The TIME LIFE Step-by-Step Guide to the Commodore 64

The illustrated guide to mastering the Commodore 64 and its uses

A Random House Book

Useful Terms

BASIC. Acronym for Beginners All-purpose Symbolic Instruction Code. BASIC is the most common high-level computer language used in microcomputers.

Bit. The smallest unit of computer memory; an acronym for BInary digiT.

Byte. The most common unit of computer memory, equivalent to eight bits. A byte is the amount of memory required to store a single letter, numeral or punctuation mark.

Data. Any information, such as letters, numbers, words, sentences or symbols, that the computer stores or processes.

DOS. Acronym for Disk Operating System. Includes the programs that control the basic operations of the disk drive.

File. A collection of logically related information that the computer keeps together for processing and storage.

Floppy disk. A circular piece of flexible plastic coated with metal oxide, used by the 64 for long-term information storage. Information is recorded on the surface of the disk in the same way that music is recorded on magnetic tape. A floppy disk can be removed from the disk drive, which is connected to the computer; when the information on it is required, it is reinserted. Floppy disks are also called disks or diskettes.

Hardware. The physical components of the computer system.

IC. Integrated circuit, a silicon chip that combines many electronic components in a small space. The primary building block of computer systems.

I/O. Input/Output. *See Input and Output.*

Input. Information put into the computer from an outside source. The computer is said to read when it receives input.

Memory. The part of the computer where programs and information are kept while the computer is operating. The 64 has two general types of memory. The first, read-only memory (ROM), holds data that cannot be changed or erased. The second, read-and-write memory, can be expanded or erased by the user. Its contents are lost when the computer is turned off.

Microcomputer. A microprocessor-based computer that normally processes eight or 16 bits of information at a time.

Microprocessor. The "brains" of a small computer, usually a single integrated-circuit (IC) chip.

Operating system. A program that controls the most fundamental operations of a computer.

Output. Information transferred from the computer to the outside world. The computer is said to write when it produces output.

Peripherals. Accessories that can be connected to the system unit of a computer to provide additional features. Common peripherals are monitors, disk drives, printers, modems and joysticks.

Program. A list of instructions that tell the computer, step by step, how to carry out a task or solve a problem.

Random access memory. The type of read-and-write memory installed in the 64. Random access memory is designed so that the computer can identify and locate any single item stored in memory without going through the entire contents of the memory. *See Memory.*

ROM. Read-only memory. *See Memory.*

Software. Programs that tell the hardware of the computer system how to perform its tasks.

The TIME-LIFE Step-by-Step Guide to the Commodore 64

Random House New York

A Versatile Computer System

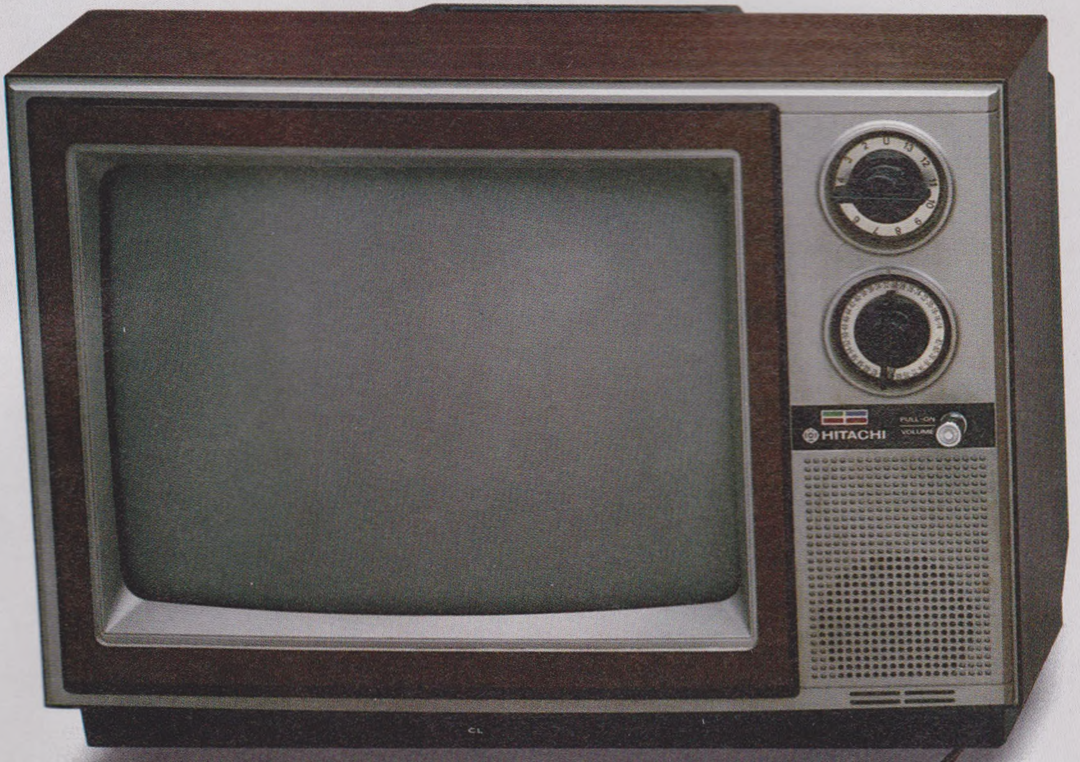
The Commodore 64 computer can be the core of many computer systems, each designed for completely different applications. In a scientific laboratory, a 64 can control equipment, store experimental data and compile results. An artist might use a 64 to create original images or to manipulate existing ones by removing lines, altering colors and rotating shapes. In a business office, a 64 may calculate accounts and keep financial records or do word processing and maintain correspondence files. At home, you can use a 64 to keep budgets, to help you learn a language, to communicate with other computers, to play games or to create music.

A computer can do all these jobs because it is designed to control a multitude of accessories used for specialized applications. Commodore supplies some of these accessories, which are called peripherals, but other manufacturers also produce peripherals that are compatible with the Commodore 64. You can tailor a system to your needs and budget by linking peripheral components to the one piece of equipment essential to any system — the Commodore 64 keyboard unit, which contains the computer itself.

The system shown on these pages includes the components needed for the most common applications of the 64 — word processing, budgeting, display of color graphics, games, storage of information and the printing out of results. Your system may include some or all of these components. This manual will show you the basic principles of operating any Commodore 64 system and explore some of the ways in which you can use the computer for specialized applications.

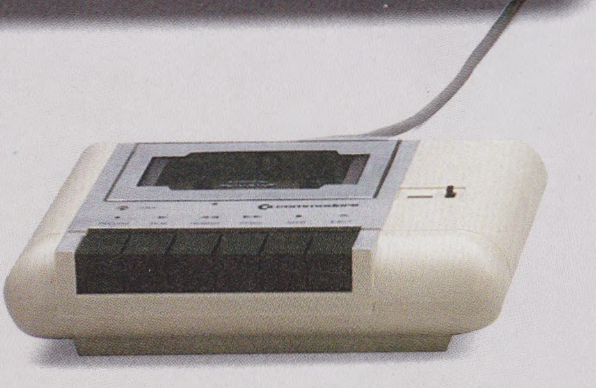
An Adaptable Array of Equipment

Using the household television set to display work helps keep the cost of a Commodore system low. Whether black and white or color, a TV is usually adequate for games and educational programs.



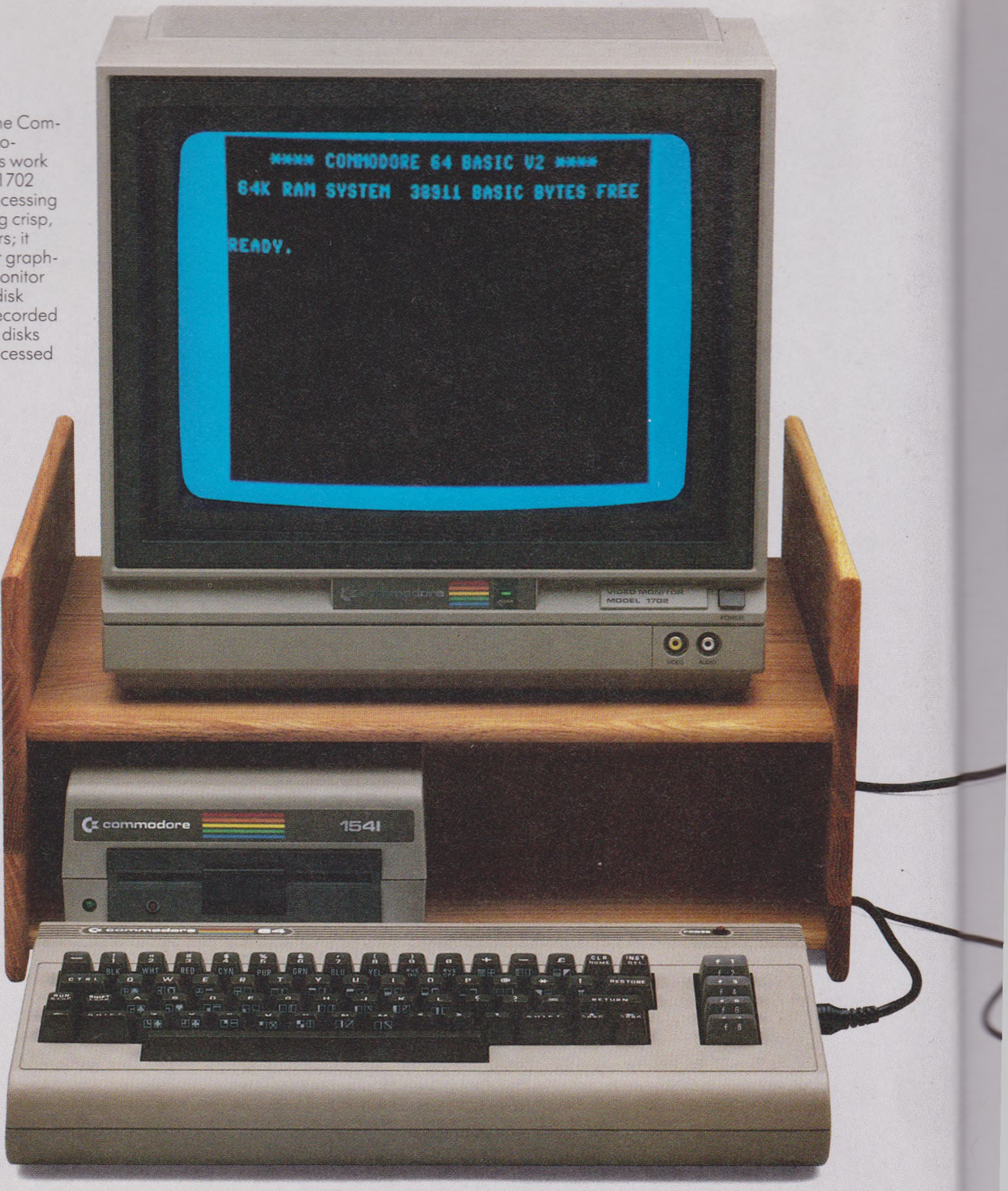
A color monitor
modore 1702) c
chrome monitor
as it is being do
serves well for v
and other tasks
clear letters and
also produces fi
ic displays. Belo
is a Commodor
drive, which rea
information fron
or copies comp
information ont

A modem connects the computer to telephone lines so it can communicate with other computers. The modem translates the computer's electronic information codes into signals that telephone lines can carry.



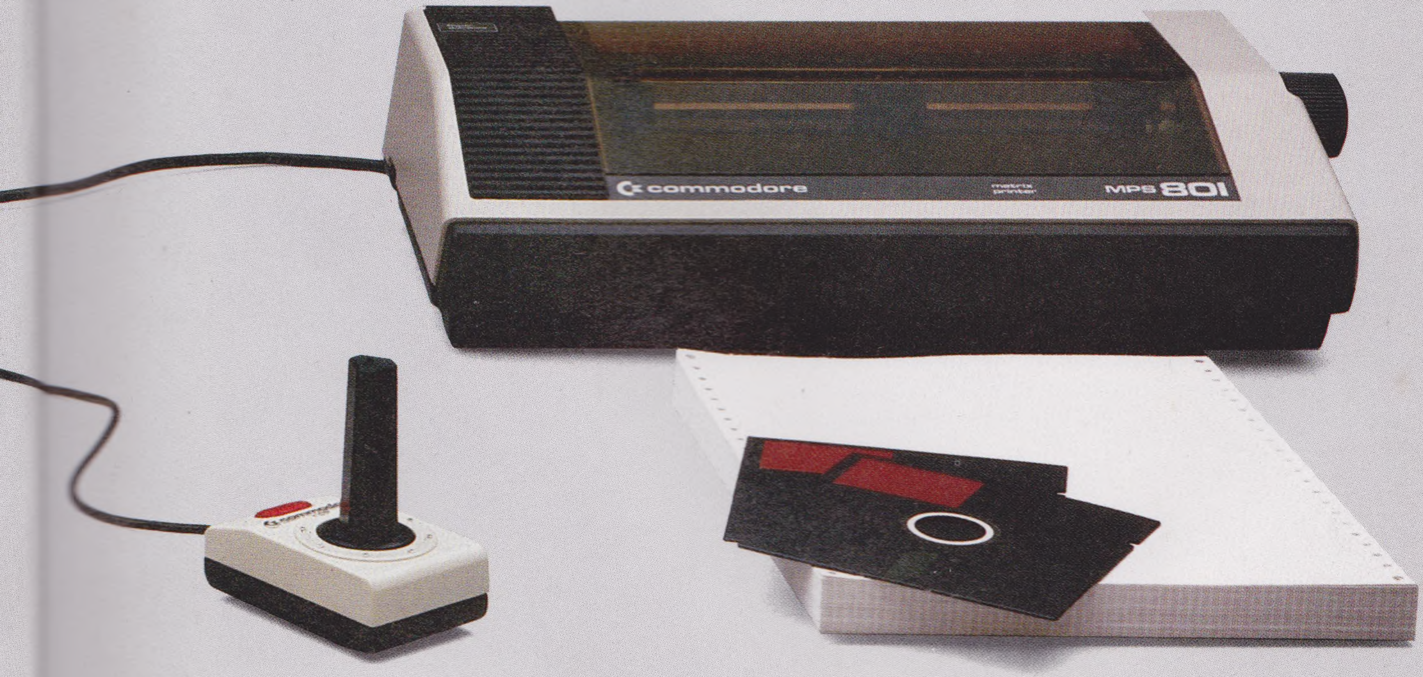
A digital cassette recorder transfers prerecorded information from magnetic tape into the computer and, conversely, from the computer onto tape.

A color monitor (here, the Commodore 1702) or a monochrome monitor displays work as it is being done. The 1702 serves well for word processing and other tasks requiring crisp, clear letters and numbers; it also produces fine color graphic displays. Below the monitor is a Commodore 1541 disk drive, which reads prerecorded information from floppy disks or copies computer-processed information onto disks.


**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.
commodore
VIDEO MONITOR MODEL 1702
commodore 1541
commodore 64

The keyboard unit is the heart of the computer system, housing the electronic circuits that process and store information. The keyboard is the device most commonly used to enter information into the computer, and it also controls the workings of all the other parts of the system.

A printer produces a paper record of the computer's work. The Commodore 801 printer, shown here, produces rough but clear type; more specialized printers produce fine-quality print or color graphics.



A joystick is used to control the computer for games. Joysticks may also be connected in pairs, so that more than one person can play the game at once.

Floppy disks, which are used with a disk drive, are kept in plastic sleeves to prevent damage and consequent distortion of the information stored on them. The perforated paper under these disks is for the printer.

Created and designed by
TIME-LIFE BOOKS INC.

EDITOR: George Constable
Executive Editor: George Daniels
Director of Design: Louis Klein
Director of Administration: David L. Harrison
Director of Research: Phyllis K. Wise
Director of Photography: John Conrad Weiser

PRESIDENT: Reginald K. Brack Jr.
Senior Vice President: William Henry
Vice Presidents: George Artandi, Stephen L. Bair, Robert A. Ellis, Juanita T. James, Christopher T. Linen, James L. Mercer, Joanne A. Pello, Paul R. Stewart

THE TIME-LIFE STEP-BY-STEP
GUIDE TO THE COMMODORE 64

Editor: Peter Pocock
Book Manager: Judith W. Shanks
Designers: Ellen Robling, Edward Frank
Associate Editor: Sarah Brash (text)
Project Administrator: Caroline A. Boubin
Picture Editor: Linda Lee
Assistant Designer: Robert K. Herndon
Copy Coordinator: Margery duMond
Picture Coordinator: Nancy L. Scott
Special Contributors: Bruce O. Boston, Marilynne Rudick, Brooke Stoddard (text)

EDITORIAL OPERATIONS
Copy Room: Diane Ullius
Production: Anne B. Landry (director), Celia Beattie
Quality Control: James J. Cox (director), Sally Collins
Library: Louise D. Forstall

Published in the United States by Random House, Inc., New York, and simultaneously in Canada by Random House of Canada Limited, Toronto.

Library of Congress Cataloguing in Publication Data
Main entry under title:

THE TIME-LIFE STEP-BY-STEP
GUIDE TO THE COMMODORE 64.
Bibliography: p.
Includes index.
1. Commodore 64 (Computer)
I. Time-Life Books.
QA76.8.C64T56 1984 001.64 83-24566
ISBN 0-394-72515-8

Manufactured in the U.S.A.
98765432
First Edition

THE CONSULTANTS

Jim and Ellen Strasma are the editors of *The Midnite Software Gazette,* a highly respected magazine for Commodore computer owners. They have founded two prominent Commodore user groups, and written six books about the Commodore 64. Both teach at Lincoln College in Lincoln, Illinois, where their Commodore Computer Camp is a regular summer event.

Loren Wright, a contributing editor and columnist for *Micro* magazine, is a writer, programmer and photographer specializing in the Commodore 64. He has worked with Commodore computers since 1977, when a graduate project at the University of California in Berkeley called for the use of a Commodore PET. He programmed many of the graphics screens for this book.

Jim Butterfield, associate editor of *Compute!* magazine, is a well-known computer journalist and lecturer. Since retiring from CNCP Telecommunications after 25 years in new technology and communications, he has produced a videotape instruction course on the Commodore 64 and written numerous articles on diverse computer subjects; he is now completing the manuscript on a new book about machine language.

PHOTOGRAPHS by Larry Sherer

Contents

commodore
VIDEO MONITOR
MODEL 1702
POWER
commodore
1541
commodore
64
POWER

The Hardware: Set-up to Start-up

Getting your Commodore 64 computer ready to use is easily accomplished if you take the process step by step. All you have to do is unpack the components, position them, connect them with cables and turn the system on. The only tool you need is a screwdriver. As for the time the job takes, give yourself at least an hour.

Set up the system where you will be comfortable using it. Pick a place away from direct sunlight and heat: The computer needs to stay cool, and you need to avoid glare on the display screen.

Ideally, the work surface should allow you to place the monitor at eye level for comfortable viewing. There are two-level desks designed for computer systems, but you can improvise a satisfactory arrangement by setting a monitor stand about 10 inches tall on a table. The table should be of a convenient height for typing — about 26 inches — and have a surface at least 30 inches by 36 inches — large enough for the system components, some papers and a lamp.

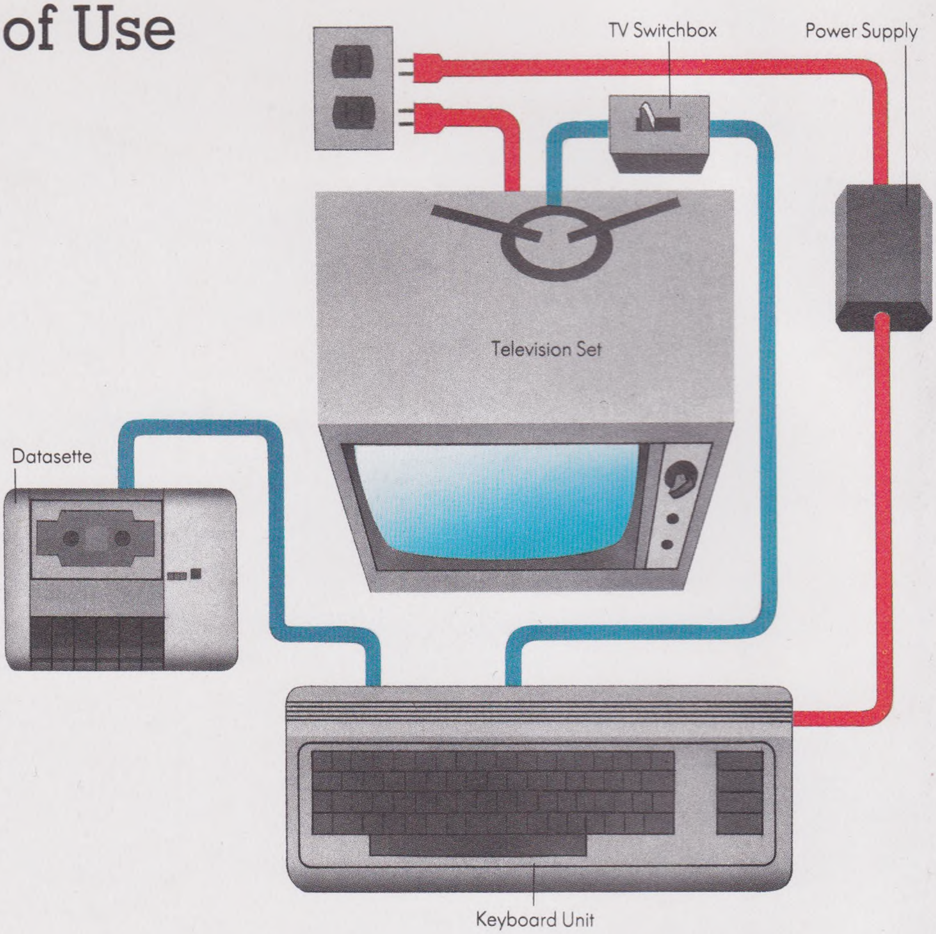
The system components are packed in sturdy boxes and, within those, in plastic bags and foam inserts that prevent the accumulation of static electricity, which could wreak havoc with the electronics of the system. Save all the packing materials; you will need them to protect the equipment anytime you move it. Check through each carton before storing it, to make sure you have removed all cables, connectors, tools and printed matter, including warranties. You will want to familiarize yourself with the *Commodore 64 User's Guide,* which is packaged with the keyboard unit.

