin feed
Paper Types:	Single sheets Roll paper Fanfold sprocketed paper (hole centers 4.0-9.5 inches)
Paper Entry:	Top rear of printer
Ribbon:	Cassette containing inked fabric ribbon (black recommended), 13 millimeters wide by 13,000 millimeters long, automatically reversing
Power Options:	115 volts ac ± 10%, 60 hertz 100 volts ac ± 10%, 50/60 hertz 220 volts ac ± 10%, 50 hertz 240 volts ac ± 10%, 50 hertz
Power Consumption:	Operating: 180 watts maximum Standby: 16 watts maximum
Data Interface:	8-bit serial (see Appendix F)
Weight:	8.5 kilograms (18.75 pounds)

Dimensions:

Width	Depth	Height
398	285	125 millimeters
15.7	11.3	5.3 inches

Ambient Temperature:	
Operating	5 to 40 degrees Celsius (41 to 104 degrees Fahrenheit)
Storage	−25 to +60 degrees Celsius (−13 to +140 degrees Fahrenheit)
Maximum Humidity:	
Operating	85% relative humidity, noncondensing
Storage	90% relative humidity, noncondensing

Data Input Form:	7-bit or 8-bit serial: 1 start bit, data bits (7 or 8), and 1 stop bit (no parity bit)
Data Input Codes:	Characters: ASCII, 8- or 7-bit Graphics: 8-bit binary
Transmission Speed:	300, 1200, 2400, or 9600 baud
Input Buffer Size:	1 K bytes
Printer Connector:	DB-25 male, or equivalent
Mating Connector:	DB-25S female, or equivalent



Pin No.	Symbol	Description	Direction
1	FG	Frame Ground	
2	SD	Send Data	Output
3	RD	Receive Data	Input
4	RTS	Request to Send	Output
7	SG	Signal Ground	
14	FAULT	Fault	Output
20	DTR	Data Terminal Ready	Output