The inside of each component is a marvel of electronic and mechanical complexity, but you don't need to know how a computer works before you start using it. If anything doesn't work after you have set up the system, retrace your steps and try again. If it still doesn't work, call the dealer from whom you purchased it.

Two systems are shown in this chapter. The simpler of the two consists of the Commodore 64 keyboard unit, a television set, and a Datasette recorder. The second system is composed of four Commodore components: the keyboard unit, a Commodore 1702 color monitor, a 1541 disk drive and an 801 printer. More specialized components such as communications devices and game attachments are discussed in later chapters.

The core of the Commodore 64 system is the keyboard unit, which houses information-processing circuits. In the arrangement shown here, the keyboard unit is linked to a Commodore 1702 monitor, which displays the computer's output, and to a disk drive (on the shelf below), which records data or feeds it into the computer.

Putting Together the Pieces for Different Kinds of Use



The components of a basic Commodore 64 system are the keyboard unit, the power supply, a television set, and a Datasette recorder. Cables that carry the electricity for powering each component are shown here in red. The power supply converts the household's 110-volt alternating current to the low-voltage direct current required by the computer. Other cables *(blue)* carry the electrical impulses that encode information to be processed by the components.

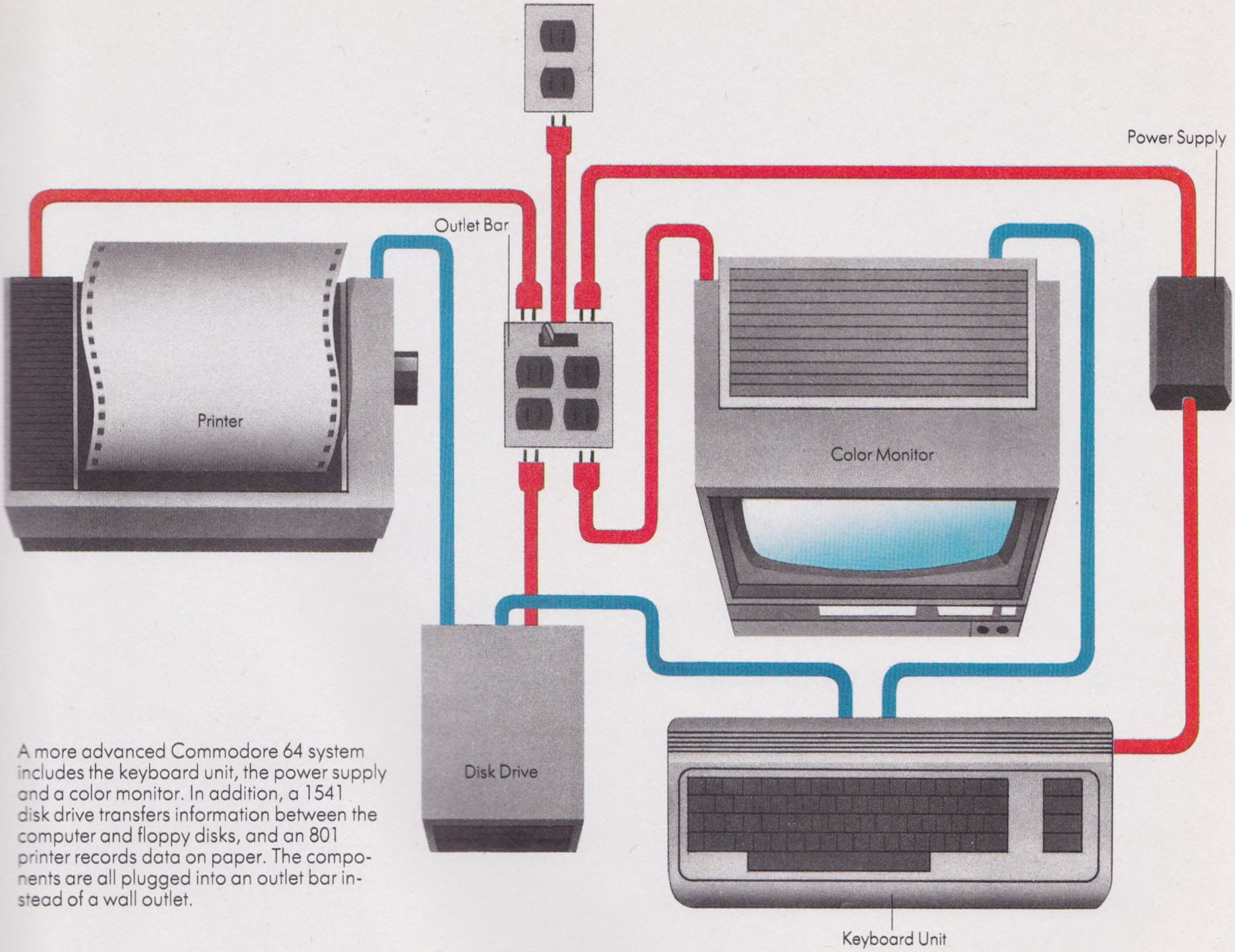
Linking the components of your 64 can be as simple as plugging in a lamp, if you first position the components for convenient use and take a few precautions. Before you connect any components, set their power switches to OFF.

Put the keyboard on a table and place the monitor stand and the monitor or TV set just behind it. For comfortable reading, the screen should be at least 14 inches from your eyes. If you are using a TV set, position the disk drive and disks or the Datasette and cassette tapes at least 12 inches away from it. It can generate signals that interfere with these devices. A printer can be put on a table or on a stand — anywhere up off the floor. Allow space for the paper, either behind the printer or on the floor.

Set the power supply to the 64 on a back corner of the table or on the floor near a wall, out of the way of accidental knocks or kicks.

Wherever you place the components, don't block the vents on any part of the system, and always keep coffee cups, snacks and ash trays away from all components. Spilled liquid, crumbs and tobacco smoke can get inside electronic devices, often with expensive results.

The data cables linking the components contain multiple wires that carry the electrical impulses used to encode information. A data cable may have as many as eight wires,



A more advanced Commodore 64 system includes the keyboard unit, the power supply and a color monitor. In addition, a 1541 disk drive transfers information between the computer and floppy disks, and an 801 printer records data on paper. The components are all plugged into an outlet bar instead of a wall outlet.

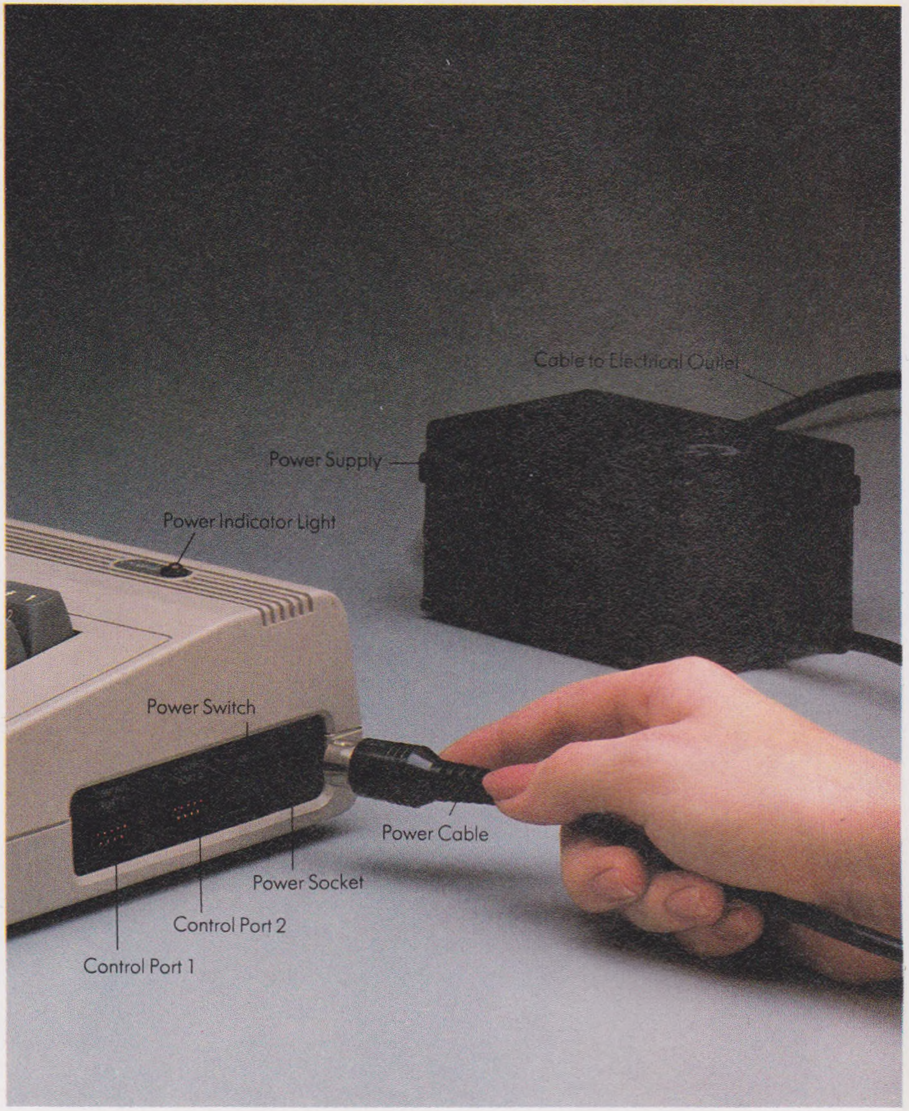
and connectors to match. The wires are protected against electrical interference, but they can be damaged if the cable is bent at an extreme angle or pinched. If a connector doesn't plug in easily, turn it over: All connectors have only one correct orientation. Look for the dimple on the connector — it should face up when the connector is inserted. If the connecter still doesn't fit, you may be trying it in the wrong socket.

The system's power cables for carrying household current have, in most cases, three wires and three-prong plugs. Don't get power from just any wall outlet: only grounded (three-hole) outlets protect the computer's circuits properly. Avoid jury-rigged arrangements with extension cords or three-way plugs.

It is preferable to plug the components into an outlet bar, rather than directly into a wall outlet. Such a bar has several outlets controlled by one switch. A circuit breaker in the outlet bar protects the system by shutting off power if there is a short circuit. Get a bar that also has built-in protection against sharp voltage surges. As an additional safeguard, unplug the system during electrical storms.

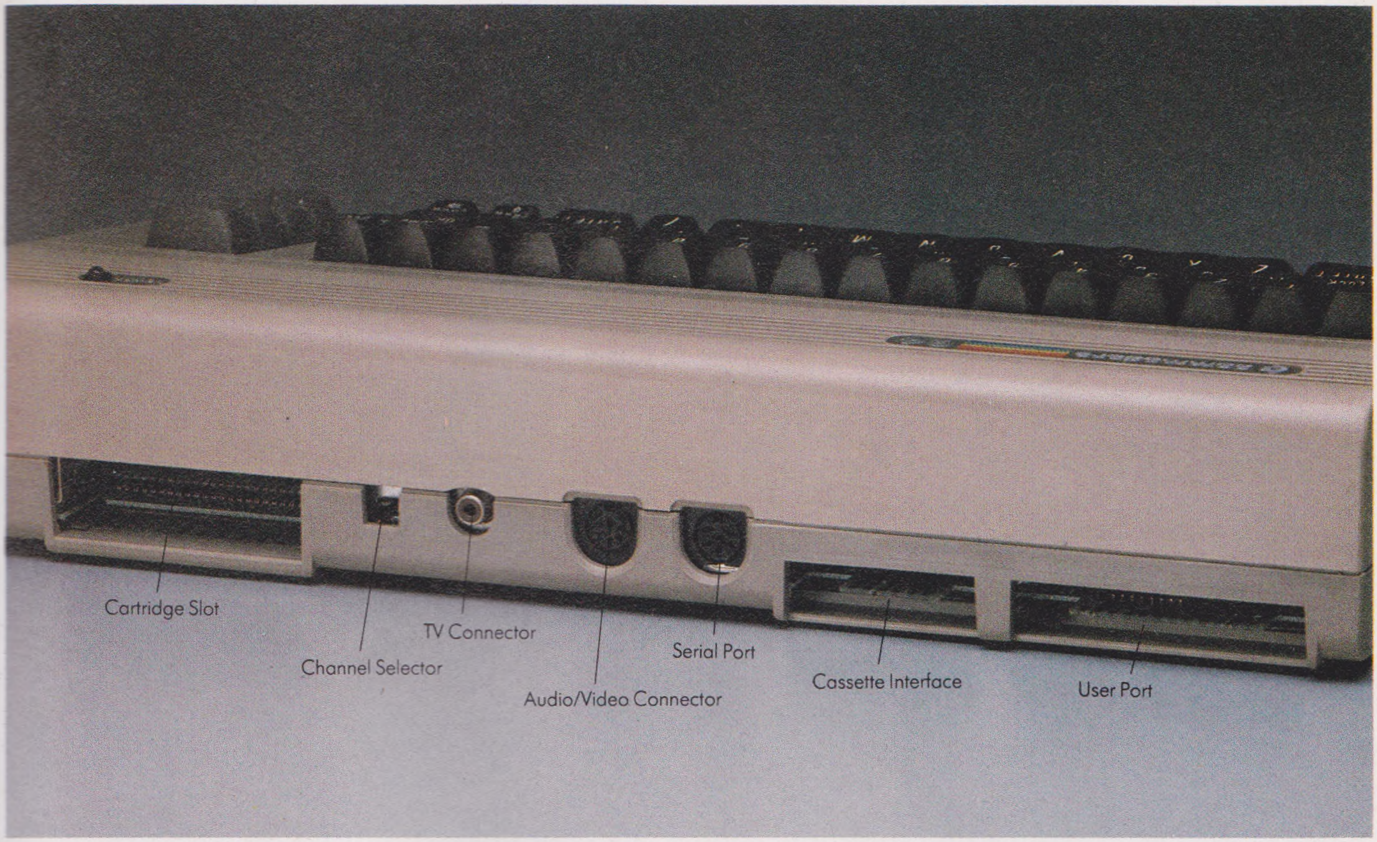
Avoid sharing circuits with large appliances such as refrigerators, which often cause peaks and dips in the house voltage when they switch on. The computer stops operating when voltage drops too far, and any information being processed is lost.

At the Heart of the System



The keyboard unit of the Commodore 64 serves two different functions. The keyboard itself is an input device — a means of getting information into the system. In addition, inside the unit is the control center of the Commodore 64, the circuitry that processes and stores the information introduced via the keyboard or other input devices.

The keyboard bears a comfortable resemblance to a typewriter. Most of the keys have letters and familiar symbols arranged in the same order as those of a typewriter. However, pressing a key on the Commodore 64 does not produce a letter on a piece of paper. Instead, the keystroke sends an electronic signal to the computer. Signals, or input, from the keyboard are handled by an integrated electronic circuit called the central processing unit (CPU), which executes your instructions, organizes bits of information and sends them along the proper channels. The com-



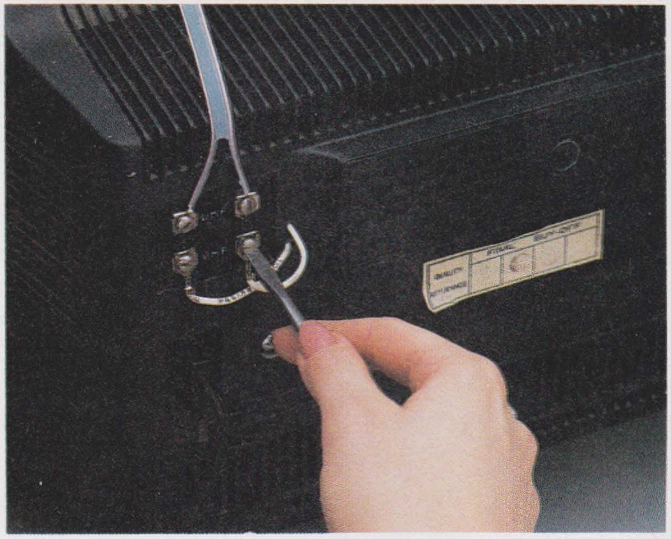
One socket on the keyboard unit is for connecting the power supply; the others are input and output ports, for connecting peripheral equipment. Fit the circular metal connector of the power-supply cord into the socket labeled POWER in the side of the keyboard unit *(left)*, with the connector's dimple upward. Do not plug the power supply into an outlet until the rest of the system is connected. The two control ports on the side of the unit are used to connect a variety of devices, including joysticks. On the unit's back panel are ports for a monitor, cartridges and other input and output devices *(above)*.

puter is said to read when it takes information in, and to write when it sends information out.

Much of the information passes to and from the computer's memory, which is housed in a set of specialized circuits. Part of the memory, called "ROM" for "read-only memory," holds a permanent record of information that the computer needs each time it is turned on. Another part, the read-and-write memory, lets the computer store and retrieve information. You can erase the contents of read-and-write memory to make room for new information. In fact, whenever the system unit is switched off, read-and-write memory is wiped clean.

All input coming into the Commodore 64 — from the keyboard, from a disk drive or from some other input device — is processed by the CPU before it is sent to memory or to an output device such as a monitor or a printer. There is no direct link between any of the Commodore 64's input and output devices: Whenever information is to be moved from one place to another within the system, it has to pass through the CPU before it can be sent along to the desired destination.

Connecting a Television for Computer Output

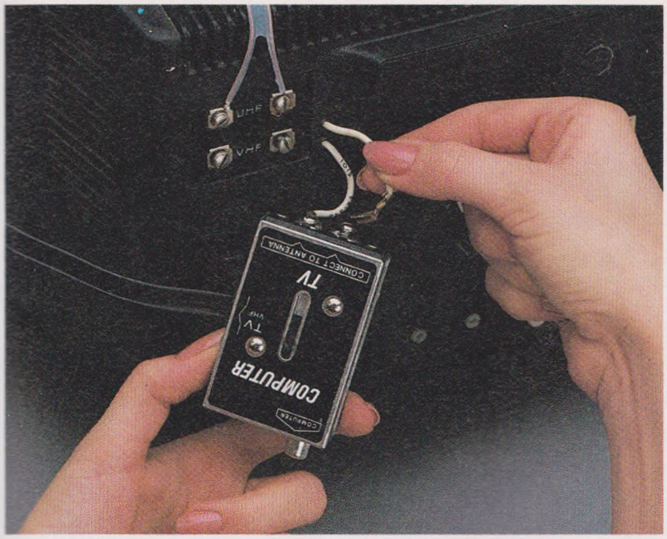


To attach the TV switchbox to the back of your television set, use a screwdriver to loosen the two screws that hold the VHF antenna wires in place *(above)*. Pull the U-shaped connectors at the ends of the antenna wires from under the screws.

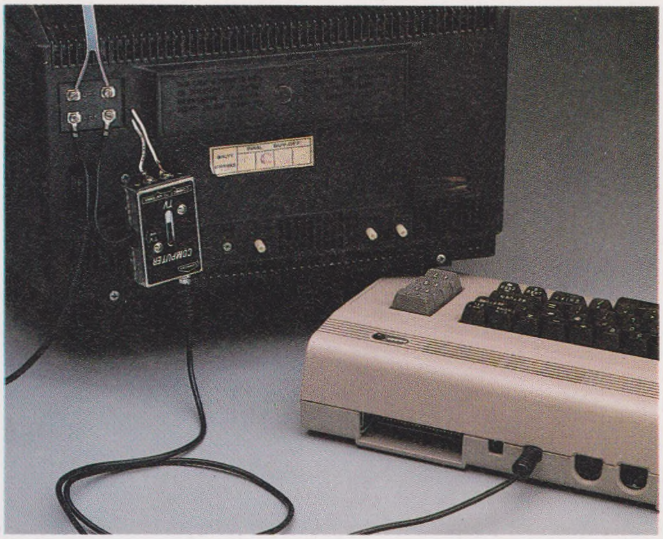
Built into the Commodore 64 keyboard unit is a TV modulator — a device that allows the computer to send video and audio signals to an ordinary television set.

The 64 can be connected to most TVs, either black and white or color, through the TV switchbox that comes with the keyboard unit. Once the hook-up is made, you need only flip the switch to the appropriate setting to use the set with the computer. The computer simulates a television station, broadcasting, through wires to the VHF antenna terminals of the TV set, on channel 3 or channel 4. Tune the TV set to whichever channel gives a clearer display; it will usually be the one not competing with a local station using the same channel.

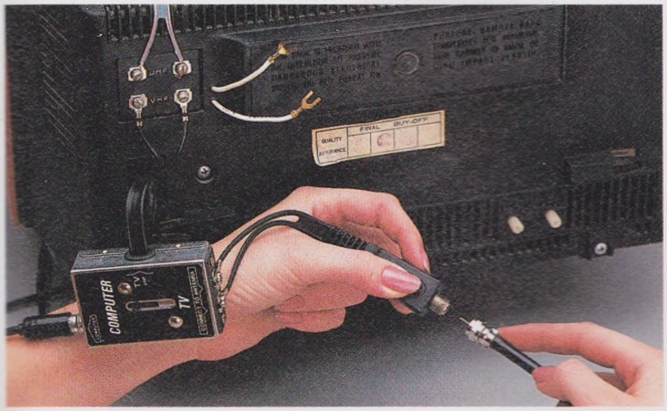
Impulses generated by the system components may interfere with reception on your TV set and cause a poor display. This problem is more severe with a color set than with a black and white one. Sometimes you can improve the display simply by shifting the components and cables in relation to one another. If the TV has automatic picture controls, you may find that it performs better when the controls are turned off. If the interference persists, you may need to substitute a better-quality cable for the equipment that comes with your Commodore 64 *(box, opposite)*.



Turn the TV/game switchbox upside down, with the two screws labeled CONNECT TO ANTENNA facing up. Slip the U-shaped connectors of the TV's VHF antenna wires into place under the switchbox screws *(above)* and tighten them securely.



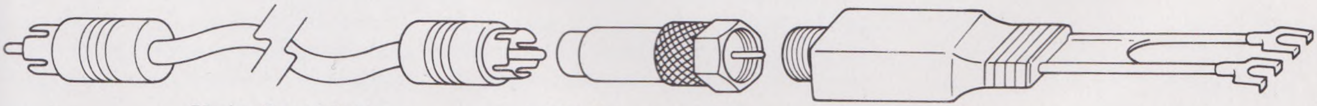
The antenna wires coming from the side of the switchbox and labeled TV VHF end in U-shaped connectors; slip these two connectors under the TV's VHF screws and tighten them. Plug one end of the coaxial, or shielded, cable supplied with the switchbox into the switchbox, and the other end into the TV connector on the keyboard unit *(above)*. Set the switchbox to COMPUTER or TV, to choose your use of the set.



A Cable-TV Adapter

If you have cable TV, you will need to buy a 75-300-ohm adapter to connect your set to the Commodore 64. Unplug the TV set from its cable and follow the procedure shown opposite to free the VHF antenna wires on the back of the set. Attach the antenna wires of the TV switchbox to the TV's VHF screws, then attach the two U-shaped connectors of the adapter to the screws labeled CONNECT TO ANTENNA on the TV switchbox. Plug the TV cable into the adapter's socket *(left)*.

An Extra Step to Beat Bad Reception

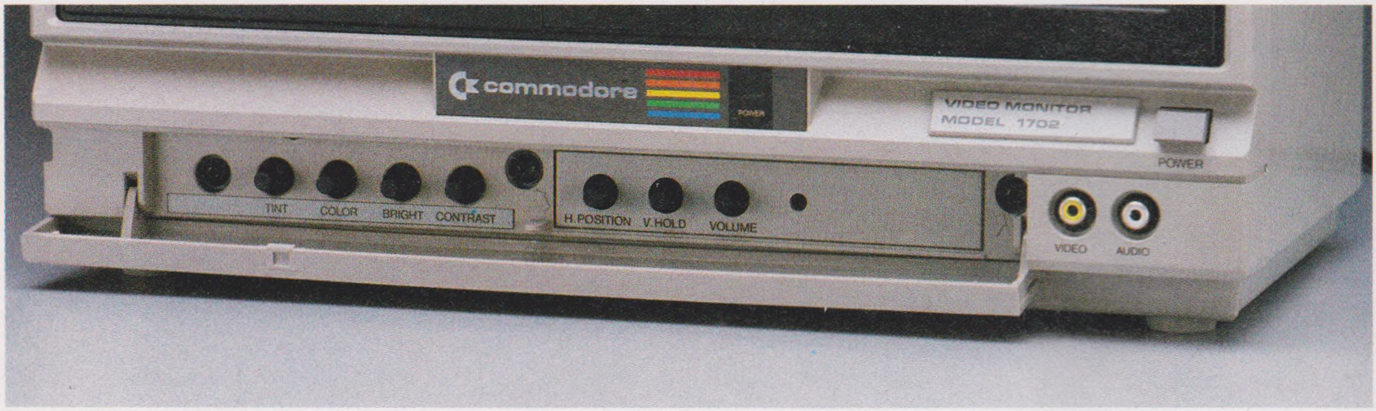


If simple measures fail to improve display on your TV screen, replacing the standard coaxial cable and switchbox with a different set-up may help. Use a heavy-duty 75-ohm coaxial cable with RCA phono plugs, an RCA phono-to-F connector and a 75-300-ohm adapter. Attach the adapter to the VHF screws of the TV and use the F connector to link the adapter and the coaxial cable. (If your TV is cable-ready, you do not need the adapter.) As long as this set-up is in place, the TV cannot receive regular broadcasts.

Using a Monitor for Clearer Display

The power switch of the 1702 monitor *(right)* is near the lower right-hand corner of the screen. A green power light indicates when the monitor is on. Below the power switch are audio and video inputs. They can be used if you have a monitor cable with two RCA phono plugs instead of three, but the screen image will be inferior to that of a hook-up made through the inputs on the rear of the monitor *(opposite)*.

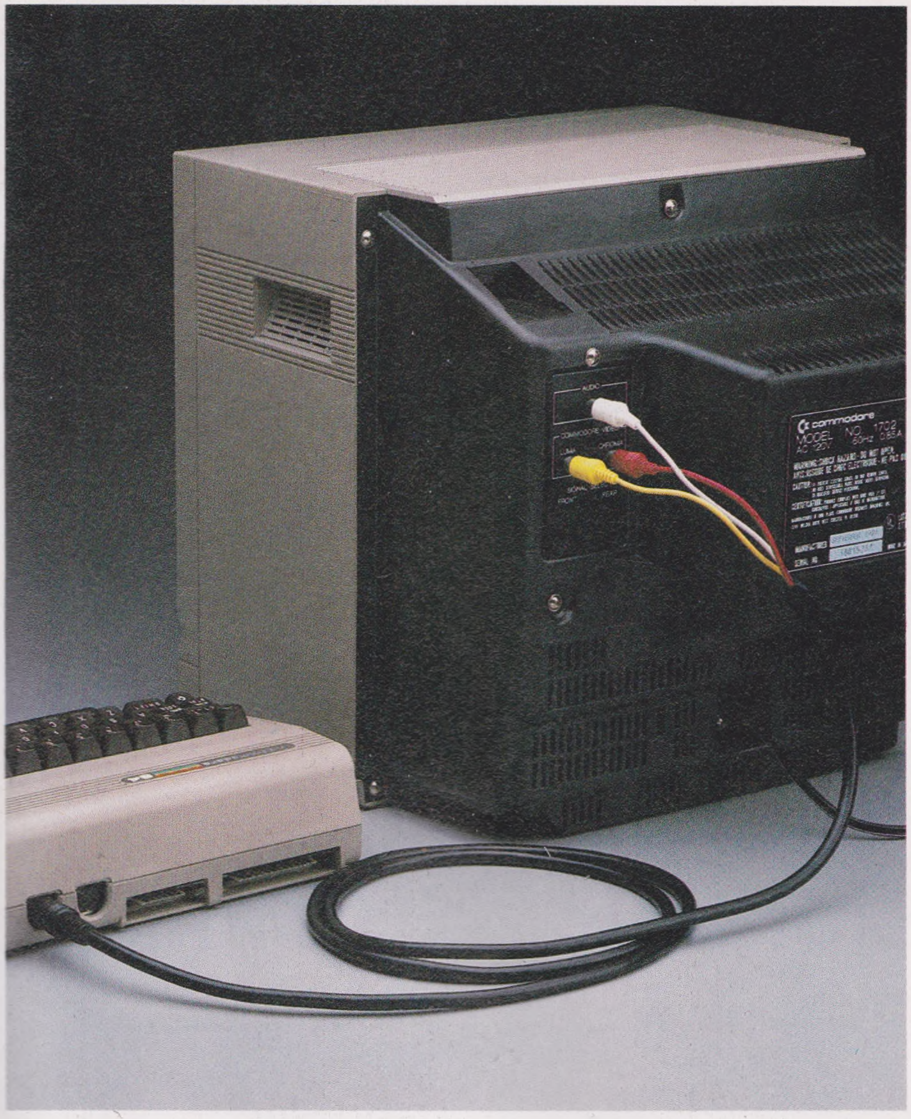
Behind the hinged panel at the bottom of the monitor are the volume control plus six knobs for controlling images on the screen *(below)*. Before using the tint, color, brightness or contrast knob, rotate each one until you feel it click into its detent position. Then adjust for brightness and contrast first, before changing tint or color. Use the vertical and horizontal controls to change the position of a screen image.



A monitor looks like a television set, but it produces a much clearer image of the Commodore 64's output. In exchange for this clarity, most monitors give up the tuner that would allow reception of regular TV broadcasts. (Combination TV/monitors are available.) The superiority of a monitor over a TV set lies in the fact that the signal supplied by the computer undergoes minimal translation when it is delivered to a monitor for display, and the computer has maximum control over the image displayed on the screen.

A monitor can display all of the numbers, letters, and graphic symbols that appear on the keyboard. Most monitors also have a built-in speaker and an audio connector, to take advantage of the sound-producing capacity of the 64. Unless sound is unimportant to your work with the 64, you will want to avoid the monitors that lack speakers.

Either a monochrome or a color



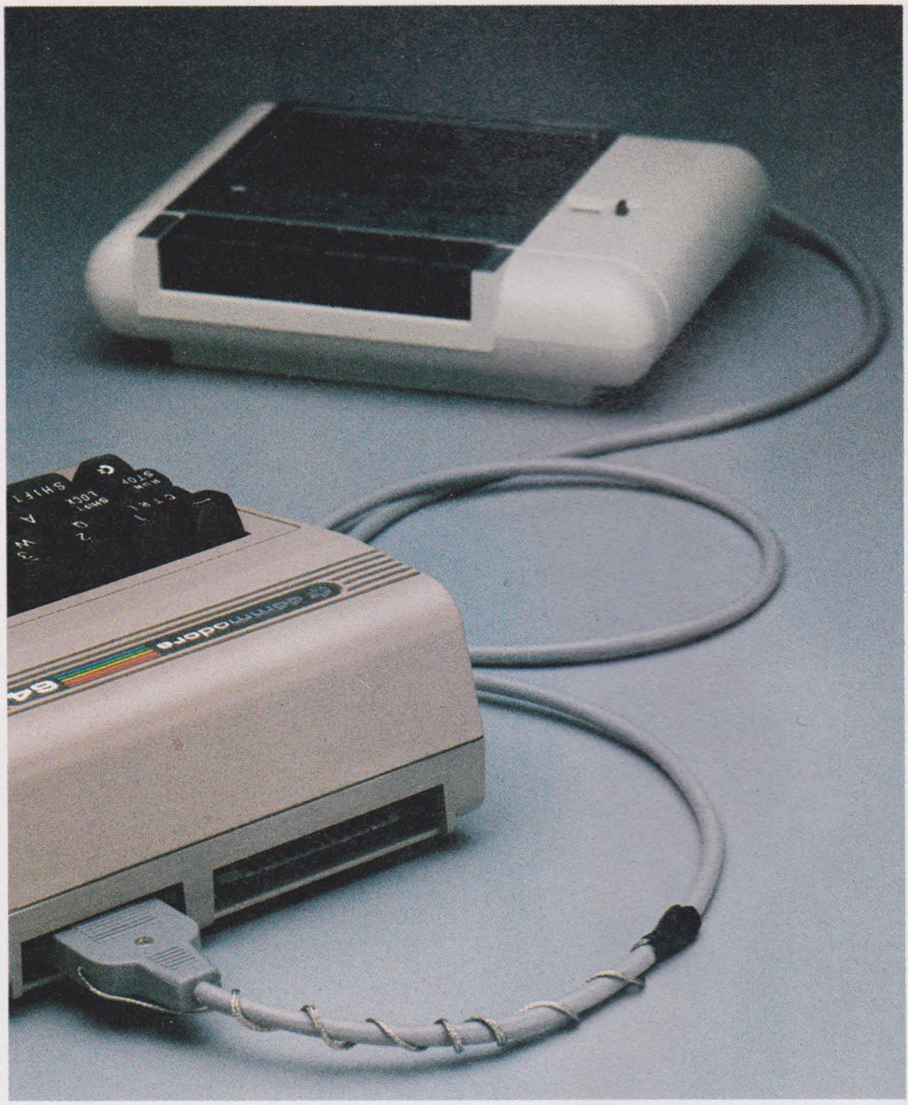
The eight-pin DIN connector, found at one end of the monitor cable packed with the 1702, plugs into the audio/video port on the back of the keyboard unit *(left)*. Insert the red, white and yellow RCA phono plugs found at the other end of the cable into the corresponding color-coded inputs on the back of the monitor. Set the signal-select switch to the REAR position.

monitor can be used with the Commodore. On a Commodore 1702 and other color monitors, the maximum number of easily legible columns, or characters per line, is 40. The 40-column display is standard and, for most uses, is sufficient. However, with certain applications such as word processing, an 80-column display is more convenient. For 80 columns, you need a monochrome monitor, plus an interface called an 80-column board.

On most monitors the hook-up for the monitor cable is on the front. The 1702 has audio and video inputs on the front and on the back as well. The back hook-up is superior: Two connectors, rather than a single one, are provided for the video signal. Separating the signal into two components, called chrominance and luminance, produces a better color image. (If you have a video cassette recorder, you can take advantage of the 1702's superior image. Plug the VCR into the two inputs on the front of the monitor.)

Place your monitor where glare on the screen will not be troublesome. If the room has an overhead light fixture, the best place to put the monitor is directly beneath the light. If you use a desk lamp, aim it at the work surface next to the computer and not at the screen.

A Tape Recorder for Storage and Playback



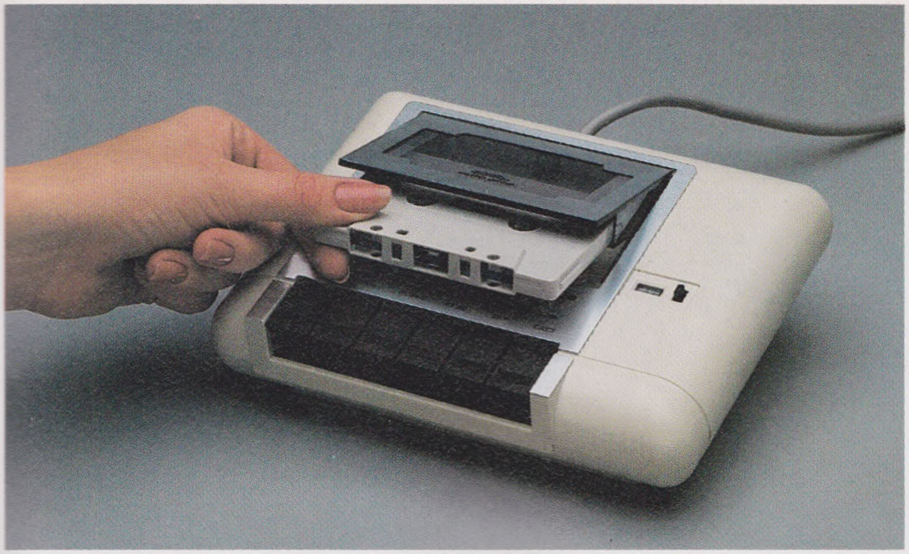
With the 64 turned off, push the Datasette's plug into the cassette port on the back of the keyboard unit *(above)*. Attached to the cord is a ground wire, which is required by some Commodore computers but not the 64. Wrap that wire around the Datasette's cord and secure it with electrical tape to keep its metal end from brushing against an electrical connection and possibly causing a short circuit.

With Commodore's digital cassette recorder, called a Datasette, you can load commercial programs that are recorded on tape into the 64. The Datasette also provides a reasonably accurate and reliable way to store instructions and information that the 64 holds in its memory. Because it is simple to operate and harder to damage than a disk drive, the Datasette is an appropriate piece of hardware for systems that children will be using.

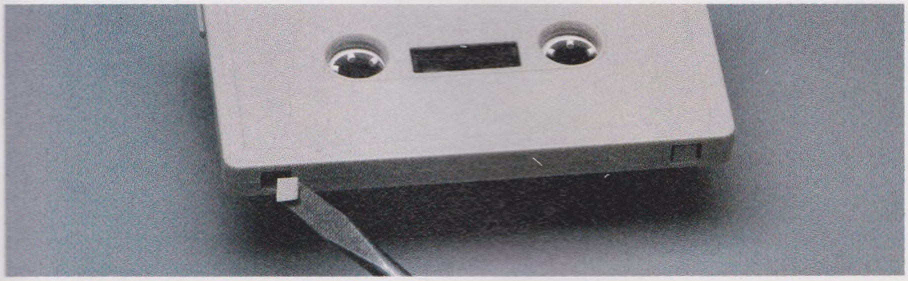
Standard blank audio cassettes can be used with a Datasette to store information. High-quality, short-playing ferrous oxide AGFA PE-11 tapes give the best results. It does not matter whether the tape has a leader, since the Datasette automatically waits a few seconds before storing information.

It is possible to substitute an audio recorder for a Datasette if you use a special cassette interface — a device that converts the audio recorder's signals to a form the computer can understand. However, an interface costs almost as much as a Datasette, so unless you already own a very high-quality audio recorder that you can devote to computer use, you are better off buying a Datasette.

The portion of the recorder that touches the cassette tape should be cleaned about once a month to remove the accumulated brown oxide. Soak a cotton swab in isopropyl alcohol or a commercial cleaner and



Slip the cassette into the brackets on the lower side of the Datasette's lid, with the exposed tape facing forward *(above)*. Shut the lid and press the button next to the tape counter at right to set it to 000. The counter keeps track of how much tape is read or written on. To remove the cassette, press the EJECT key.



To prevent accidental erasure or further recording on one side of a cassette, punch out the small tab in the write-protect notch for that side *(above)*. With the side you want to protect facing up and the notches facing you, punch out the tab on the left. To record on that side later, cover the notch with tape.

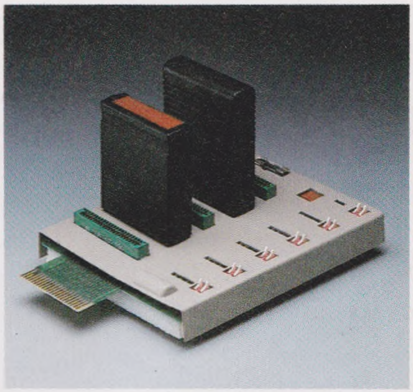
rotate it firmly against the tape heads, the idler wheel and the capstan that moves the tape. If your Datasette fails to load commercial programs that other Datasettes can handle without problems, it may need realignment — an adjustment that a dealer can make. After realignment, you may not be able to load the programs you saved while the Datasette was misaligned.

Care of cassettes consists mainly of protecting them from heat, dirt and magnetism. Store cassettes in a case in a cool location that is at least 12 inches away from the 64's power supply, a monitor, TV set, or a telephone. All of these devices emit electromagnetic signals that can destroy taped data. In addition, avoid touching the exposed portion of the tape.

Using Cartridges

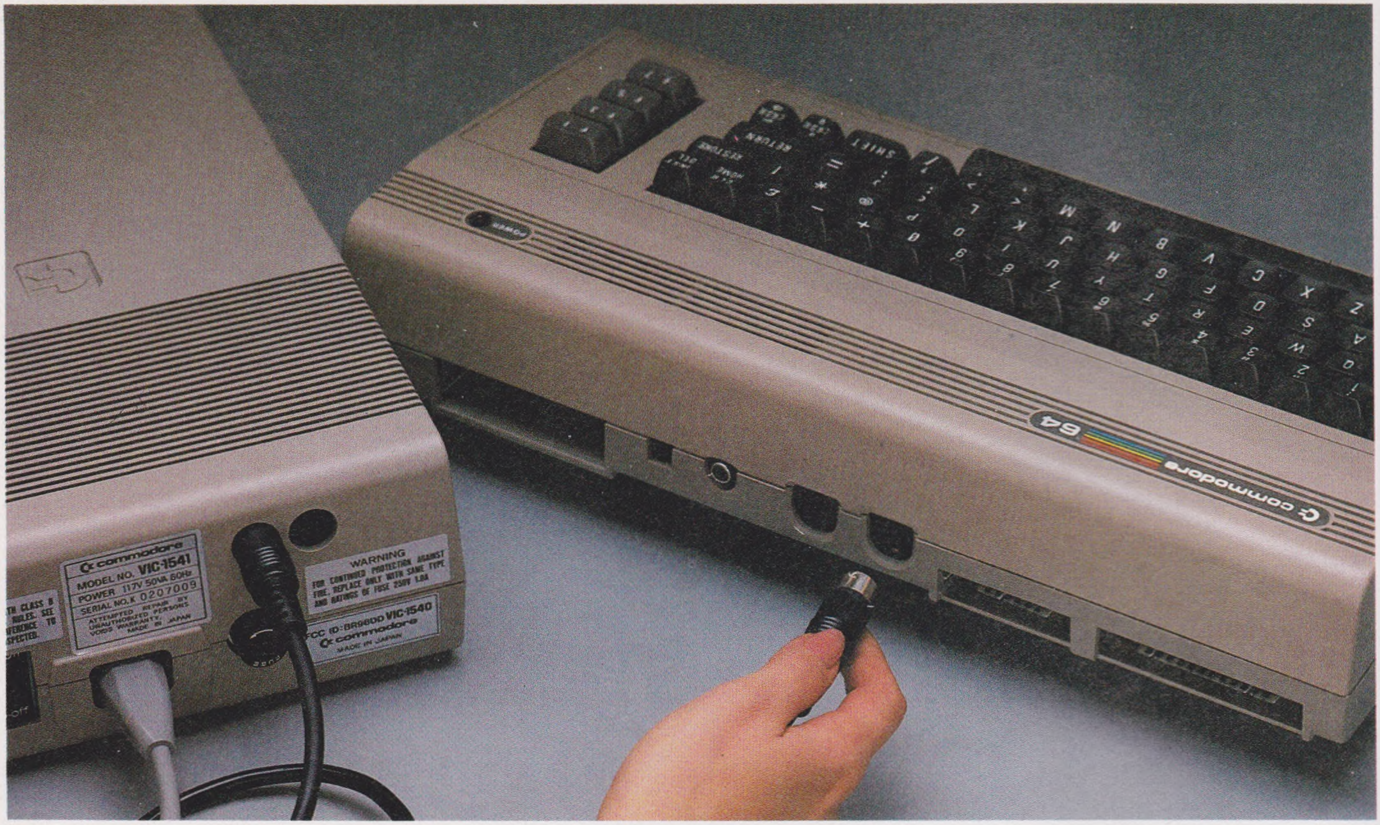
Cartridges provide a very fast, easy way to load programs into the 64's memory and are an excellent supplement to a cassette recorder or a disk drive. Essentially, a cartridge consists of a small plastic box containing one or more ROM microchips on which a program is inscribed. Many popular kinds of programs come in cartridge form, including games, word processors, spreadsheets and educational programs.

Before plugging in a cartridge, be sure that the 64 is turned off — otherwise, both the cartridge and the computer may be damaged. Slide the cartridge firmly into the cartridge slot on the back of the keyboard unit so that the contacts mesh. The program is instantly loaded when you turn the computer on, and in most cases it will begin to run automatically.



Often-used cartridges can be kept plugged into a five-slot motherboard *(above)*, which in turn is plugged into the 64's cartridge slot. When you flip the switch next to a cartridge, the program is instantly loaded into computer memory.