Appendix D
ASCII, Binary, and Hexadecimal Print Codes

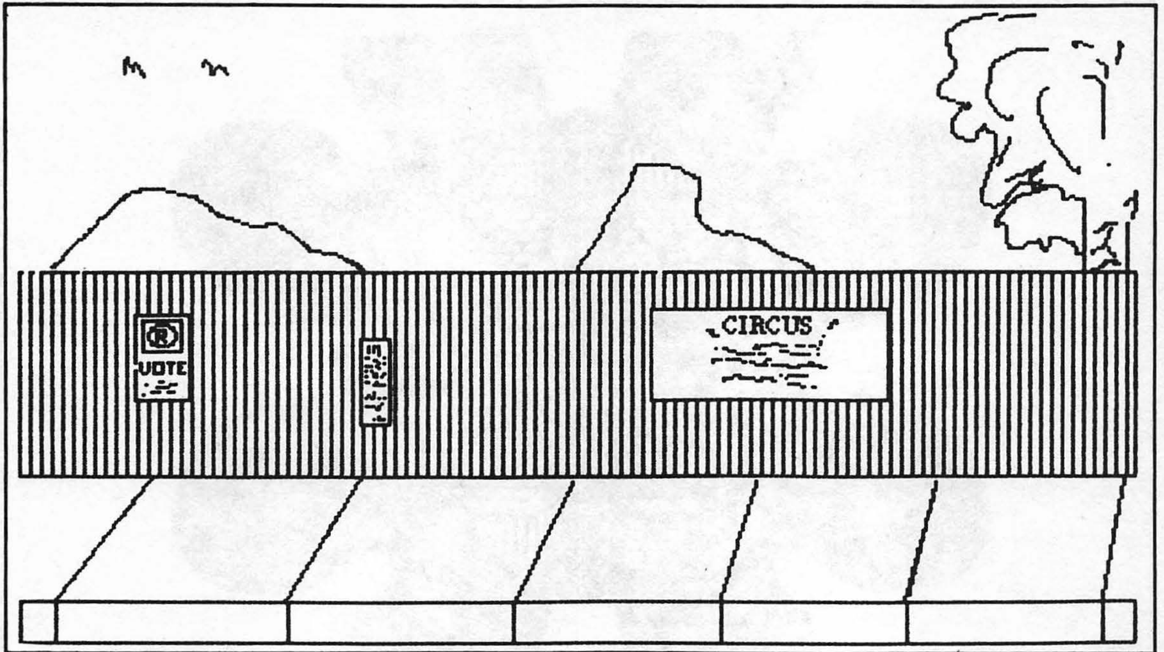
ASCII		Low ASCII Dec	Low ASCII Hex	High ASCII Dec	High ASCII Hex
CONTROL-@	NUL	0	00	128	80
CONTROL-A	SOH	1	01	129	81
CONTROL-B	STX	2	02	130	82
CONTROL-C	ETX	3	03	131	83
CONTROL-D	EOT	4	04	132	84
CONTROL-E	ENQ	5	05	133	85
CONTROL-F	ACK	6	06	134	86
CONTROL-G	BEL	7	07	135	87
CONTROL-H	BS	8	08	136	88
CONTROL-I	HT	9	09	137	89
CONTROL-J	LF	10	0A	138	8A
CONTROL-K	VT	11	0B	139	8B
CONTROL-L	FF	12	0C	140	8C
CONTROL-M	CR	13	0D	141	8D
CONTROL-N	SO	14	0E	142	8E
CONTROL-O	SWI	15	0F	143	8F
CONTROL-P	DLE	16	10	144	90
CONTROL-Q	DC1	17	11	145	91
CONTROL-R	DC2	18	12	146	92
CONTROL-S	DC3	19	13	147	93
CONTROL-T	DC4	20	14	148	94
CONTROL-U	NAK	21	15	149	95
CONTROL-V	SYN	22	16	150	96
CONTROL-W	ETB	23	17	151	97
CONTROL-X	CAN	24	18	152	98
CONTROL-Y	EM	25	19	153	99
CONTROL-Z	SUB	26	1A	154	9A
	ESC	27	1B	155	9B
	FS	28	1C	156	9C
	GS	29	1D	157	9D
	RS	30	1E	158	9E
	US	31	1F	159	9F
	SP	32	20	160	A0
	!	33	21	161	A1
	"	34	22	162	A2
	#	35	23	163	A3
	$	36	24	164	A4
	%	37	25	165	A5
	&	38	26	166	A6
	'	39	27	167	A7
	(	40	28	168	A8
	)	41	29	169	A9
	*	42	2A	170	AA
	+	43	2B	171	AB
	,	44	2C	172	AC
	-	45	2D	173	AD
	.	46	2E	174	AE
	/	47	2F	175	AF
	0	48	30	176	B0
	1	49	31	177	B1
	2	50	32	178	B2
	3	51	33	179	B3

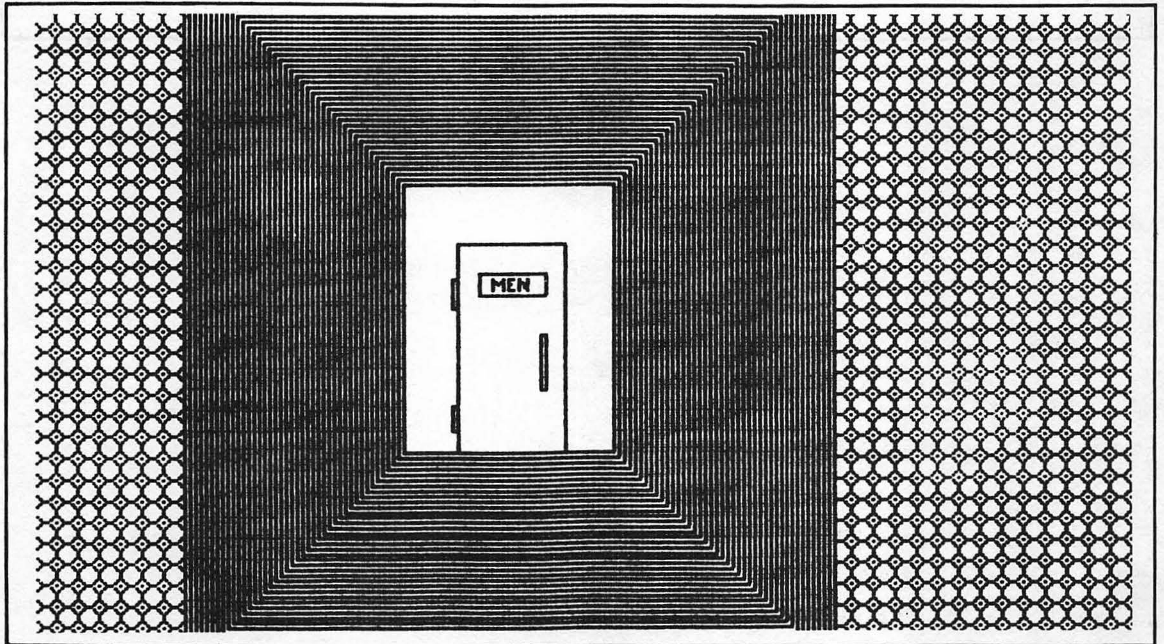
ASCII	Low ASCII Dec	Hex	High ASCII Dec	Hex
4	52	34	180	B4
5	53	35	181	B5
6	54	36	182	B6
7	55	37	183	B7
8	56	38	184	B8
9	57	39	185	B9
:	58	3A	186	BA
;	59	3B	187	BB
<	60	3C	188	BC
=	61	3D	189	BD
>	62	3E	190	BE
?	63	3F	191	BF
@	64	40	192	C0
A	65	41	193	C1
B	66	42	194	C2
C	67	43	195	C3
D	68	44	196	C4
E	69	45	197	C5
F	70	46	198	C6
G	71	47	199	C7
H	72	48	200	C8
I	73	49	201	C9
J	74	4A	202	CA
K	75	4B	203	CB
L	76	4C	204	CC
M	77	4D	205	CD
N	78	4E	206	CE
O	79	4F	207	CF
P	80	50	208	D0
Q	81	51	209	D1
R	82	52	210	D2
S	83	53	211	D3
T	84	54	212	D4
U	85	55	213	D5
V	86	56	214	D6
W	87	57	215	D7
X	88	58	216	D8
Y	89	59	217	D9