A Disk Drive System for Rapid Service



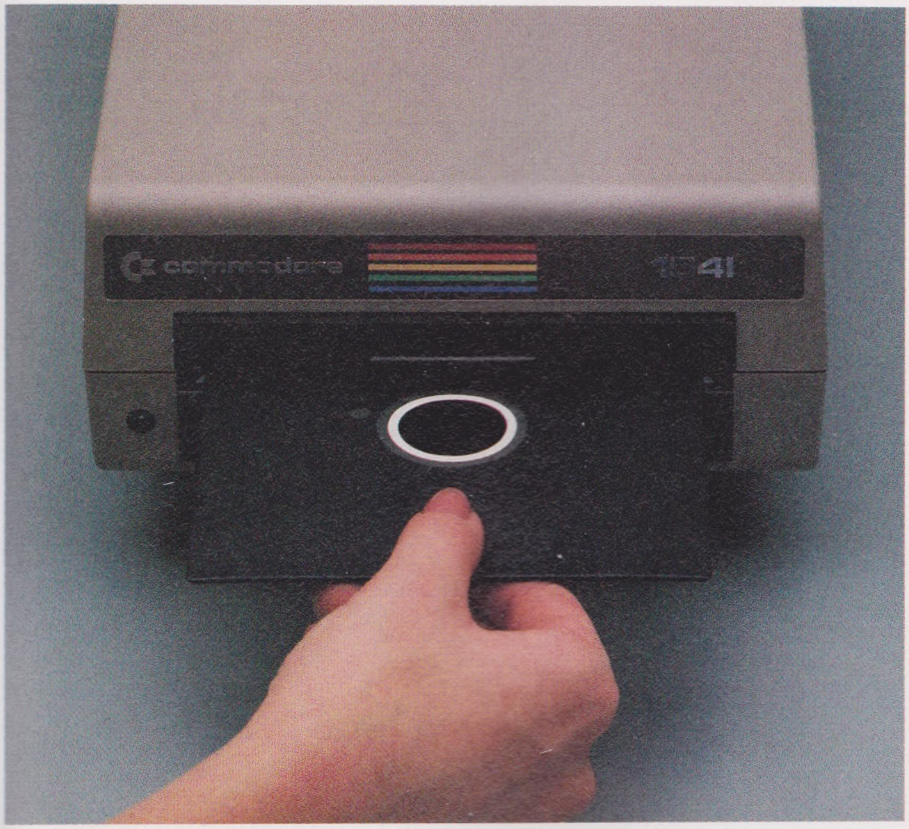
Plug one of the six-pin DIN connectors of the data cable supplied with the disk drive into either of the round serial ports on the back of the drive. Plug the cable's other connector into the keyboard unit's serial port *(above)*. The power cord is plugged into the rectangular outlet on the back of the drive.

With a disk drive connected to the keyboard unit, the Commodore 64 can read or record information on plastic disks *(box, opposite)*. Many people prefer to use a disk drive system rather than a cassette recorder because of its greater convenience and its greater speed of operation.

The way a disk drive works is roughly analagous to the working of a record player, whose needle can be moved about freely to pick up signals from anywhere on the surface of a phonograph record. The disk drive can hop directly from one block of information to the next one you need, instead of having to read through all the intervening information as a cassette recorder must.

The 1541 disk drive is shipped with a protective cardboard insert. To remove it, first peel away any tape from the horizontal opening of the disk drive. Put your finger on the latch lever that covers the slot and push the lever in until it springs upward to open. Pull the cardboard straight out of the slot.

The 1541 disk drive also comes with a demo/test disk that contains instructions for special diagnostic tests. Whenever you set up the computer or add a component to it or move it, you should run the performance test program on the demo disk to make sure that all components are functioning correctly. (See Chapter 3 for instructions on how to load the program.)



Insert a disk into the drive so that the jacket's oval opening, or head slot, goes in first and the rectangular write-protect notch is on the left *(above)*. Slide the disk in until it is firmly seated, then push the door down to latch it. Be careful not to touch any parts of the disk that are exposed through openings in the jacket. To remove a disk, push in on the door.

The diagnostic tests take a little time, so be patient. If a test reports a malfunction, first check all the connections and power switches, then run the diagnostic routine again. If you still get the same result, call your dealer for service under your warranty, or turn to the troubleshooting procedures in your *Commodore 64 User's Guide*. When the diagnostic routines are successfully completed, the computer is ready to work.

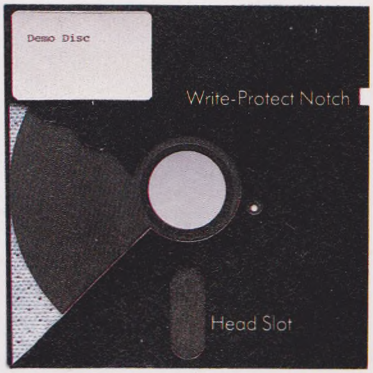
Unless you regularly smoke near your disk drive, or ever spill a liquid on it, or mistakenly record information on both sides of a single-sided disk, cleaning will rarely, if ever, be necessary. If you must clean your disk drive, use a commercial kit designed for this purpose.

Floppy Disks

The Commodore 64 stores information on disks of flexible plastic, known as floppy disks to distinguish them from the rigid ''hard disks'' used in some computers. Coated with metal oxide that can be magnetized to hold information, each disk has a protective plastic jacket. The disk drive holds the jacket still and spins the disk, reading information on the disk surface through an opening in the jacket called a head slot.

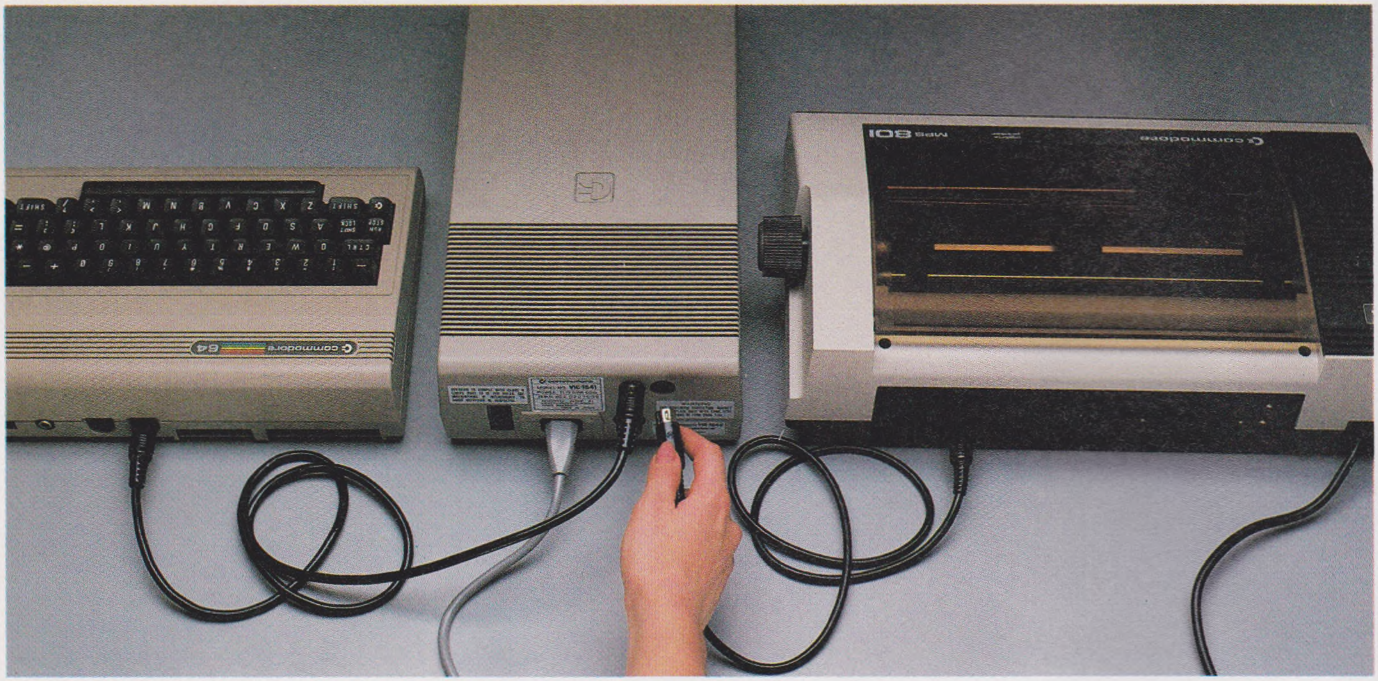
Handle a disk only by its jacket, and store it in its paper sleeve. Dust or scratches can alter or erase information on the disk.

Store disks away from heat and from equipment that may generate magnetism, such as the power supply, the monitor or TV, telephones and tape recorders. To prevent accidental erasure, cover the write-protect notch *(below)* with a gummed tab provided by the maker. When the notch is covered, the disk can only be read.



The protective jacket of this floppy disk is cut away to show the 5¼-inch-wide plastic disk inside. The head slot gives the disk drive access to the disk surface to record and read information.

A Printer for Permanent Copies

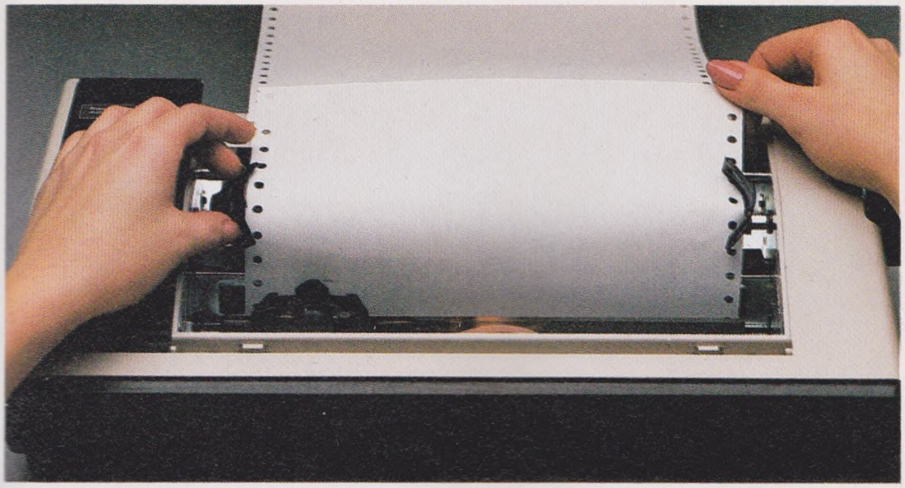


In the set-up shown here, the six-pin DIN connector of the 801 printer *(far right)* is plugged into one of the serial ports on the back of a 1541 disk drive. The disk drive, in turn, is connected to the keyboard unit. If you are not using a disk drive, plug the printer cable into the serial port on the back of the keyboard unit.

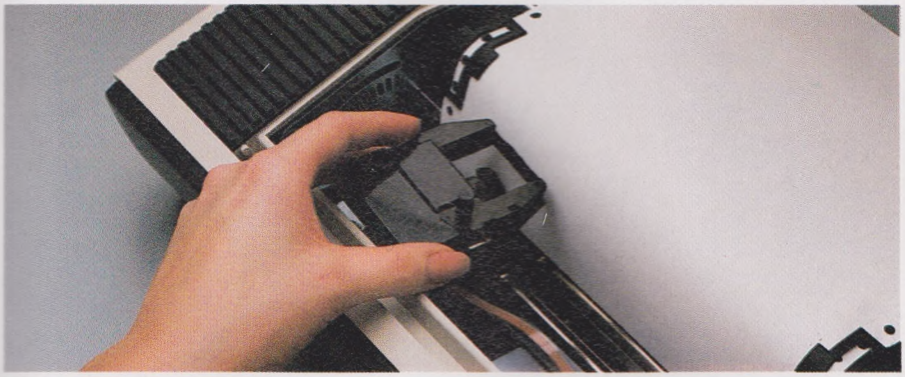
A printer is like a typewriter without a keyboard. It generates permanent images on paper, called hard copy, when told to do so by the computer. Printers come in two primary varieties, dot matrix and letter quality *(box, opposite)*.

The Commodore 801 is a dot matrix printer; that is, the characters it prints are formed out of dozens of tiny dots. The printing mechanism, called the print head, contains tiny wires and one or more electromagnets. When the electromagnets are energized by an electric current, they propel the wires against the printer ribbon to make an image on the paper. The wires then spring back, ready to make another character.

Printer paper comes in long, continuous sheets that you later separate into page-sized sections by tearing along perforated lines. The 801 printer can accommodate paper of various widths, from 4½ inches to 10 inches, and it can also produce a



A tractor feed mechanism at each end of the printer's rotating bar, or platen, keeps the paper from skewing to the side. When you insert paper, fit the perforated holes along both edges onto the pins of the tractor feed and snap down the latches at each end, to hold the paper firmly against the platen *(above)*.



To change the ribbon of the 801, slip the ribbon cassette *(above)* out of the two plastic brackets holding it in place. Turn the ribbon-advance knob of the new cassette to take up any slack, slide the exposed ribbon between the platen and the print head, and snap the cassette into place between the brackets.

carbon copy along with the original.

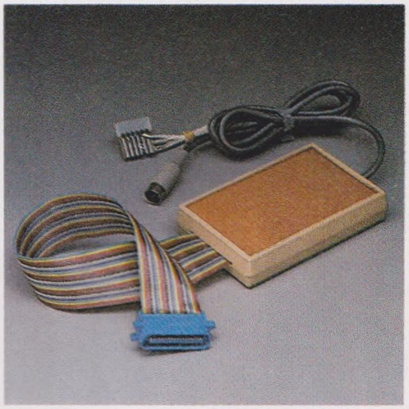
A major advantage of the dot matrix method of printing is the large number and variety of characters that can be produced. Most printers are capable of making any character, including arithmetic symbols, foreign alphabets and special typefaces. Some dot matrix printers can also produce the graphic symbols displayed on the fronts of a number of the 64's keys. Most dot matrix printers, including the Commodore 801, have a special setting called graphic mode, in which each dot can be controlled individually, allowing the user to create highly detailed charts and pictures.

High speed is another advantage of dot matrix printers. Even the least expensive dot matrix printers can produce 30 characters per second; the best of them print at speeds of 200 characters per second.

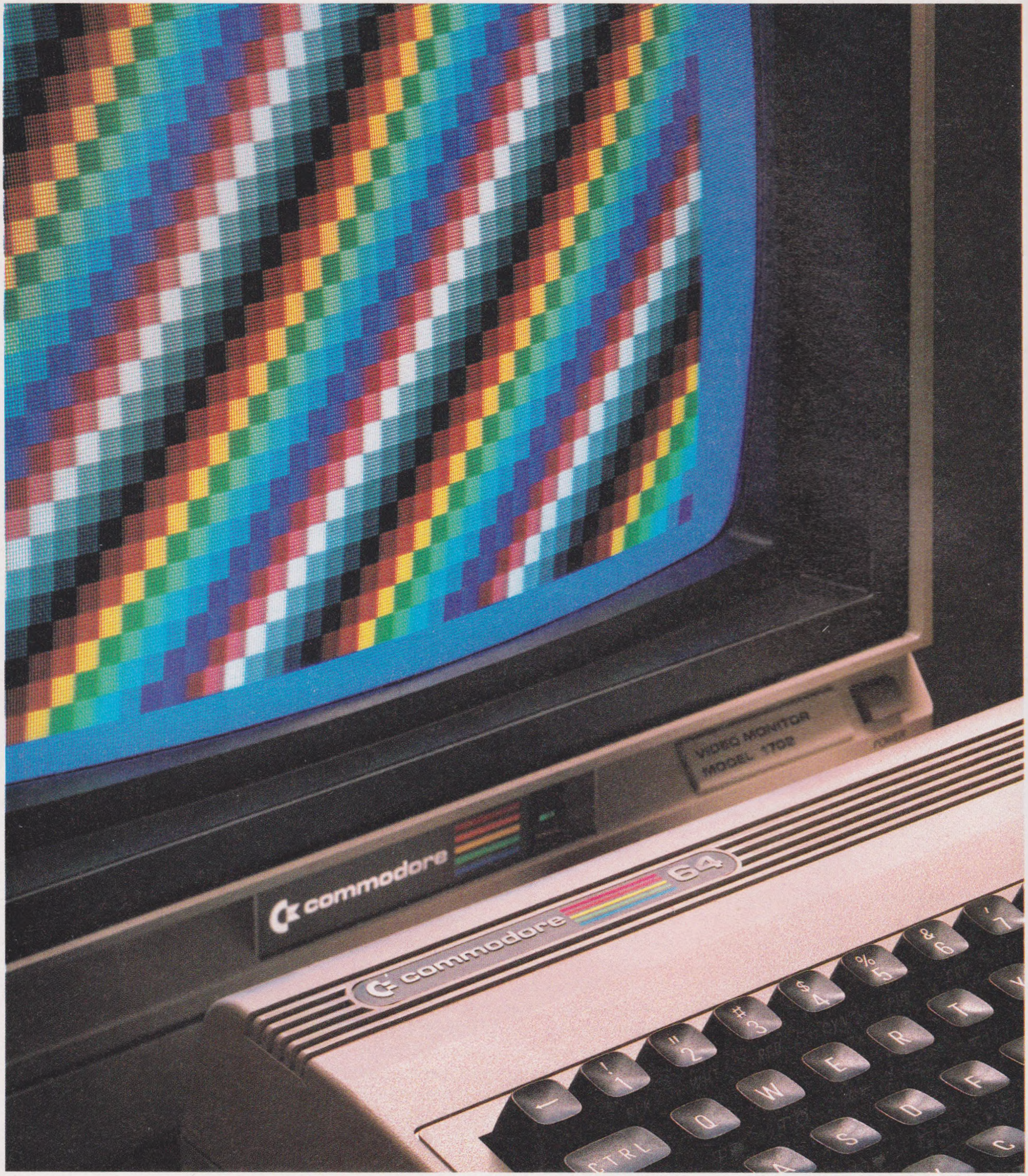
Interfaces for Other Printers

The 801 is not the only dot matrix printer that can be used with the Commodore 64. If you are interested in another brand, be sure that it can print the 64's graphic symbols. For type of higher quality than a dot matrix printer provides, you can get a daisy-wheel printer, named for the shape of its print head. It prints with fully formed characters instead of wires.

If you choose a printer other than the 801, it will probably need an interface — a piece of equipment used to link components that would otherwise be incompatible. Printers receiving their signals through a port called a Centronics parallel require a parallel interface, which ordinarily plugs into the cassette port of the 64. Another common type of interface is the RS-232, which plugs into the 64's user port. If you buy the Commodore 6400 daisy-wheel printer, you will need to get an IEEE-488 interface.



The blue connector of the Centronics parallel interface *(above)* plugs into a printer's parallel port, the white one into the 64's cassette port, and the DIN connector into a disk drive's serial port.


VIDEO MONITOR
MODEL 1702
commodore
commodore
64

Getting Started: Discovering the Power of the Keys

The keyboard of the Commodore 64 is a marvel of flexibility. It is the primary input device of the system and also controls the activity of all the peripheral components.

Your first step toward mastering the 64 system and getting it to work for you is to become thoroughly acquainted with the keyboard. It provides all of the alphabetic, numeric and punctuation keys that are found on a standard typewriter. These keys can be used to print characters on the display screen, but some of them have totally new functions.

Certain punctuation marks, for instance, have been assigned new uses in BASIC, the computer language the 64 is designed to understand. A single punctuation mark can, in the proper context, succinctly deliver a complex command to the computer. Other keys are analagous to the margin release or the backspace of a typewriter: They never print characters but instead affect the way in which the machine operates. These keys let you move text around on the display screen or erase it, instruct the computer to store information in its memory, or call into play a dazzling array of colors and graphic symbols.

The color and graphic possibilities of the Commodore are among its most desirable features. It can generate 16 different colors, and it has 62 graphic symbols, which are shown in pairs on the fronts of keys. Among them are arcs, straight lines, bars both horizontal and vertical, grids, and even the symbols for playing-card suits. The symbols can be combined in a limitless number of ways to create larger figures, with arcs becoming circles and diagonal lines herringbone patterns.

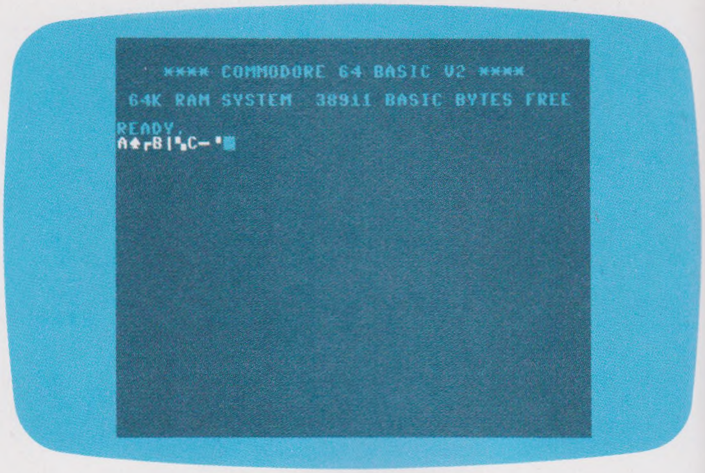
Interweaving color changes with the graphic symbols makes the Commodore an immensely rich graphic medium. A child can use it to make drawings, while a business person can illuminate data with charts and graphs. These possibilities can be brought into play by the keyboard alone: You don't need to know how to program before you start putting the Commodore through its paces with graphics and color.

Besides helping you become familiar with the keyboard, this chapter shows you how to use a command — **PRINT** — to make the computer perform arithmetic calculations for you. In addition, it outlines a simple program, or set of instructions, and describes the functions that the program's different elements perform.

Stair-stepped rectangles in all 16 colors of the Commodore palette fill a monitor screen. A display like this one is easy for a beginning user to accomplish, since nothing more than a simple sequence of keystrokes is required to select and position the colors.