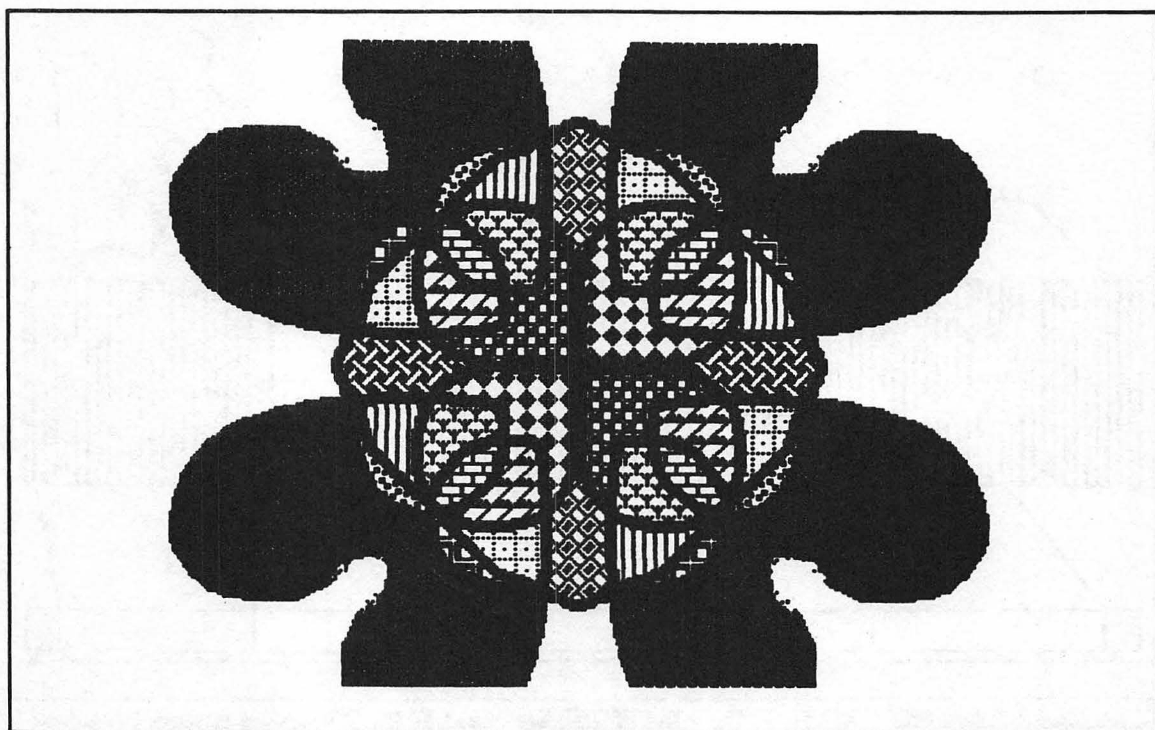
ASCII	Low ASCII Dec	Hex	High ASCII Dec	Hex
Z	90	5A	218	DA
[	91	5B	219	DB
\	92	5C	220	DC
]	93	5D	221	DD
^	94	5E	222	DE
_	95	5F	223	DF
`	96	60	224	E0
a	97	61	225	E1
b	98	62	226	E2
c	99	63	227	E3
d	100	64	228	E4
e	101	65	229	E5
f	102	66	230	E6
g	103	67	231	E7
h	104	68	232	E8
i	105	69	233	E9
j	106	6A	234	EA
k	107	6B	235	EB
l	108	6C	236	EC
m	109	6D	237	ED
n	110	6E	238	EE
o	111	6F	239	EF
p	112	70	240	F0
q	113	71	241	F1
r	114	72	242	F2
s	115	73	243	F3
t	116	74	244	F4
u	117	75	245	F5
v	118	76	246	F6
w	119	77	247	F7
x	120	78	248	F8
y	121	79	249	F9
z	122	7A	250	FA
{	123	7B	251	FB
|	124	7C	252	FC
}	125	7D	253	FD
~	126	7E	254	FE
DEL	127	7F	255	FF