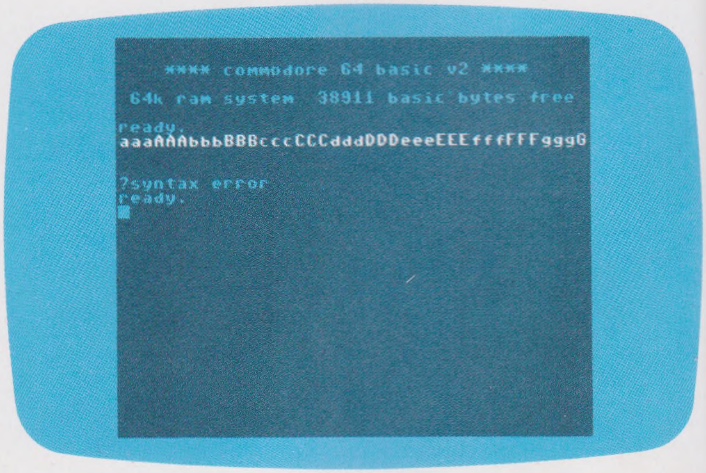
Warming Up with Keys and Screen

Turn on the system's components by switching on first the outlet bar *(page 11)* and then the individual components, leaving the system unit for last. **READY** will appear on the screen *(right)*. The computer comes on in graphic mode, with all letters in upper case and many of the keys able to generate graphic elements. For example, if you hold down **SHIFT** and press *A*, the graphic symbol on the right side of the *A* key front will appear on the screen. The **COMMODORE** (C=) key plus the *A* key will give you the graphic symbol on the left front of the *A* key.



```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM  38911 BASIC BYTES FREE

READY.
A♠┌B|▚C–'█
```

The sign-on message is converted from graphic mode to text mode when you hold down the **SHIFT** key and press the **COMMODORE** key: All letters instantly change from upper to lower case *(right)*. To produce upper-case letters in text mode, hold down the **SHIFT** key and press the letter keys. Pressing the **RETURN** key at the end of a line of letters, as shown at right, usually produces the message **?SYNTAX ERROR**, which means that the computer cannot process what you have typed. If you want to get back to graphic mode, hold down the **SHIFT** key and press the **COMMODORE** key.



```
    **** commodore 64 basic v2 ****

 64k ram system  38911 basic bytes free

ready.
aaaAAAbbbBBBcccCCCdddDDDeeeEEEfffFFFgggG

?syntax error
ready.
█
```

Each time you turn on the 64, it performs a quick wake-up routine. While the computer checks the functioning of its memory, the red lights on the front of the disk drive and the printer flash. The display screen flashes, then darkens for a second before a sign-on message appears:

******COMMODORE 64 BASIC V2******

64K RAM SYSTEM 38911 BASIC BYTES FREE

READY.

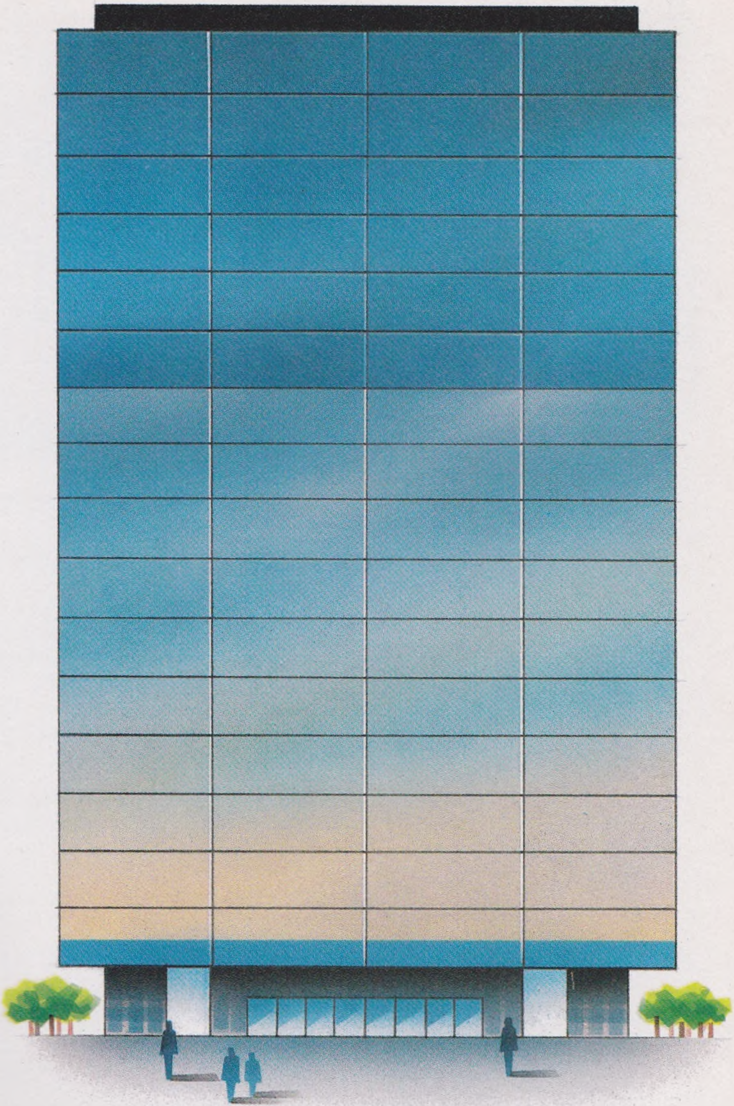
This message communicates several important facts. The first line announces that the 64 is using BASIC V2, a version of the BASIC programming language developed for Commodore machines. (BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code.)

The second line informs you that the 64 has about 64,000 bytes of read-and-write memory. (A byte is the unit of memory that can hold one character — number, letter or symbol.) Of those memory locations, 38,911 are available for BASIC programs and for data you put into the computer; the others already contain instructions the computer uses to control its own elementary functions or to process input in BASIC.

READY — the final word of the message — means that the computer is ready to work in BASIC.

On the next line is a flashing square called a cursor (from the Latin word for "run," because it moves about on the screen). When the com-

Workspaces for Memory



In the picture at right, the 64,000-byte memory of the Commodore 64 is likened to a large office building in which different areas are devoted to specific functions. Memory is divided into major blocks, called workspaces. The darkened workspaces on the upper floors (accounting for 24,000 bytes of memory) and a suite on the mezzanine (another 2,000 bytes) are given over to instructions built into the computer. The remaining workspaces, covering 38,911 bytes, are vacant and can be filled by the user with instructions and data.

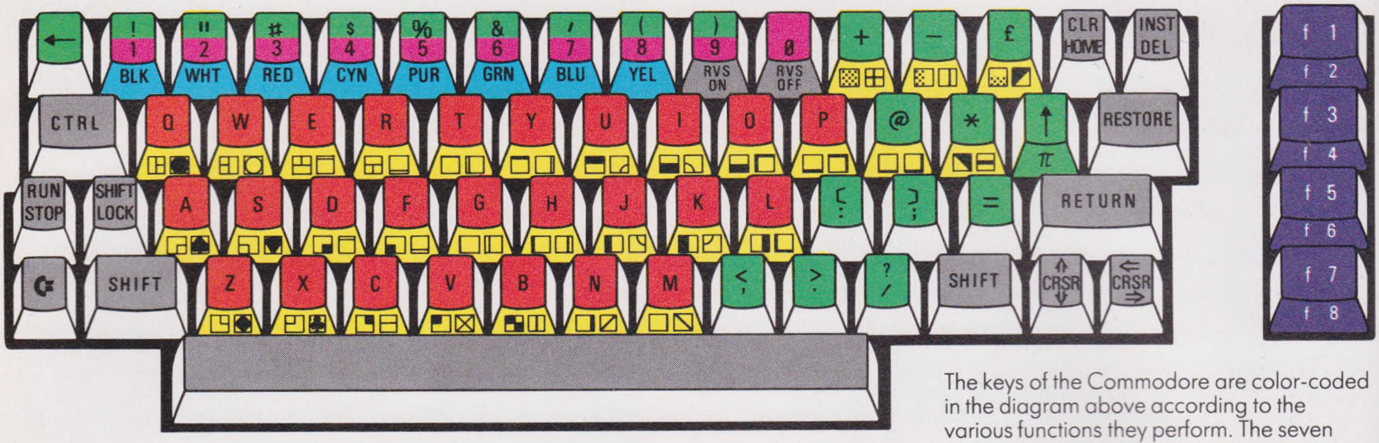
puter is waiting for you to do something, the cursor flashes. It also acts as a marker, showing you where on the screen the next character will appear. When you press a character key, the cursor moves one column, or position, to the right. At the end of a line, or row, the cursor moves down to the first column of the next row. If you are in mid-word when the cursor reaches the end of a row, simply keep typing: The word will automatically be continued on the next row.

Do not press **RETURN** at the end of a row, as you would on a typewriter, or the 64 will probably send the message **?SYNTAX ERROR.** A press of the **RETURN** key is a signal to the computer to process the line just typed in, and when that line is not in BASIC, the 64 cannot interpret it.

The screen can hold a total of 25 rows. When the last one is filled, a process called scrolling takes place: Everything on the screen moves up one row, with the top row disappearing from view and a blank row appearing at the bottom of the display.

NOTE: On a color TV or color monitor, all characters — both those you type in and those that make up messages the computer generates — appear in light blue. In the sample screens in this book, only computer messages are printed in light blue. All information you type in is printed in white, so that you can tell at a glance the origin of a particular group of characters.

An Introduction to the Keyboard



The keys of the Commodore are color-coded in the diagram above according to the various functions they perform. The seven categories into which the keys are divided are listed at left, with the color code and a short description of each category.

ALPHABETIC	🟥	Arranged in the same order as on the keyboard of a typewriter.
NUMERIC	🟪	Arranged in the same order as on the keyboard of a typewriter. Letters *O* and *L* cannot be substituted for numerals *0* and *1*.
SYMBOL	🟩	Punctuation marks; mathematical and business symbols; used in commands.
COLOR	🟦	Abbreviated on the faces of number keys. Used with the **CONTROL** and **COMMODORE** keys to change cursor color.
GRAPHIC	🟨	Characters shown on faces of alphabetic and five symbol keys. Produced in graphic mode, with **SHIFT** key or **COMMODORE** key down.
SPECIAL	⬜	Direct the computer to take specific actions, described in chart on opposite page.
PROGRAMMABLE FUNCTION	🟦	Keys that do nothing until they are assigned specific chores by a program.

Unless it is informed otherwise, the 64 expects to receive all of its instructions via the keyboard. The layout of its numeric and alphabetic keys is identical to that of a standard QWERTY typewriter — so named for the top row of alphabetic keys. Other characters, such as punctuation marks and mathematical signs, do not appear in the same positions on every typewriter.

With a few exceptions, each key of the 64 has at least two functions; some have three. Keys bearing numbers, for instance, also let you alter the color of a character when you press them along with the **CONTROL** or **COMMODORE** keys.

Besides the letters on their tops, alphabetic keys bear graphic designs on their fronts. These characters appear on the screen when the keys are pressed in tandem with the **SHIFT** key or the **COMMODORE** key. These two keys, like several others, do not themselves produce any characters on the screen; their function is to alter the operation of other keys. Used alone or in combination, the 66 keys of the Commodore keyboard can send more than 200 different signals to the computer.

The computer reads keyboard input literally and accepts no substitution of similar characters. If you type the letter *O* for the numeral *0* as part of a mathematical operation, the computer may not be able to process the operation correctly.

The Functions of Special Keys

The chart below summarizes the principal uses of the special keys. The first five control the movement of the cursor; the remainder have diverse functions.

NAME OF KEY	KEY	FUNCTION
CLEAR/HOME	CLR HOME	Press key alone to return cursor to home position in top left corner of screen. Press **SHIFT** plus key to return cursor to home and erase all text on screen.
CURSOR UP/DOWN	⇧ CRSR ⇩	Press key and release to move cursor down to next row. Press **SHIFT** plus key and release to move it up one row. Hold key down for continuous cursor movement.
CURSOR LEFT/RIGHT	⇦ CRSR ⇨	Press key and release to move cursor one column to right. Press **SHIFT** plus key to move cursor one column to left. If key is held down, cursor keeps moving.
INSERT/DELETE	INST DEL	Press to move cursor one column to left, deleting character there. Press **SHIFT** plus key to move all text from cursor onward one column to right, making space for insertion.
SPACE BAR		Press and release to move cursor one space to right and erase character under cursor's original position. Cursor keeps moving if space bar is held down.
CONTROL	CTRL	Used with many other keys. When pressed with a number key, changes the cursor color to the one indicated on the key front.
COMMODORE	C=	Used with many other keys. With alphabetic or punctuation key, produces the graphic symbol on the left side of the key face. With **SHIFT,** converts computer from graphic to text mode.
SHIFT and SHIFT LOCK	SHIFT SHIFT LOCK	Press for upper symbols on tops of punctuation and numeric keys, for right-hand graphic symbols on key fronts, for pi, for uppercase letters in text mode. **SHIFT LOCK** keeps shift function engaged.
RETURN	RETURN	Instructs computer to process the line of information just typed. With **SHIFT,** moves cursor to first column of the next line without processing the instruction.
RUN/STOP	RUN STOP	Stops a program in progress. To start the program again, type **CONT** and press **RETURN.** With **SHIFT,** automatically runs program from Datasette.
RESTORE	RESTORE	Press **RESTORE** while holding down **RUN/STOP** to clear the screen. Use this combination if **RUN/STOP** alone fails to stop the program.
REVERSE ON and REVERSE OFF	RVS ON RVS OFF	Press **CONTROL** and **REVERSE ON** to reverse the colors of characters and background. Press **CONTROL** and **REVERSE OFF** to restore original colors.

Symbol Keys with Distinctive Meanings

MATHEMATICAL SYMBOLS	
*	Multiplication
/	Division
+	Addition
−	Subtraction
↑	Exponentiation
(8) 9	Sets apart a group of related numbers and variables

COMMAND SYMBOLS	
" 2	A pair is placed around numbers or other characters that are to remain unchanged.
[:	Separates multiple commands on the same line.
] ;	Instructs computer to print items one after another across the screen instead of on different lines.
< ,	Separates elements within a command; advances the cursor to a predetermined column.
? /	Abbreviation for **PRINT** command in BASIC. In commands to disk drive, can stand in for missing characters.
$ 4	Identifies a string, or collection, of related numbers or other characters. In a command to disk drive, stands for the directory, which lists programs on a disk.

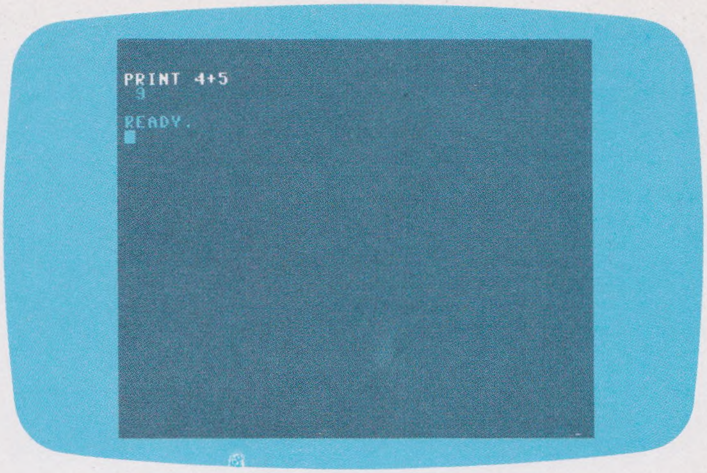
When the 64 is operating in BASIC, you must be careful with the symbols on the keyboard; some are assigned unconventional meanings. The charts above show two groups of frequently used symbols. Those in the column at left represent mathematical operations; at right are symbols that either issue a command to the computer or are essential elements within a command.

Punctuation marks and many of the symbols used in mathematics, accounting and other similar operations appear on 22 of the Commodore's 66 keys. Many of these marks and symbols are assigned new meanings and uses in addition to their conventional ones, giving the keyboard, within its size limitations, maximum flexibility.

Two of the symbols used for mathematical operations are unusual. On the 64, an asterisk is used in place of an *x* to denote multiplication, because the computer understands *x* only as a letter. If you try to use it in a multiplication problem, you will get a wrong answer. Also, because the 64 cannot produce superscripts, an exponent is identified by the vertical arrow on the key to the left of the **RESTORE** key.

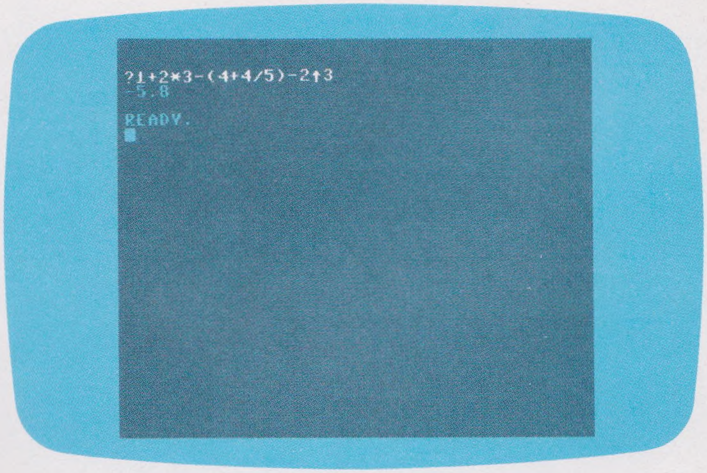
Some of the symbol keys are redefined in BASIC. The question mark, for instance, becomes an abbreviation for **PRINT,** a frequently used BASIC command, which instructs the computer to display on the screen any numbers, text or other data that follow. When you want the computer to display the results of mathematical calculations, you must use the **PRINT** command. The calculation will be carried out in what is called immediate mode: As soon as you have typed the problem and pressed the **RETURN** key, the computer will execute the command immediately, without storing it in its memory. The

If the BASIC command **PRINT** appears at the beginning of a line, the 64 will solve problems like the one shown at right. An equal sign is unnecessary; the computer automatically performs the calculation required by the plus sign as soon as you press the **RETURN** key. The first number of the solution appears in the second column, leaving the first column free for a positive or negative sign. Unless instructed to, the computer does not print a positive sign.



```
PRINT 4+5
 9

READY.
█
```

On the screen at right the question mark is a shorthand version of the **PRINT** command, and the vertical arrow indicates that the number following is an exponent. When the **RETURN** key is pressed, the computer performs the various calculations required in a standard order. It first evaluates the numbers in parentheses, and then performs exponentiation; next it does the multiplication and division, and last of all, addition and subtraction. The negative sign of the solution occupies the first column.



```
?1+2*3-(4+4/5)-2↑3
-5.8

READY.
█
```

results of the calculation appear on the following screen line.

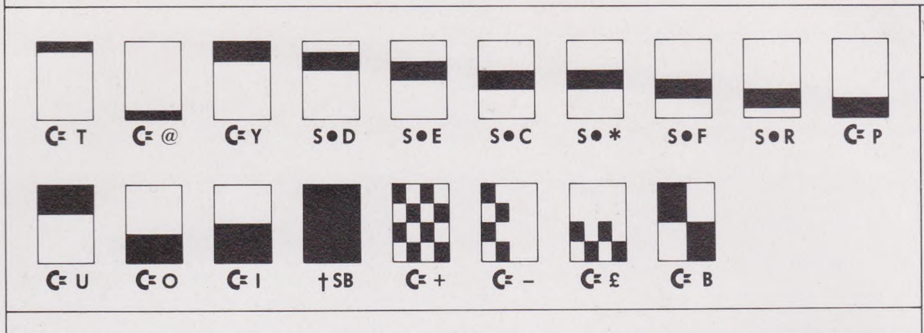
Another mode, called quote mode, is widely used in programming. In this mode, certain keys such as the left/right and up/down cursor controls produce symbols not shown on the keys themselves. You call this mode into play by typing an odd number of quotation marks. If you put the computer into quote mode through a typing error, you can escape from it by pressing **RETURN.**

An Aunt to Aid the Memory

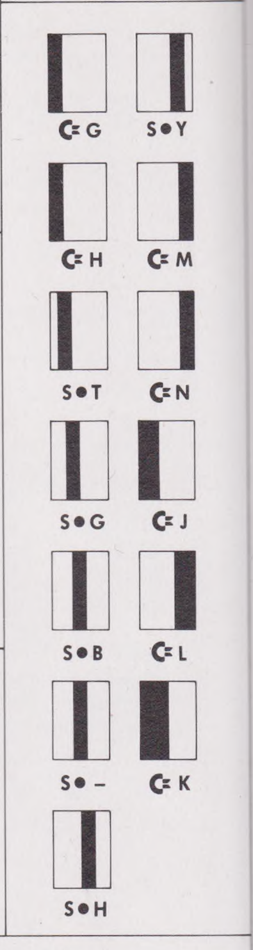
The computer performs arithmetic operations as you would — in several stages, working each time from left to right and following a conventional order of precedence. Numbers grouped within parentheses are processed from the innermost set to the outermost set. In each set, numbers with exponents are evaluated first, and then the other arithmetic operations are performed. The phrase "My Dear Aunt Sally" is a device for remembering this order of precedence. The first letter of each word and the order of the letters stand for the operations and their order — **m**ultiplication and **d**ivision first, followed by **a**ddition and **s**ubtraction.