Appendix E
More *MacPaint* Drawing Samples

THE SHADOW WINDOW EFFECT IS PRODUCED BY LAYING DOWN A SOLID BLACK WINDOW OVERLAYED BY A SOLID WHITE ONE USING THE RECTANGLE FILL TOOL ▬ FOR BOTH IMAGES.

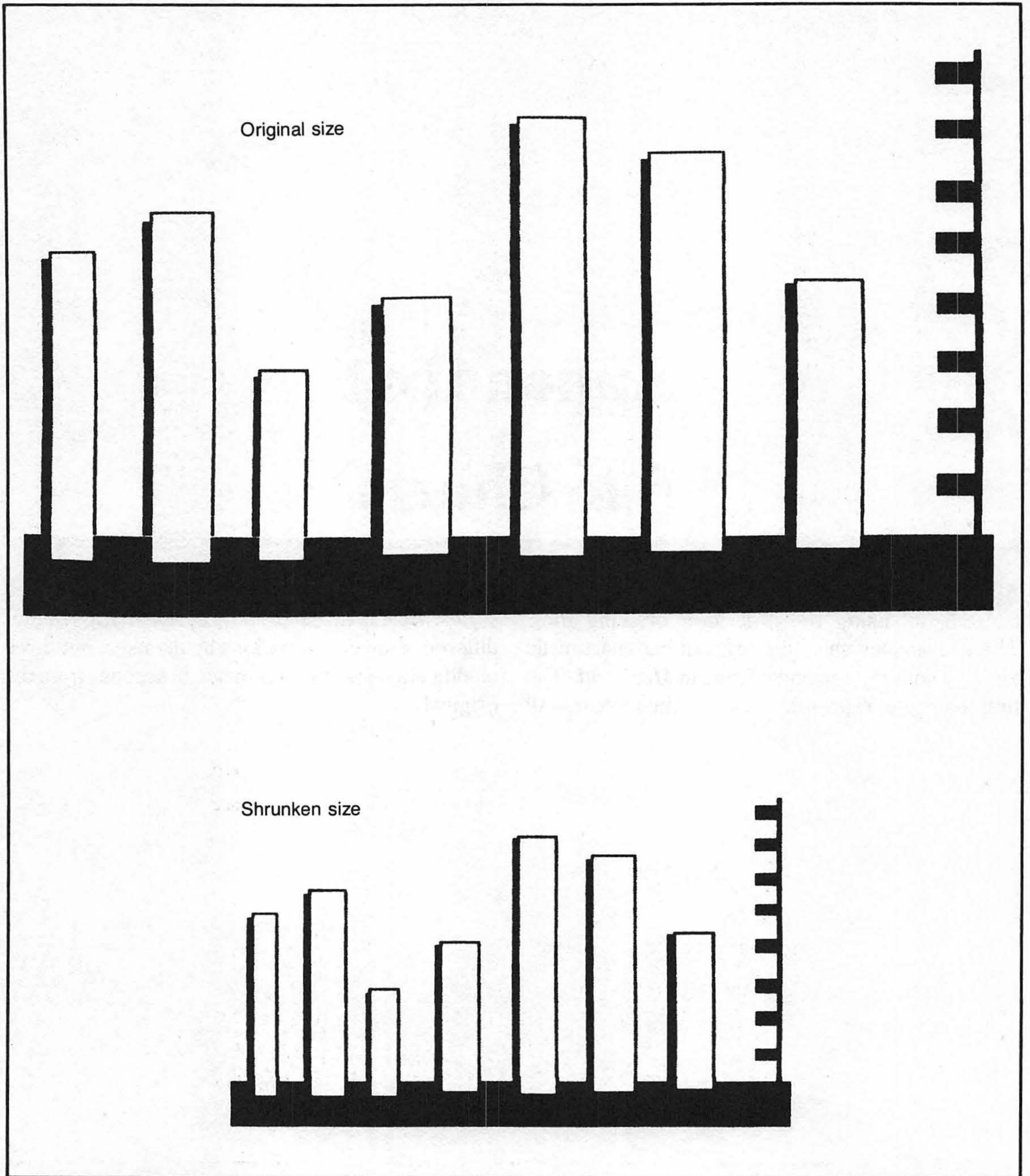

VOTE
CIRCUS

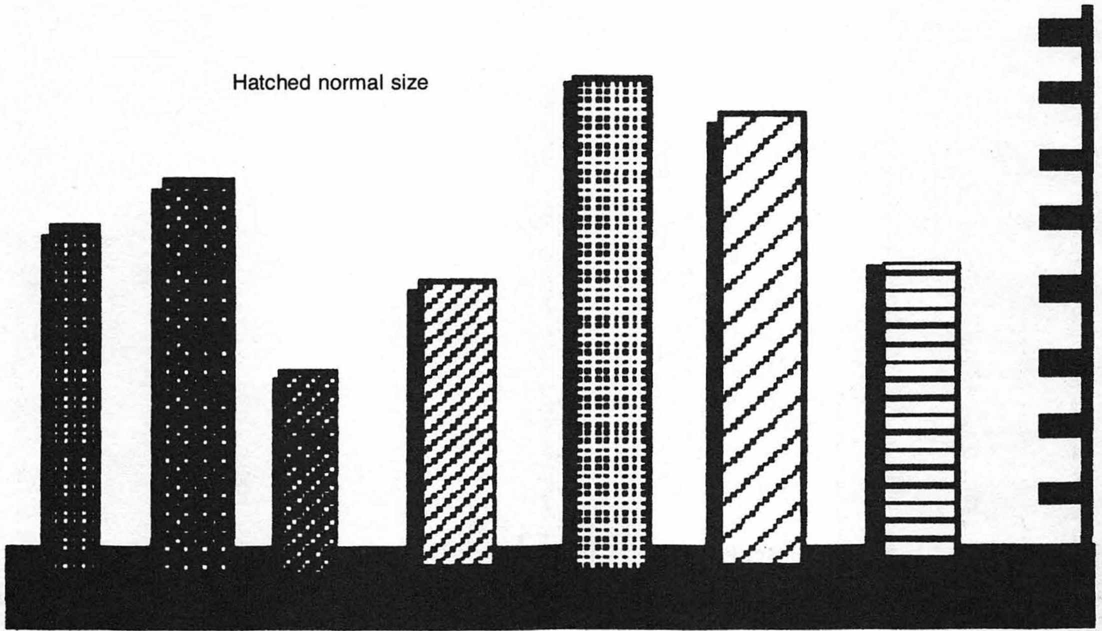

MEN

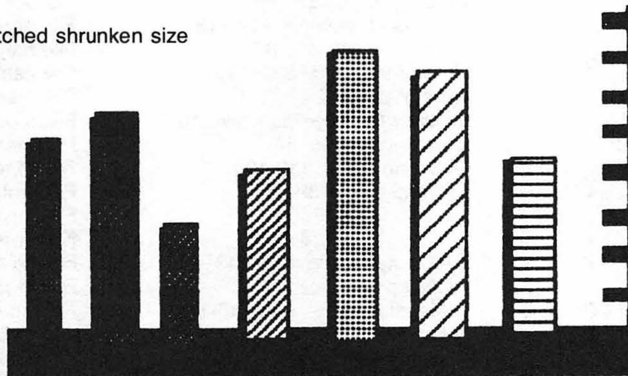


Appendix F
Bar Charts

All types of charts can be quickly and accurately constructed using the *MacPaint* drawing tool. These examples show the original bar chart made entirely with the rectangle icons in *MacPaint.* The first example represents the original chart. All others show this same chart in shrunken or hatched styles. Again, these are not representative of four different drawing operations by the user, but three modifications which were made in seconds from the original.


Original size
Shrunken size


Hatched normal size


Hatched shrunken size

Index

A

Absolute, the method, 152
Alarm clock, 25
Aligning text, 45
Alphanumeric variables, 127
AND, 144-145
Animation techniques, 157-160
Appending, files, 172
Apple II series, 4
Apple logo menu, 25
Apple logo menu, Finder, 73-81
Arrays, 146-147
Assignment statement, 119
AUTO command, 92
BASIC, Programming, 115-147
BASIC, Microsoft, 86-87
BASIC, types of, 86-87
BEEP statement, 160
Border icon, 24, 41
Branching, 119-121, 141-142
Breaking, a program, 92

C

Calculator, 25
CALL LINETO function, 184
Chip, 1
CINT function, 139
Circle icon, 32
CIRCLE statements, 150-152
Clearing screen, 117-118
CLS statement, 92, 117-118
Code, 5
Color, use of, 153-154
Command, defined, 117-118
Command window, 87
Control menu, BASIC, 113
Control panel, 25
Convert to integer function, 139
Correcting typos, 54
Counting loops, 119-120
Cut and paste, 39-41

D

DATA statement, 142p143
Digitized graphics, 16
DIMension statement, 146-147
Direct mode, BASIC, 118
Draft, printing document, 63-65
Editing, BASIC, 96-98-99
Edit menu, BASIC, 109-112
Edit menu, Finder, 81
Edit menu, *MacPaint*, 26
END statement, 121
Enlarging, 39-41
Eraser icon, 23
Error trapping, 145-146

F

Fat bits, 37
File appending, 172
File handling, 85
File item, searching for, 174-177
File menu, BASIC, 106-109
File menu, Finder, 81
File menu, *MacPaint*, 26
File reading, 168-169
File writing, 169-171
Filing program, complete, 172-174
Finder menu, 68-85
Finder, the, 68
Finder menu, 68-85
Font menus, *MacPaint*, 28
Fonts, 61
Formulas, 133-134
FOR-NEXT loops, 121-125
FOR-NEXT loops, STEPping, 123-125
GET statement, 156

Goodies menu, *MacPaint*, 28
GOSUB-RETURN, 141-142
GOTO, 118-119
Graphics tools, 20-22

H
Hand icon, 23

I
IBM PC, 1
Icons, 2
IF-THEN, 120-121
Incrementing loops, 119-120
Indenting text, 56
INPUT statements, 130-133
Integer function, 136
INT function, 136

J
Joystick, 2
Justifying text, 45

K
Keyboard interface icon, 25

L
Lariat icon, 24
LEFT$ function, 139-140
LEN function, 135-136
LET, 119-120
Letter "A" icon, 25
Line maker icon, 25
LINE statements, 152-153
LINETO, 184
LINETO-CALL function, 184
Lisa, 2, 17
Lisa 2, 19
List window, 87
List windows, 99-102
List windows, multiple, 99-103
Logical operators, 144-145
Loop, creating, 118
Loop, defined, 118
Loops, FOR-NEXT, 121-125
Loops, incrementing, 119-120

M
MacBASIC, 86-87
Macintosh chip, 6
Macintosh software, 16

Macpaint, 20
Mathematical operations, 129
Menu bar, MacPaint, 20-25
Menu-driven, defined, 5
Microprocessor, 1
MID$, function, 140
Mouse, 2
MOUSE function, 179-184
MOUSE(0) function, 179-181
MOUSE(1) function, 181-183
MOUSE(2) function, 183-184
MOUSE(3) function, 185-189
MOUSE(4) function, 185-189
MOUSE(5) function, 189-190
MOUSE(6) function, 189-190
Mouse, clicking, 20
Mouse, maintenance, 178
Mouse, programming, 178-190
Multiple-statement lines, 143-144