The Graphic Symbols of the Keyboard

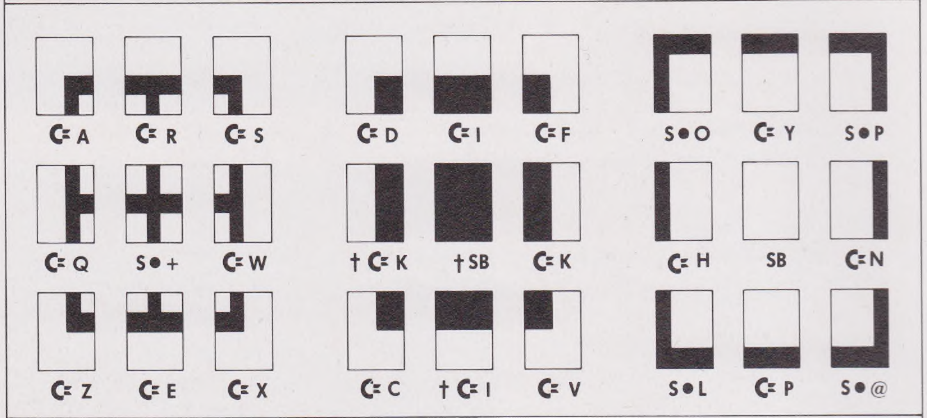
HORIZONTAL LINES, BARS AND GRIDS



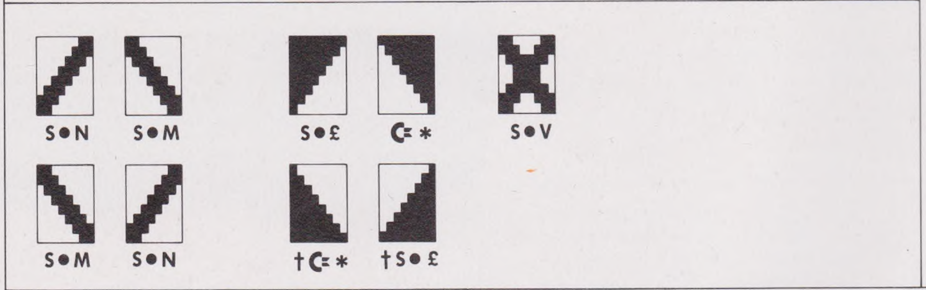
VERTICAL LINES AND BARS



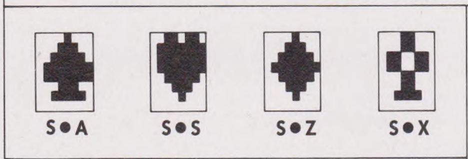
RECTANGLES



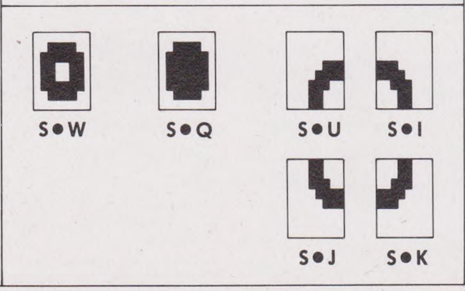
DIAMONDS AND DIAGONALS



CARD SUITS



OVALS AND ARCS



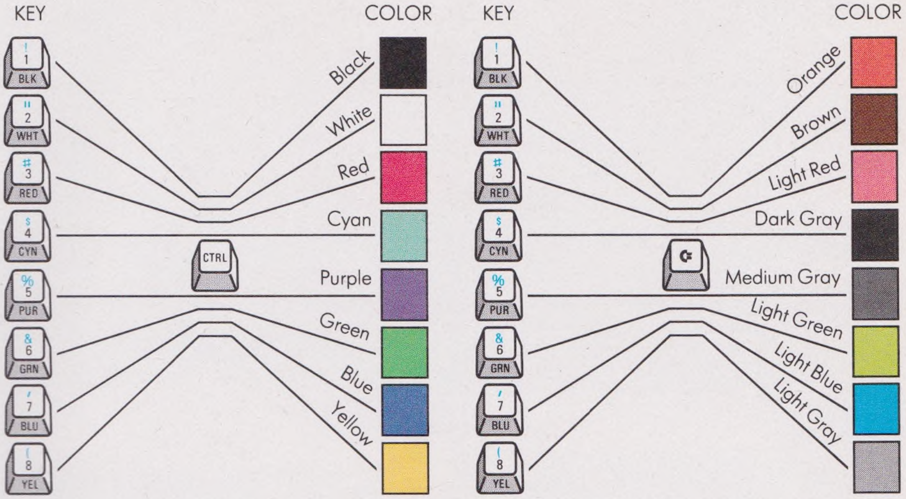
To print any of the Commodore's 62 graphic symbols, two keys are required: either the **SHIFT** key or the **COMMODORE** key, plus the key bearing the symbol on its front. In this chart, the key combination is shown beneath each symbol. The computer must be in graphic mode — with all letters on the screen displayed in upper case — in order to print symbols. To reverse the colors of a character and its background for more variation in a graphic display, press **CONTROL** and **REVERSE ON**. To return to the original color arrangement, press **CONTROL** and **REVERSE OFF**, or simply press **RETURN**. In addition to the symbol keys, the chart includes the space bar, which is used to make a rectangle.

†Hold down **CONTROL** and press **REVERSE ON,** then the pair of keys listed. After symbol is made, hold down **CONTROL** and press **REVERSE OFF.**

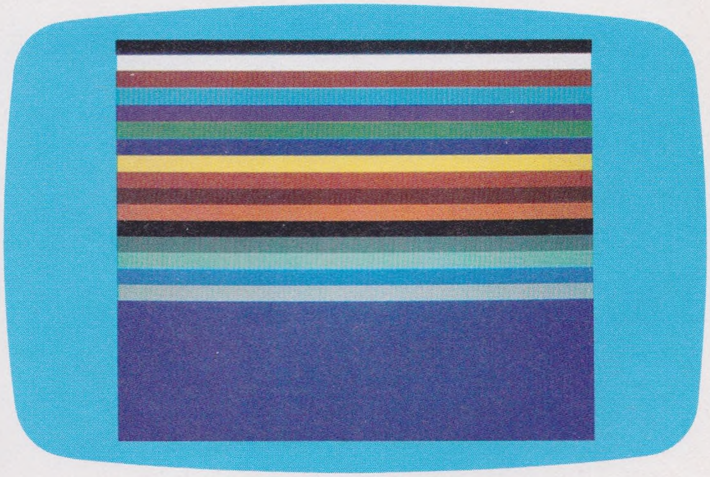
S• =SHIFT KEY   C= =COMMODORE KEY   **SB** =SPACE BAR

Keying Up for Color and Graphics

In the diagrams at right, lines connect number keys to the colors that they control. To select the colors in the chart at near right, hold the **CONTROL** key down and press the appropriate number key. For the colors at far right, hold the **COMMODORE** key down while pressing the number key. All subsequent characters will appear in the color you have chosen. To reverse cursor and background colors, press **CONTROL** and **REVERSE ON** together.



To create a striped screen like the one shown at right, hold down **SHIFT** and press **CLEAR/HOME** to return the cursor to home position, then hold down **CONTROL** and press **REVERSE ON**. Use the 16 combinations of keys shown in the charts above to change the cursor color, and press the **SPACE BAR** to move the cursor across the screen; it will leave a line of color. When the cursor reaches the first column of the following line, release the **SPACE BAR**, select a new color and press the **SPACE BAR** again.

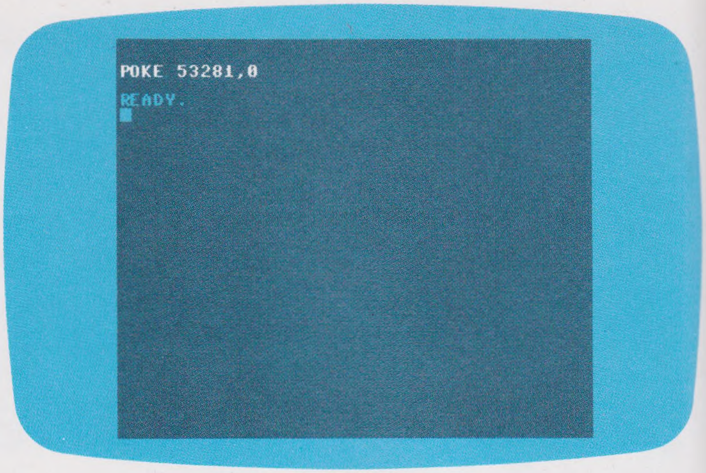


The 64 has excellent color capability. When color is combined with the many graphic symbols the computer can print, there is a wide range of possibilities for creating visually exciting screen displays. The keyboard alone, with no special instructions or programs, gives you access to the color and graphics features.

Sixteen colors are available, via the number keys 1 through 8. Two keys, **CONTROL** and **COMMODORE**, work much like a typewriter shift key; a key that alone produces one character will, when pressed in tandem with the shift, produce a different character. The **CONTROL** key plus numbers 1 through 8 give you one group of colors. The other eight colors are also selected through the same eight number keys, when you press them while holding down the **COMMODORE** key.

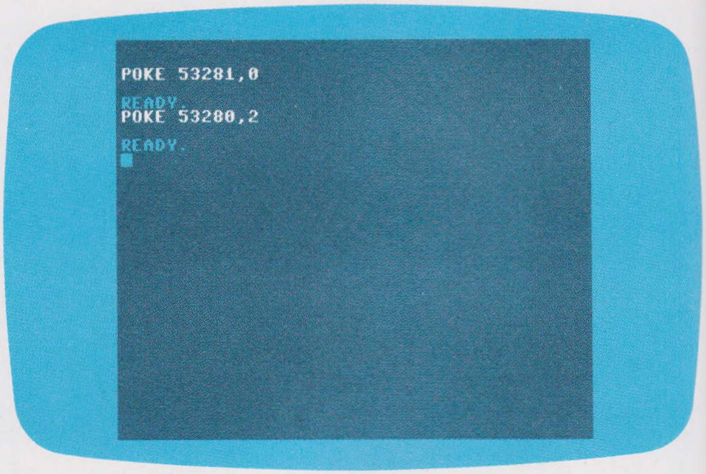
The **SPACE BAR**, like many of the other keys on the 64's keyboard, has graphics potential in addition to its ordinary function. First select a cursor color — the color in which characters appear — then press **CONTROL** and the **REVERSE ON** key together. Hold the **SPACE BAR** down to make a horizontal stripe in the cursor color you selected. For a properly adjusted color display on a TV set being used as a monitor, make horizontal stripes with the **SPACE BAR** in white, red, cyan, purple, green, blue and yellow; then use these stripes as guides when you adjust the TV controls.

New Colors for Border and Background



```
POKE 53281,0

READY.
■
```

To replace the usual dark blue background of the screen with another color, type in the BASIC command **POKE 53281** followed by a comma and the number that encodes your color choice. In the example here, the number is zero. When you press **RETURN,** the screen is converted from dark blue to black.



```
POKE 53281,0

READY.
POKE 53280,2

READY.
■
```

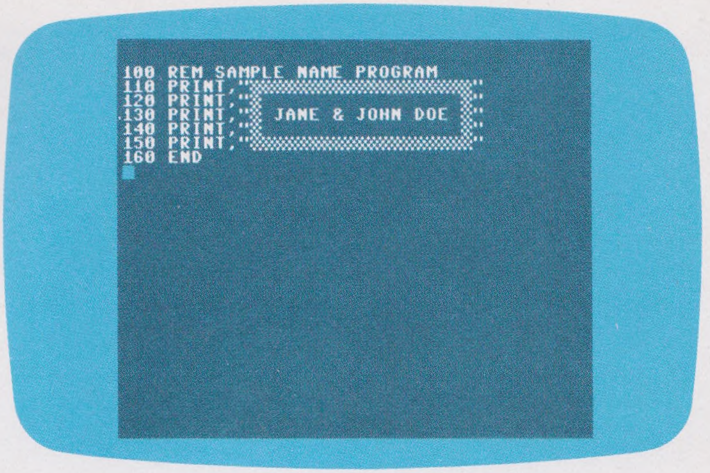
To change the colors of both background and border, type the two **POKE** commands shown at right, pressing **RETURN** after each command. The screen will change in two stages, background first and then border, to the colors shown here. To return the screen to its usual shades of blue, hold down **RUN/STOP** and press **RESTORE.**

BASIC provides two commands for changing the colors of the border and the background on the monitor screen. The commands differ by only a single digit. For border color, the command is **POKE 53280,x** with *x* being a number chosen from the list below. For example, **POKE 53280,4** produces a purple border. For background color, the command is **POKE 53281,x.** When you are using these two commands, bear in mind that the pairings of colors and numbers used for background and border changes are different from the color-number associations that appear on the keyboard and that alter cursor color *(page 33).* The numbers to use with the **POKE** commands are:

NUMBERS FOR COLOR CHANGE

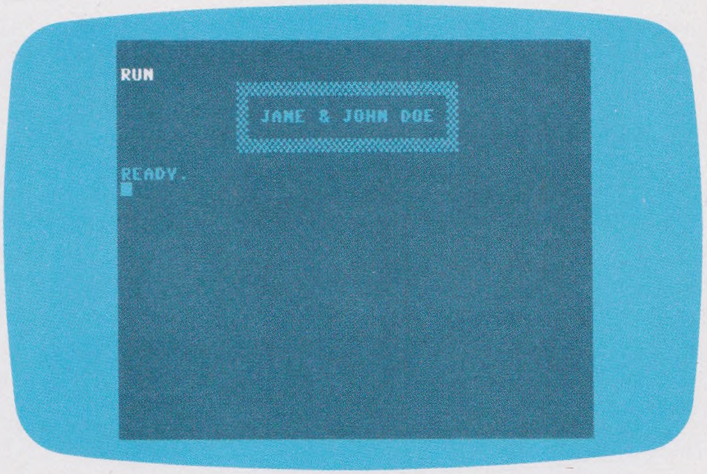
0 BLACK	**8 ORANGE**
1 WHITE	**9 BROWN**
2 RED	**10 LT. RED**
3 CYAN	**11 DARK GRAY**
4 PURPLE	**12 MED. GRAY**
5 GREEN	**13 LT. GREEN**
6 BLUE	**14 LT. BLUE**
7 YELLOW	**15 LT. GRAY**

The Elements of a Simple Program



```
100 REM SAMPLE NAME PROGRAM
110 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
120 PRINT,"▒                  ▒"
130 PRINT,"▒ JANE & JOHN DOE  ▒"
140 PRINT,"▒                  ▒"
150 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
160 END
```

Copy the program shown on the screen at right on your 64, substituting your name on line 130. Press **RETURN** when you complete each line. The box around the names is composed of the left-hand graphic character appearing on the front of the **PLUS** key. On lines 120 and 140, move the cursor with the **SPACE BAR** to make the blank spaces between the graphic characters.



```
RUN
          ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
          ▒                  ▒
          ▒ JANE & JOHN DOE  ▒
          ▒                  ▒
          ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

READY.
```

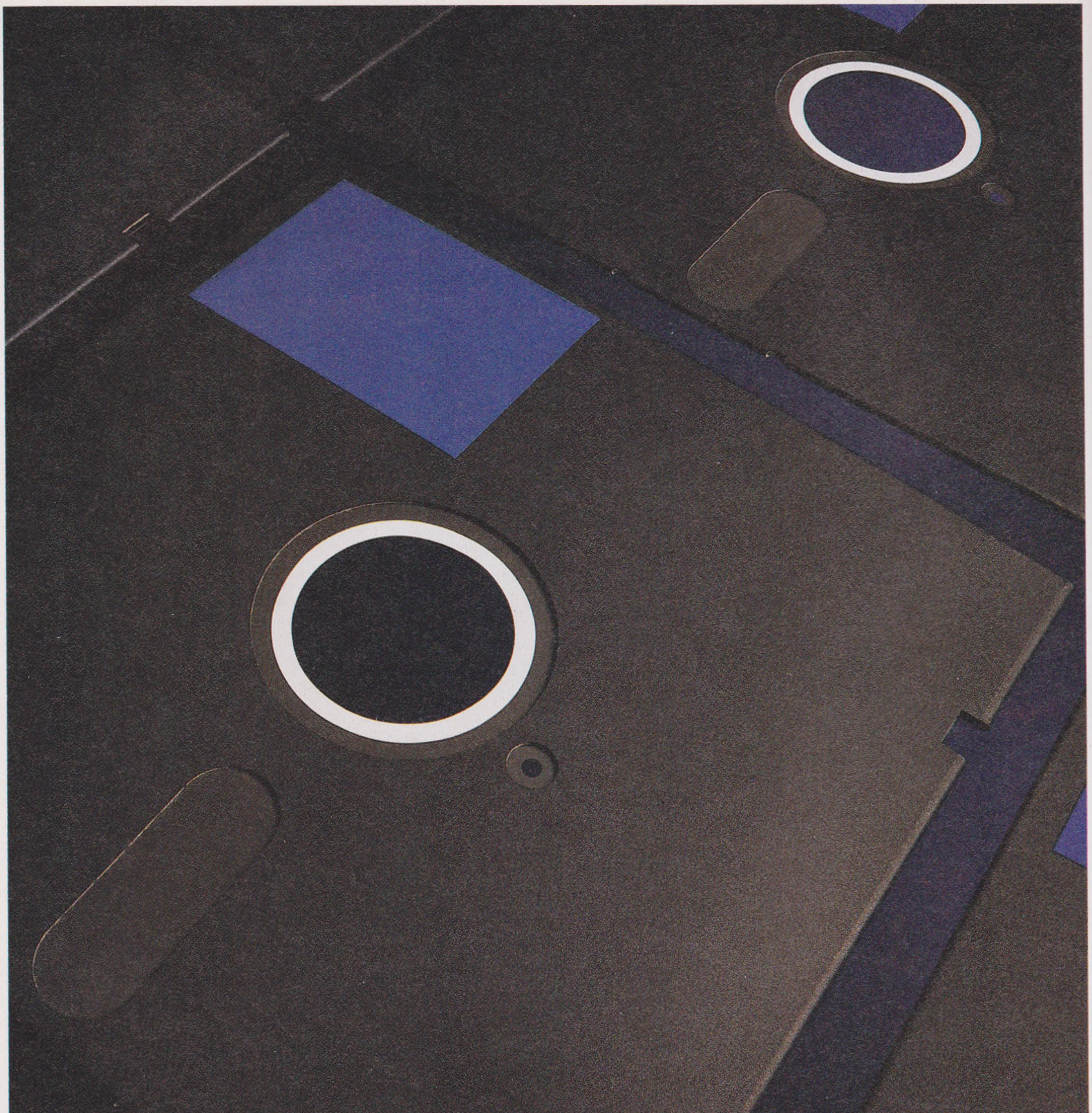
After you have entered the program shown above, hold down **SHIFT** and press **CLEAR/HOME,** clearing the screen. To run the program, type the command **RUN** and press **RETURN.** Your name surrounded by a box will appear on your screen, looking like the example shown at right.

Instructions you write, to tell your computer what to do, must be couched in BASIC and organized in program form. An essential element in any program is line number. If you preface a line with a whole number, followed by a BASIC command such as **PRINT,** the 64 will read and store the information on that line. Lines are often numbered in increments of 10 so there are locations where new lines can be inserted without disrupting the sequence of numbers.

The line number signals the computer to operate in program mode — that is, to store information a line at a time as the **RETURN** key is pressed, until another command is given. In this mode, you can enter hundreds of lines before the computer processes the information; that is the difference between program mode and immediate mode, in which the computer processes each line as soon as **RETURN** is pressed.

A program's first line contains the abbreviation **REM** (for "remark"), followed by the title. This formulation tells the computer that the title is not information on which it will act later, but is instead an aid for the user.

When a program is complete, type the command **RUN** and press **RETURN** to signal the computer to execute the program. If you want to read over a program or make corrections or additions, type **LIST** and press **RETURN** to bring it back from memory to the screen.



Saving and Retrieving Data with Peripherals

Your Commodore 64 is a versatile machine, capable of performing many tasks. Each task has its own set of instructions, called software, to tell the computer exactly how to use its circuits and mechanisms (hardware) to do the job at hand.

The 64's keyboard unit and the monitor are the core of the hardware system, but three basic peripherals — the Datasette recorder, the disk drive and the printer — greatly enhance the usefulness of the 64. When the 64 is turned off, all of the information in its random access memory is lost. The peripherals serve as adjuncts to memory, saving programs or data on tape, disk or paper until you need them. This chapter shows how to use these devices to store and retrieve programs and data. Most programs for the Commodore 64 are stored on cassettes or disks, to be loaded into the computer when required. You can buy commercial programs. You can also buy blank cassettes or disks to record programs you create.

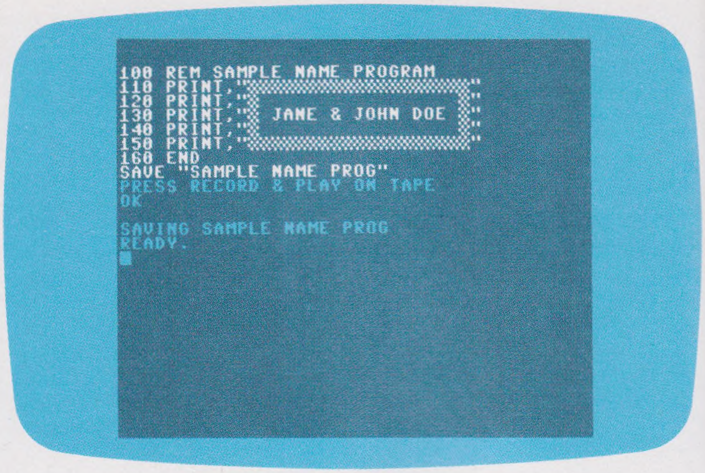
In contrast to the programs on cassette or disk are those built into the Commodore's memory. BASIC, the language in which the 64 communicates, is one such program. Even more fundamental than BASIC is the set of programs called the operating system. These programs give the computer essential instructions, such as how to interpret information received through the keyboard or other input devices. The Commodore 64 actually has two separate operating systems, one in the keyboard unit and another, called DOS (short for Disk Operating System), in the disk drive. Any program you write or buy relies on an operating system to do the low-level work of taking in, storing and putting out information; in turn, each program tells the computer how to further process this information to yield desired results.

To use a program, you give the computer a few commands in BASIC that are tailored to the operation and the device that is to perform it. Each peripheral has what is called a device number. This number, part of the information built into the 64's read-only memory, is included in the commands so that the 64 knows instantly which peripheral to communicate with. The Datasette recorder is Device 1, the printer Device 4, and the disk drive Device 8. If you do not specify the device, the 64 will either send you an error message or open a line of communication to a default device — the Datasette or the monitor.

Floppy disks *(left)* are part of the information storage and retrieval system that makes the Commodore 64 a powerful tool. Once the programs and data stored on the disks are fed into computer memory via the disk drive, they direct the activities of the 64.