N
NEW, 92
Numeric variables, 127

O
OR, 144-145
Output window, 87
Oval icon, 25

P
Paintbrush icon, 23
Paint bucket icon, 23
Paint sprayer icon, 23
Patterns, graphic, 32-33
Pencil icon, 20, 22
Pie chart, 32
POINT function, 163-166
Pointing technology, 2
PRESET statement, 154-157
Price, 5, 16, 19
Printing, draft, 63-65
Printing, high quality, 65-66
Printing, standard quality, 65-66
Printing, variables, 127-128
Program mode, 118
PSET statement, 154-157
PUT statement, 156

Q
Quickdraw routines, 20

R
Random function, 136-139
READ statement, 142-143
READ/DATA statements, 142-142
Reading, files, 168-169
Rectangle icon, 24
Rectangle icon, rounded, 25
Relational operators, 145-146
REM statement, 92
RESTORE statement, 143
RIGHT$ function, 140
RND function, 136-139
ROM chips, 4
ROM routines, 20, 184
Rounding function, 139
Rubber band line icon, 23, 32
Rubber band line maker icon, 25
Ruler, formatting, 56-61

S
Saving documents, 66
Scrapbook, 25
Screen resolution, 22, 149-150
Scrolling, screen, 94
Search and change function, 47-54
Searching, file item, 174-177
Searching text, 47
Setting margins, 56
Shrinking, 39-41
Size, type, 61
Software, 6
Spacing text, 45
Special menu, Finder, 85
Statement, defined, 117-118
STEPping FOR-NEXT loops, 123-125
Strings, 127
String variables, 127
String variables, adding, 129-130
Style, type, 61-63
Symbols, mathematical operations, 129

T
Tab, decimal, 61
Tab, regular, 61
Tabbing within text, 56
TIME$ function, 140-141
Typefaces, 61

U
Undo, 35

V
Variables, naming, 128-129
Variables, types, 126-127
VIEW menu, Finder, 72

W
WIDTH statement, 91
Window, command, 87
Window, list, 87
Window, output, 87
Windows, BASIC, 87-111
Windows, list, 99-103
Windows, moving, 103-106
Wrap, around, 45
Writing, files, 169-171

Using and Programming the Macintosh™ including 32 Ready-to-Run Programs

by Frederick Holtz

Apple's exciting new Macintosh is being hailed as the most revolutionary microcomputer of our time . . . offering incredibly advanced capabilities including super-high-resolution graphics, desktop icons, sophisticated word processing abilities, plus compact, "totable" hardware and affordable price.

And keeping right in step with Mac's new directions in personal computing, this comprehensive, up-to-date guide gives you all the helpful advice and step-by-step instructions needed to fully understand and use all your Mac's special features.

Complete with unique programs and routines designed to get you started, this is your springboard to a whole new world of creative computing action. Here's where you'll find all the basic how-to's needed by the first-time programmer . . . as well as the advanced guidance and programming shortcuts experienced programmers are looking for. You'll explore:

- How and why the Macintosh was developed.
- The Mac's unique features and operation.
- BASIC programming with the Mac.
- Creating graphics with MacPaint.
- Word processing with MacWrite.
- Fundamental programs for the Mac.
- Graphics programs for the Mac.
- Accessories and add-ons available for the Macintosh.
- Plus a look at Mac software that is—or soon will be—available.

Written by an author whose latest programming guide for TAB sold over 100,000 copies in less than a year, this is a sourcebook that takes you beyond the information included in standard operating manuals and gives you a head start in your own creative programming efforts!

OTHER POPULAR TAB BOOKS OF INTEREST

Using and Programming the IBM PCjr.®, including 77 Ready-to-Run Programs (No. 1830—$11.50 paper; $16.95 hard)

Using and Programming the ADAM™, including Ready-to-Run Programs (No. 1706—$7.25 paper; $10.95 hard)

TAB BOOKS Inc.

Blue Ridge Summit, Pa. 17214

Send for FREE TAB Catalog describing over 750 current titles in print.

FPT > $12.50

ISBN 0-8306-1840-6

PRICES HIGHER IN CANADA

1195-0484