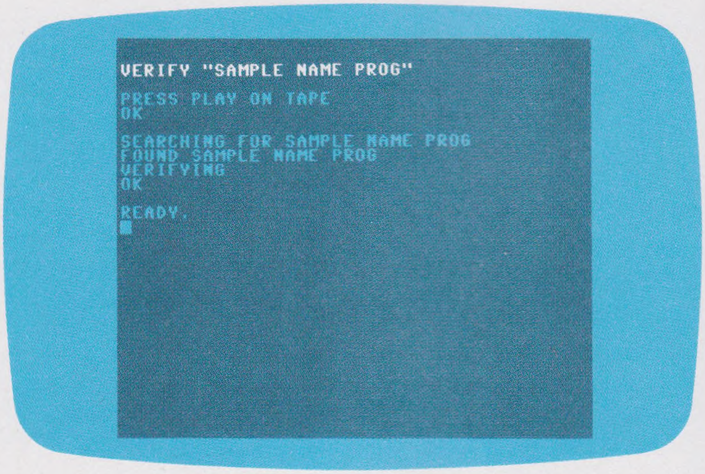
Saving and Loading with a Datasette

To save **SAMPLE NAME PROGRAM** *(right)*, put a tape into the Datasette, type in the **SAVE** command, and press **RETURN**. The computer will instruct you to press the Datasette's **RECORD** and **PLAY** buttons. While the program is being saved, the screen is blank. The **SAVE** command and the computer's instruction then reappear on the screen, plus a message stating what the computer has been doing. The **READY** message indicates you can give new instructions.



```
100 REM SAMPLE NAME PROGRAM
110 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
120 PRINT,"▒                  ▒"
130 PRINT,"▒ JANE & JOHN DOE  ▒"
140 PRINT,"▒                  ▒"
150 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
160 END
SAVE "SAMPLE NAME PROG"
PRESS RECORD & PLAY ON TAPE
OK

SAVING SAMPLE NAME PROG
READY.
■
```

To see if a program was saved properly, rewind the tape to its beginning, type the **VERIFY** command shown on the first line at right, and press **RETURN**. The 64 then instructs you to press the **PLAY** button and goes blank. When the program has been found, **SEARCHING** and **FOUND** messages appear on the screen. Press the **COMMODORE** key, or just wait a few seconds. The screen goes blank as the 64 checks the program. After verification, all the messages shown here appear on the screen. Note that the C-64 shortened the program name to 16 characters, since that is all it can accommodate.



```
VERIFY "SAMPLE NAME PROG"

PRESS PLAY ON TAPE
OK

SEARCHING FOR SAMPLE NAME PROG
FOUND SAMPLE NAME PROG
VERIFYING
OK

READY.
■
```

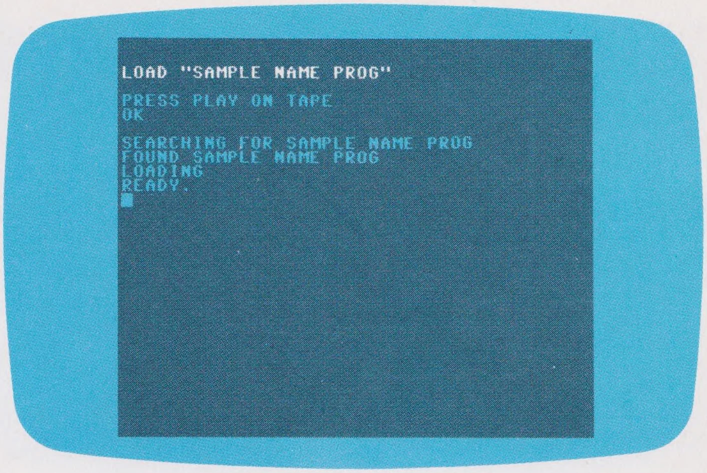
The processes of saving and loading programs on cassette tape are initiated by BASIC commands, controlled by the 64 and executed jointly by the 64 and the Datasette recorder. The Datasette is easy to master and will give good results if the tapes are handled and prepared properly.

Before saving a program on a new blank cassette tape, put the tape in the Datasette (which opens when you press the **EJECT** button), close the Datasette and press the **FAST FORWARD** (**F.FWD**) button. When the tape has reached the end, press **STOP**. This step evens out the tension on the tape and helps ensure reliable recording. Next, press **REWIND** until the tape has rewound to the beginning. The tape is now ready for use.

To reduce the risk of accidentally erasing a program by recording a new one over it, you could put only one program on each side of any cassette. You can buy standard short tapes that hold 10 to 15 minutes' worth of program per side. If you use longer-playing tapes that can accommodate several programs on each side, keep track of where each program begins and ends.

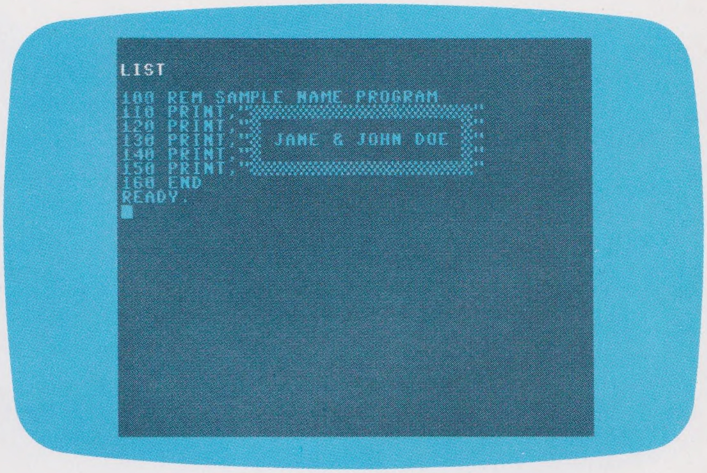
When you are ready to save a program on a new tape, reset the Datasette tape counter to 000 and follow the steps shown at top left. Once the program is saved, note the number on the counter and write it on the label for that side of the tape, along with the program name. When you

To load the **SAMPLE NAME PROGRAM** from tape into memory, type the **LOAD** command shown on the first line at right. Press the **PLAY** button of the Datasette when the computer tells you to. The screen will go blank, then the **SEARCHING** and **FOUND** messages will appear. The screen will again go blank and remain so until the program has been loaded. All of the messages shown at right will then appear.



```
LOAD "SAMPLE NAME PROG"

PRESS PLAY ON TAPE
OK

SEARCHING FOR SAMPLE NAME PROG
FOUND SAMPLE NAME PROG
LOADING
READY.
█
```

To make sure that **SAMPLE NAME PROGRAM** has been loaded into memory, type the command **LIST** *(right)*. The 64 will respond by displaying every line of the program as it was saved on tape. The **READY** message and the flashing cursor follow the program listing. To execute the program, type **RUN**.



```
LIST

100 REM SAMPLE NAME PROGRAM
110 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
120 PRINT,"▒                ▒"
130 PRINT,"▒ JANE & JOHN DOE ▒"
140 PRINT,"▒                ▒"
150 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
160 END
READY.
█
```

are ready to save a second program on that side, fast-forward the tape until the counter shows that you have passed, by a count of 10, the end of the first program. If you later modify the first program, save the new version on a fresh section of tape.

There are several ways of loading a program from cassette tape into the 64. You ordinarily begin with the **LOAD** command *(above, right)*. If you omit the program name, the computer will load the first program it finds. The 64 will instruct you to **PRESS PLAY** on the Datasette. When the program has been loaded, **READY** appears on the screen. As soon as you type in **RUN** and press **RETURN**, the program will be executed.

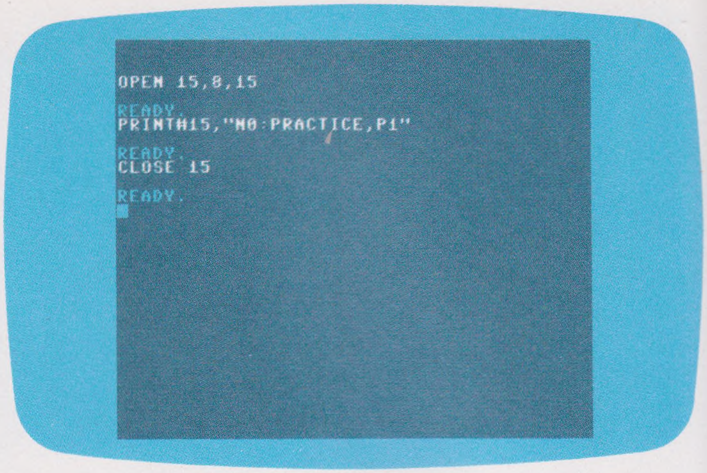
A shortcut can be used if there is only one program on one side of the tape or if you want to load the first program on a side. Hold down the **SHIFT** and the **RUN/STOP** keys. The message **PRESS PLAY ON TAPE** will appear on the monitor. When you press the Datasette's **PLAY** button, the 64 will automatically load the first program it finds on the tape. After loading, the 64 will run the program, without further keyboard input.

Some commercial programs require you to add **1,1** to the **LOAD** command after the quotation marks that follow the program name.

During saving and loading, the screen sometimes displays the message **OK**. Like a check, this indicates a task or step has been completed.

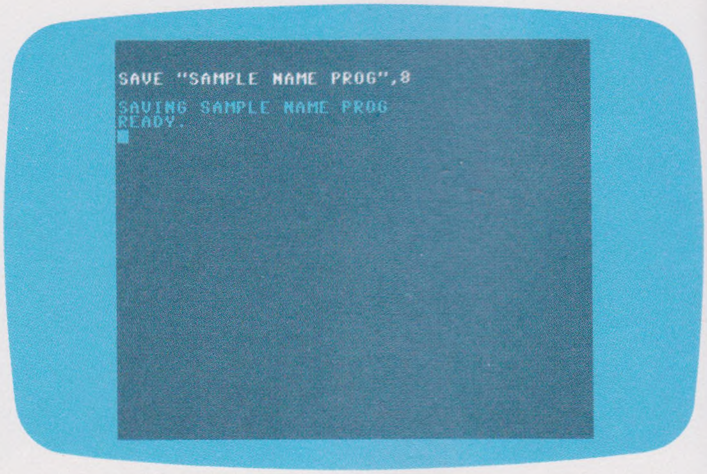
A Format for Fast Storage and Retrieval

To format a disk, type the command shown in the first line at right. At the **READY** message, type the line beginning with **PRINT** as shown; **N** stands for **NEW**. The 64 then formats the disk and records on it the name, up to 16 characters, assigned it (in this case, **PRACTICE**), plus a two-character disk identification (here, **P1**). The end of the formatting process is signaled by another **READY** message. Type the command **CLOSE 15** to close the line of communication.



```
OPEN 15,8,15

READY.
PRINT#15,"N0:PRACTICE,P1"

READY.
CLOSE 15

READY.
```

To copy the **SAMPLE NAME PROGRAM** to a disk, type the **SAVE** command exactly as it appears at right. Surround the title with quotation marks and type a comma to separate this part of the command from the disk drive's device number (**8**). While the program is being recorded, the **SAVING** message is displayed. No **VERIFY** command is needed, because DOS automatically verifies that the program has been saved. If the red light on the disk drive flashes continually, the program was not saved.



```
SAVE "SAMPLE NAME PROG",8

SAVING SAMPLE NAME PROG
READY.
```

Information stored on floppy disks is organized into collections called files, which are comparable to the folders in an orderly file cabinet. A program is one kind of file; a body of data, such as a mailing list, is also stored in file form.

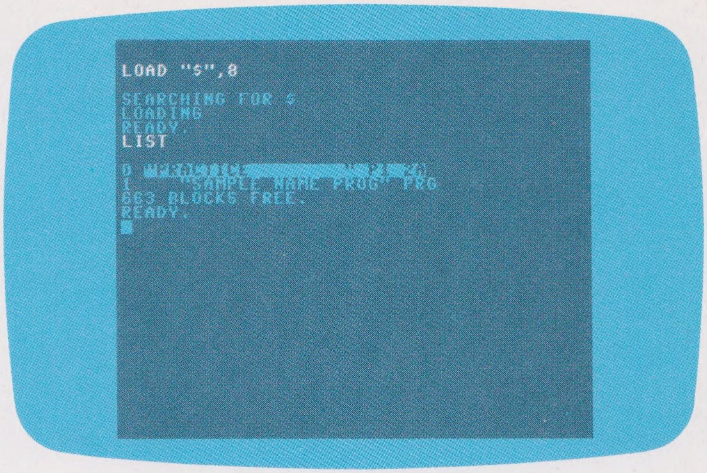
Each disk used with the 1541 disk drive can hold almost 170,000 characters, the equivalent of about 85 typed pages. Sifting through that much information to locate a particular file could cause delays; to avoid them, the Disk Operating System (DOS) uses a reference system, called a format, on each disk.

The **NEW** command causes DOS to format a disk, that is, to record on it a pattern of magnetic sectors and tracks. As part of the formatting operation, the disk must be given a name, up to 16 characters long, and also a two-character disk identification, which permits the disk drive to distinguish between two disks.

Once the disk is formatted, DOS can store a file on it in a particular spot or a collection of spots. DOS notes both track and sector, and that information, along with the name of the file, is entered in a list on the disk, called the directory. When you need a file, DOS scans the directory for its name and location. One sector of a track on the disk, called the Block Availability Map (BAM), keeps track of which sectors have been filled with files and which are still available.

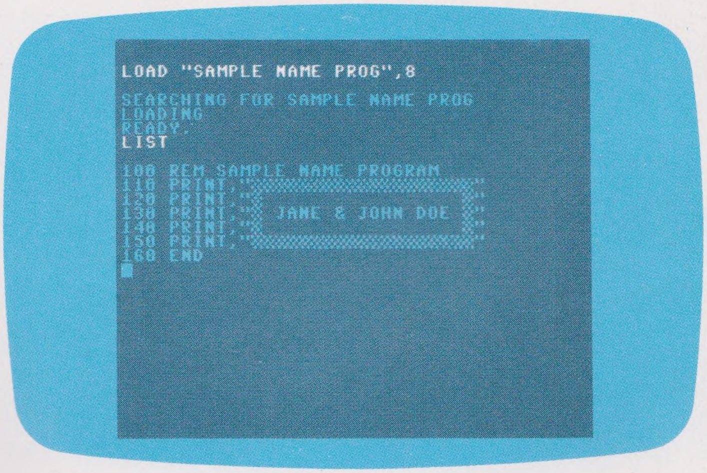
WARNING: Formatting a disk

To copy a disk's directory into computer memory, type the **LOAD** command *(first line at right)*. The dollar sign is the DOS abbreviation for the directory. When the directory has been loaded, type **LIST** to tell the 64 to print the directory to the screen. When the directory appears, the first items shown are the disk's name, **PRACTICE**, and its ID, **P1**. On the same line, **2A** refers to the version of DOS used to format the disk. The **SAMPLE NAME PROGRAM** occupies one block of space on the disk; 663 are vacant.



```
LOAD "$",8

SEARCHING FOR $
LOADING
READY.
LIST

0 "PRACTICE        " P1 2A
1    "SAMPLE NAME PROG"  PRG
663 BLOCKS FREE.
READY.
█
```

To load the **SAMPLE NAME PROGRAM** into the 64's memory, type the **LOAD** command at right and press **RETURN**. When the copying process is complete, the **READY** message appears. If you want to see the program, type **LIST**; each line will appear on the screen exactly as it was originally written. To execute the program, type **RUN**.



```
LOAD "SAMPLE NAME PROG",8

SEARCHING FOR SAMPLE NAME PROG
LOADING
READY.
LIST

100 REM SAMPLE NAME PROGRAM
110 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
120 PRINT,"▒                  ▒"
130 PRINT,"▒  JANE & JOHN DOE ▒"
140 PRINT,"▒                  ▒"
150 PRINT,"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
160 END
█
```

erases all information previously stored there. Usually you need to format a disk only once — before you use it for the first time. If you reformat an old disk, be sure it holds no valuable files.

Every file must be named, so that DOS can enter it into the directory. The file name can also be up to 16 characters long. Files on commercial disks have already been named by the manufacturers. To look at the directory of a commercial disk or one on which you have saved your own files, use a two-command sequence. The first command is **LOAD "$",8** — in which the dollar sign stands for the directory. Once the directory has been loaded from the disk into memory, the command **LIST** brings the directory to the screen.

Whenever the disk drive is carrying out a task — formatting, loading or saving — its red light is on. If the red light flashes continually or if you get an error message in response to a command, turn the 64 off, wait at least 10 seconds and remove the disk. Then switch the 64 on, insert the disk, and begin again.

In addition to the disk commands mentioned on these two pages, there are other commands to the disk drive; some of the more frequently used commands are listed on page 42. The disk commands also have shortened forms, available in a program called the Wedge *(page 43)*.

Commands to the Disk Drive

The chart below lists eight frequently used disk commands, three of which are illustrated on pages 40-41. A shorter form of each command is available in a program called the Wedge *(box, below)*. One of these shortened forms for **NEW** is illustrated on the opposite page. Only the **LOAD** and **SAVE** commands have a one-line format; the other commands require three lines.

NAME	FORMAT	WEDGE FORM	USE

COPY	`OPEN 15,8,15`
	`PRINT#15,"C0:NEWFILE=0:OLDFILE"`
	`CLOSE 15`
		`@C0:NEWFILE=0:OLDFILE`	Makes a new copy of a program on disk; copy should be assigned a new file name. A variant of command links several programs into one.

INITIALIZE	`OPEN 15,8,15`
	`PRINT#15,"I0"`
	`CLOSE 15`
		`@I0`	Used when changing disks. Makes the drive recognize the new disk. After an error message, use to reset drive and eliminate flashing of light.

LOAD	`LOAD"FILENAME",8`	`/FILENAME`	Copies a file from disk into computer memory.

NEW	`OPEN 15,8,15`
	`PRINT#15,"N0:FILENAME,ID"`
	`CLOSE 15`
		`@N0:DISKNAME,ID`	Formats a new disk; erases and reformats old disk. If you omit the disk ID, the stored information is erased, but the old format remains.

RENAME	`OPEN 15,8,15`
	`PRINT#15,"R0:NEWNAME=0:OLDNAME"`
	`CLOSE 15`
		`@R0:NEWNAME=0:OLDNAME`	Allows file name to be changed.

SAVE	`SAVE"FILENAME",8`	`←FILENAME`	Copies a file in computer memory onto a disk.

SCRATCH	`OPEN 15,8,15`
	`PRINT#15,"S0:FILENAME"`
	`CLOSE 15`
		`@S0:FILENAME`	Erases the specified file from the disk, freeing the space it occupied for other files.

VALIDATE	`OPEN 15,8,15`
	`PRINT#15,"V0"`
	`CLOSE 15`
		`@V0`	Reorganizes unused blocks on the disk, making them available for use. Should not be used with random-access files.

How to Name Your Files

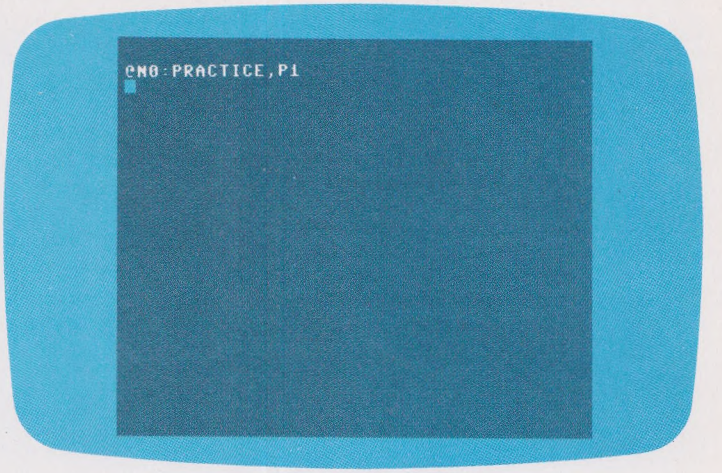
Commodore DOS recognizes the name of a file stored on a disk only if the name is written in a particular way. A file name, like a disk name, should contain from one to 16 characters; however, it should also be enclosed in quotation marks. Any combination of letters and numbers is acceptable, but you must avoid symbols, such as @, *, ? and $, since they have special meanings in BASIC, and the Commodore 64 cannot interpret them properly in the context of a name.

A short identifying tag, or extension, at the end of a file name can help you remember the contents better. For example, you might end a name with **.COR** for correspondence, or **.BAS** for programs written in BASIC, or **.14JUN** to date a preliminary version of a program. Extensions are also helpful when you want to find related files, such as all preliminary copies of a program.

Don't forget to type in the extension when you tell the computer to load a file. If you have a file called **NOTES.1**, DOS will be unable to find it if you ask for it simply as **NOTES**, and you'll get a **?FILE NOT FOUND ERROR** message.

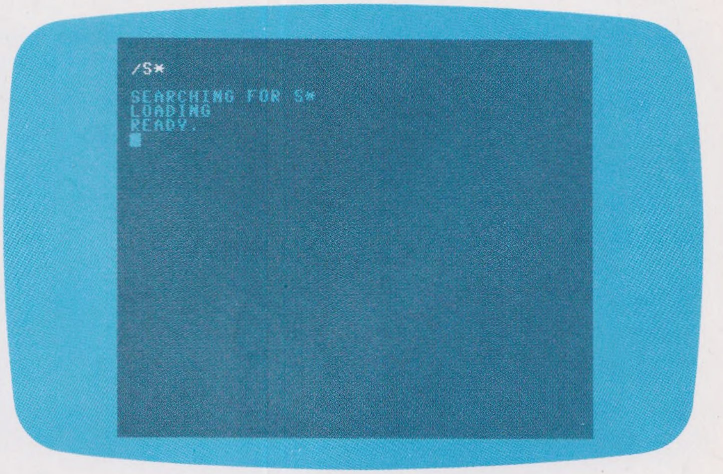
Mastering Disk Commands and Naming Files

When you wish to format a disk — named **PRACTICE** in the example at right — you can use the **NEW** command in a one-line version that is part of the Wedge program *(box, below)*. The shortened Wedge command fills the same function as the standard three-line form of the command. Other Wedge commands are listed in the chart opposite.



```
@N0:PRACTICE,P1
```

An asterisk can be used to replace almost all the characters of a file name specified in a command. In the example at right, only the single letter **S** is used to identify the file. The disk drive responds to this Wedge **LOAD** command by loading into computer memory the first program beginning with **S** that it locates in the disk directory.



```
/S*

SEARCHING FOR S*
LOADING
READY.
```

Shortcuts with the Wedge

The Wedge — also called DOS Wedge, C-64 Wedge, Universal Wedge or DOS Manager — is a program that consists of a few disk commands only two or three keystrokes in length. The Wedge streamlines several common disk procedures; for instance, it lets you use one short command to both load and run a program.

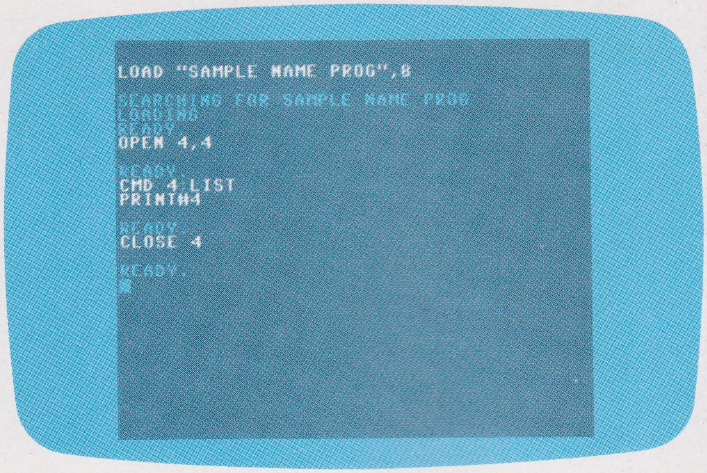
To use the Wedge from the 1541 test demonstration disk, type **LOAD "C-64 WEDGE",8** and, when loading is done, type **RUN**.

Wedge commands usually begin with either of two interchangeable symbols — the *at* symbol (@) or the *greater-than* symbol (>), whose shape gives the program its name. Wedge abbreviations for commonly used disk commands are shown on page 42 with the @ symbol. The Wedge version of **NEW**, to format a disk, is shown above; the full version is on page 40.

The Wedge has several variations of the **LOAD** command. Instead of **/** shown in the chart, **%** may be used. The command **@$** can replace the two-step **LOAD** and **LIST** sequence to list a directory *(page 41)*. To both load and run a program, use the ↑ key followed by the name of the program.

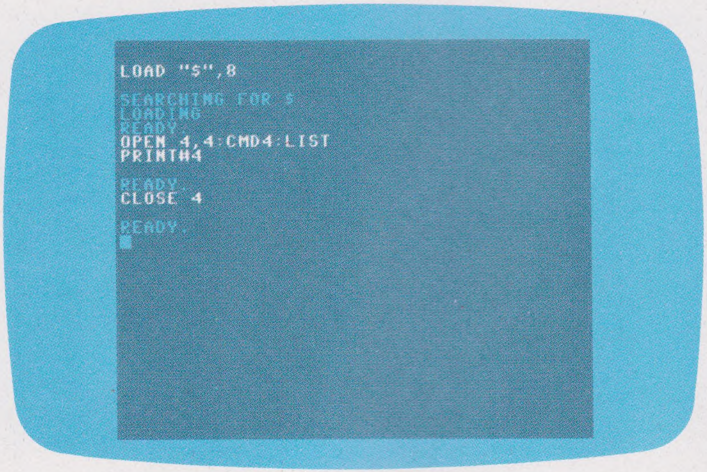
Command Sequences for Printer Output

This command sequence produces a listing on paper of a program stored on disk. In the **OPEN** command, the first number specifies the file and the second the printer's device number. Once the program is printed out, use the **CLOSE** command to cease communications with the printer. The file number must not change during an operation that uses the printer commands.



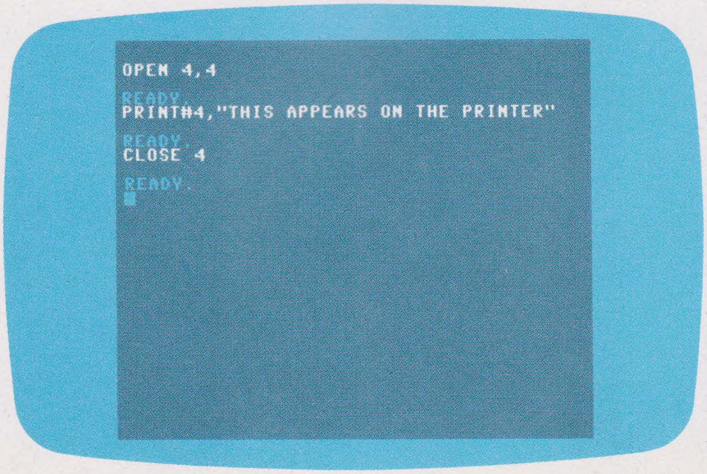
```
LOAD "SAMPLE NAME PROG",8
SEARCHING FOR SAMPLE NAME PROG
LOADING
READY.
OPEN 4,4

READY.
CMD 4:LIST
PRINT#4

READY.
CLOSE 4

READY.
```

To print out a disk directory, first load the directory, symbolized by the dollar sign, into computer memory and then follow the same sequence of commands that is used above to produce a program listing. You can paste the directory printout to the disk sleeve for quick identification of the contents.

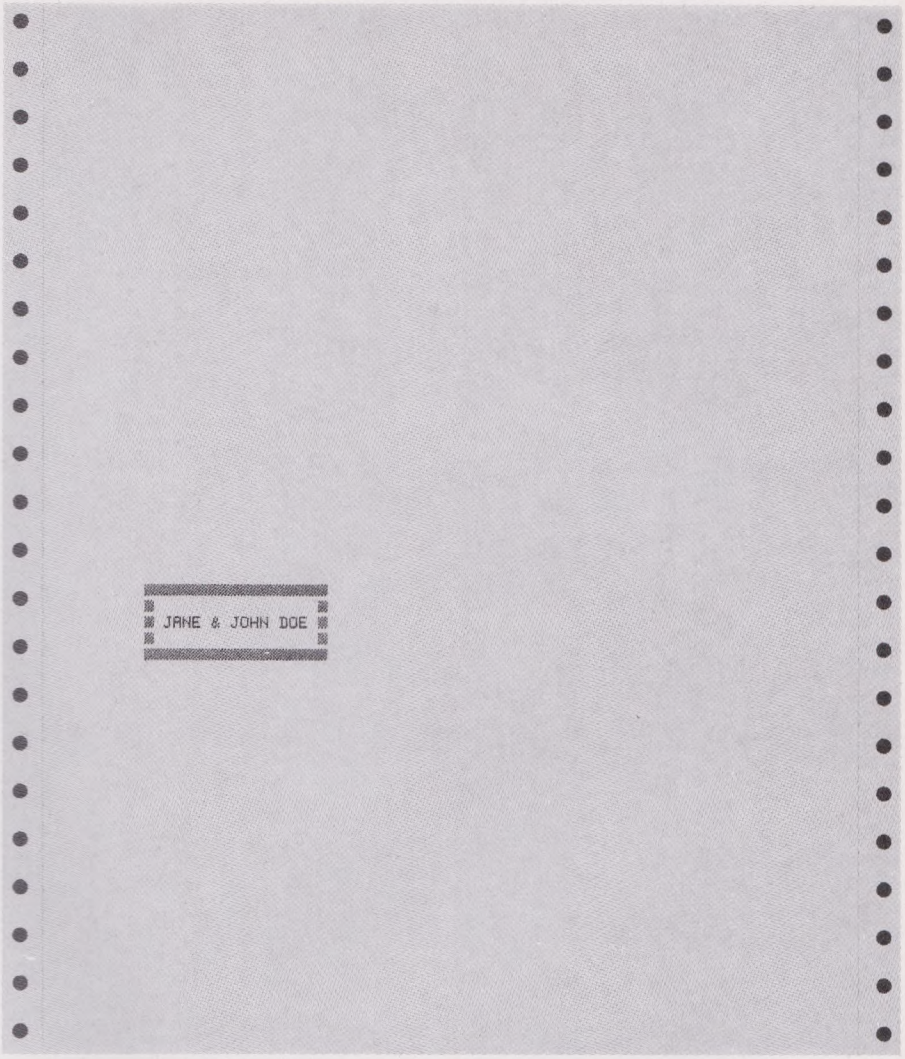


```
LOAD "$",8

SEARCHING FOR $
LOADING
READY.
OPEN 4,4:CMD4:LIST
PRINT#4

READY.
CLOSE 4

READY.
```

To print a data file, such as text or a sequence of numbers, omit the **CMD** and **LIST** commands and insert a comma immediately after **PRINT#4** as shown on the third line of the screen at right. If the data to be printed out is text, as in the example here, you must surround it with quotation marks. The text that will appear on paper is just what you typed between the quote marks. You do not need to use quotation marks when numbers are to be printed out.



```
OPEN 4,4

READY.
PRINT#4,"THIS APPEARS ON THE PRINTER"

READY.
CLOSE 4

READY.
```



```
JANE & JOHN DOE
```

Printed copy from a Commodore printer displays the execution of the sample name program. To obtain this printout, two lines containing printer commands were added to the program that appears on page 35: **105OPEN4,4:CMD4** and **155PRINT#4:CLOSE4**.

The output of a printer is sometimes more convenient to work with than the output that appears on the monitor screen, and it also provides an easy means of sharing information with other people. Perhaps the most common use for a printer is to produce a program listing of all the instructions that have been given to the computer. You may find it convenient to make major revisions in a program on paper and then to enter the corrections via the keyboard. You can also have a disk directory printed out for use as a table of contents on the disk's paper sleeve. Many commercial programs, such as spreadsheets and word processors, create data that must be printed out to be useful.

Five BASIC commands control the primary activities of a printer — **OPEN**, **CMD** (short for "command"), **LIST**, **PRINT#** and **CLOSE**. Do not abbreviate **PRINT** with a question mark. To keep the printer from printing computer messages such as **READY** at the end of a program, you need to put two commands on a single line; use a colon to separate them. Commodore printers are assigned the device number 4, which is used in the **OPEN** command. You must also choose a file number, which can be any number between 1 and 255. Use this number consistently in the **OPEN**, **CMD**, **PRINT#** and **CLOSE** commands within a single operation.

AGENTS: 3
SCORE: 11530
SCREEN: 14
VIDEO MONITOR
MODEL 1702
POWER
VIDEO
AUDIO
POWER
commodore
commodore
commodore

Entertainment Software: Pleasure and Creativity

Games and learning are strands of the same rope. By playing games that imitate the world, children sharpen their physical and social skills: coordination, memory, quick reflexes (both mental and physical), the ability to work with others. Prehistoric children may have tossed sticks and pebbles on the sand, or played at hunting with crude bows and arrows. Children since have learned to experience and master the world around them by playing such games as hide-and-go-seek and soccer, dollhouses and train sets, chess and parcheesi. Today's children have added computer games.

But you don't have to be a child to enjoy playing games on your Commodore 64, or to learn on it. Given the right software, the 64 can challenge not only the fingers but the minds of children and adults alike. Some software even encourages you to use the computer as a creative tool, by giving you easy access to the 64's superior graphics and sound capabilities.

Games for the 64 come in several guises, from the fast-action shoot-'em-ups of arcade-style entertainment to the familiar strategies of poker and chess, to the highly structured environments of adventure simulations. Some of these adventures are text-oriented interactive fiction: You must respond, via the keyboard, to a series of questions or statements. Other simulations fill the screen with vivid graphics.

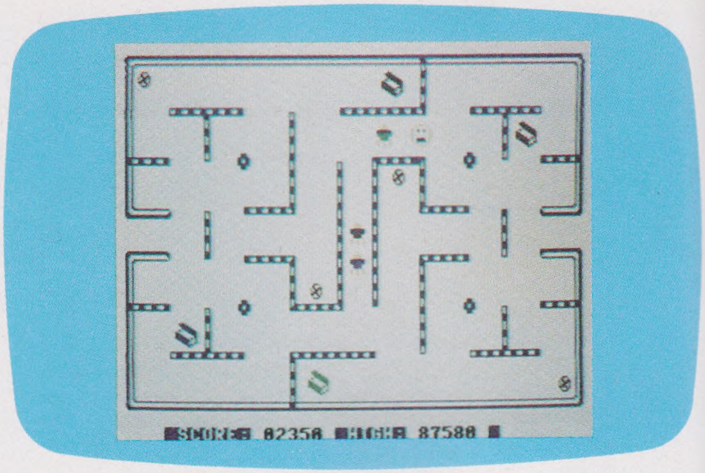
Simulations play an important part in educational software as well. As personal computers become more versatile, such approaches as drill-and-practice and electronic text are giving way to simulations and other programs that allow more interaction between machine and student. The best packages not only demonstrate ideas and concepts, but also test and evaluate a learner's performance. Moreover, they present the material through colorful graphics and a strong story line.

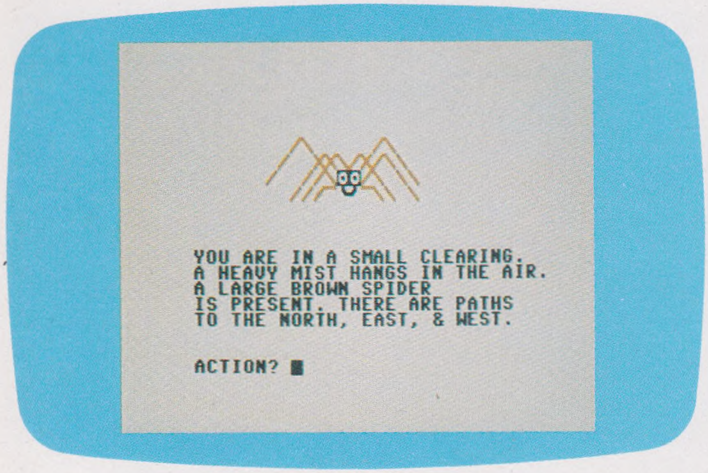
If an educational package seems too much like a glorified game, remember two things: A computer is only as good a teacher as its software allows it to be, and boring software is rarely used more than once. Consult with a schoolteacher who works with a computer for help in evaluating available programs. Whether labeled education or entertainment, the best of these software packages encourage you to learn from your mistakes — and to have fun with your computer while you do.

A joystick provides control over the fast-moving figures of an arcade-style game on a Commodore 64. In this game the player tries to make his marker climb moving ladders and avoid hostile figures, scoring points as it progresses. As the score mounts, the pace of the game speeds up and scoring becomes more difficult.

Playing Games with the Computer

The maze game displayed at right is typical of such action games for the Commodore 64. The maze is stationary, but the figures chase and dodge one another; one is controlled by a player using a joystick. Chests and obstacles within the maze represent rewards and hazards. The fun is in quick manual responses to changing situations — and in racking up points, as displayed at the bottom of the screen.





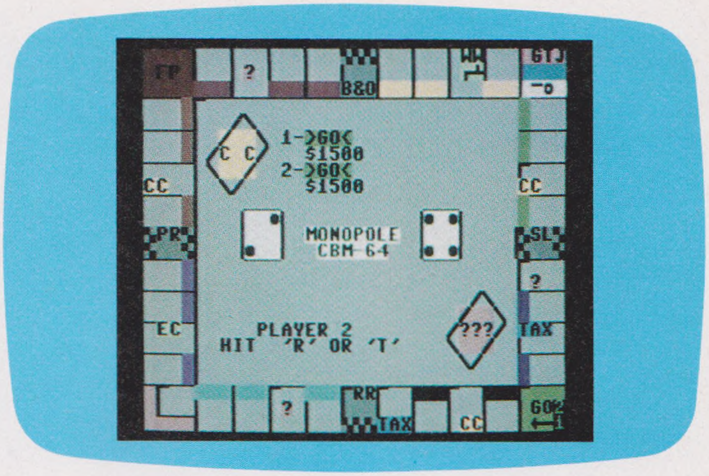
```
YOU ARE IN A SMALL CLEARING.
A HEAVY MIST HANGS IN THE AIR.
A LARGE BROWN SPIDER
IS PRESENT. THERE ARE PATHS
TO THE NORTH, EAST, & WEST.

ACTION? █
```

An adventure game such as the one shown here is more a test of the mind than of the motor skills. It appeals to the imagination by setting a scene, presenting a problem and then asking for your response. You may have to solve a riddle or escape danger. But you have to think, rather than maneuver, your way through a problem.

A wide variety of games that can be played on the Commodore 64 has developed, as programmers around the world have begun to take advantage of the computer's color and sound capabilities. The rules for a game are contained in a program, which is fed into the computer's memory from a cassette or disk. A player's moves in the game are entered through the keyboard or other control device *(opposite)*.

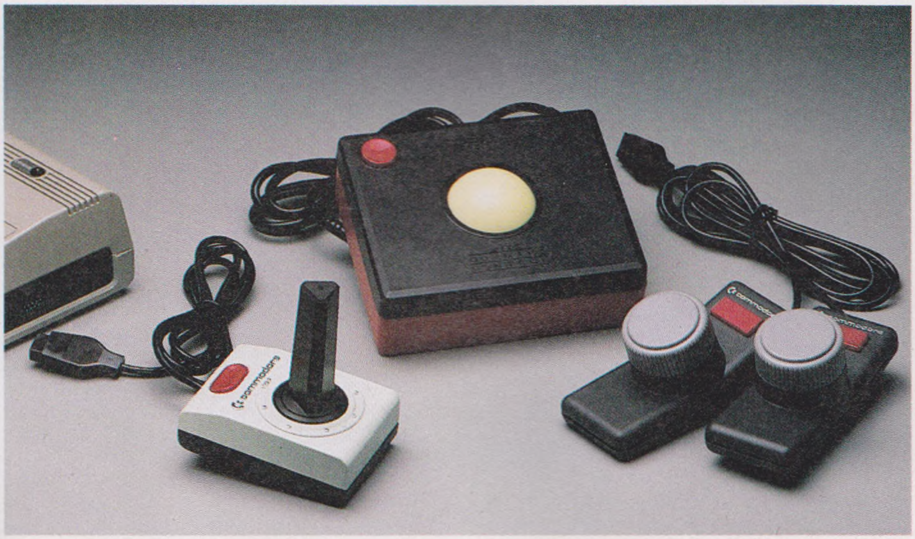
Many of the most popular games are similar to the ones offered in video arcades: action games, rich in color graphics and sound effects. Generally, they demand that players quickly evaluate information that flashes on the screen and — with equal speed — manipulate a control device. Most require players to shoot at images moving on the screen, or find a way through a maze, or steer an on-screen vehicle around obstacles. Action games are usually played against time; the longer the game goes on, the less time the players have to respond to what is happening on the screen.

Close relatives to the arcade-style action games are sports simulations, which reproduce on the screen the playing areas and rules of popular sports such as basketball, soccer and football. These games also call for good eye-hand coordination and quick reflexes, as the players maneuver their teams and the ball across the computer screen.



```
1->60<
  $1500
2->60<
  $1500

MONOPOLE
 CBM-64

  PLAYER 2
HIT  'R' OR 'T'
```

The screen at right displays a board game, set up for two players. In this game, the computer rolls the dice on command. It moves the pieces, automatically calculates money transfers and, when asked, will display net worths, property ownership or the rules. The 64 can also act as game board and as opponent and arbiter for games such as checkers, chess and backgammon.

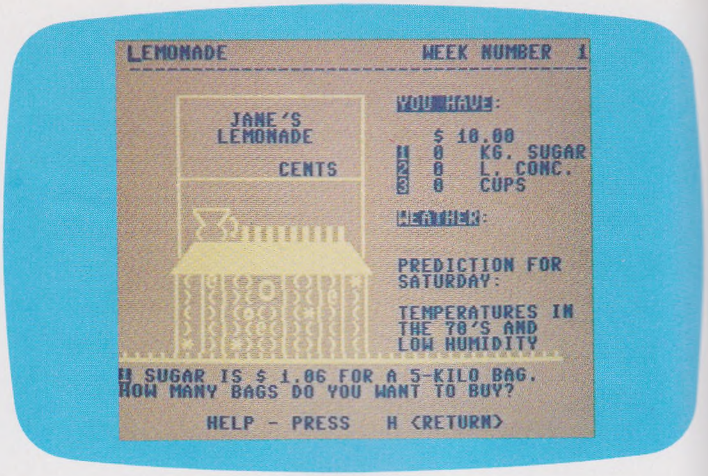


Fast-action games are best played with special control devices that plug into the control ports on the side of the Commodore 64. A pair of paddles *(right)* can be used by two players for games that require precise horizontal or vertical movement along a straight line. A joystick *(left)* commands movements in eight directions; a button on the box serves as a trigger. At top is a trackball, the most versatile controller; rolling the ball in its housing can move the cursor by minute steps in any direction.

Adventure games establish a situation using words — or graphics and words — and then put you into the situation. At different points in the story, you must make decisions that affect subsequent turns of the plot. For example, you may be cast as a detective called on to solve a mystery; or you might be a treasure-seeker or dragonslayer. You inform the computer of your decisions by typing in words such as "Go north" or "Read note." The better programs allow you to save the game you are playing so you can stop play and later resume where you left off — a useful feature for a game that could go on for hours, days or weeks.

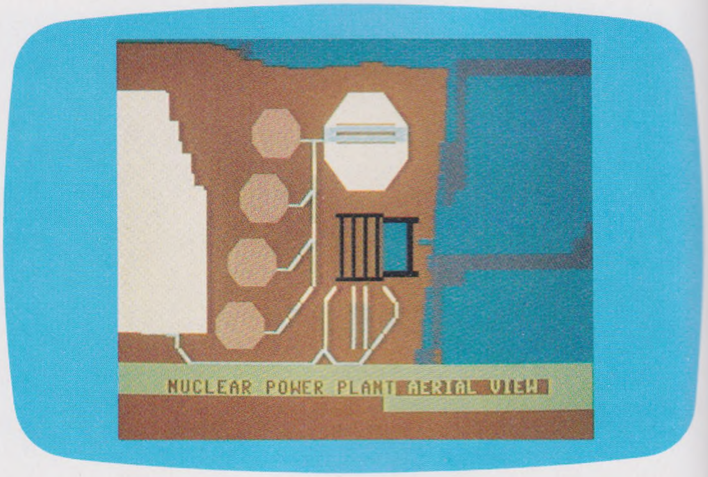
Game makers also offer electronic versions of chess and other strategy board games, and even card games such as poker. Some, like chess, you can play against the computer; some also allow you to play against a friend, putting the computer to work as a display board and record keeper. If you are playing against the computer, you can usually work from an easy level up to harder ones; the computer becomes a more skillful opponent either by giving itself more time to study a move or by learning from previous errors.

Pitting Your Mind Against the Computer



```
LEMONADE                     WEEK NUMBER  1
-------------------------------------------
                        YOU HAVE:
        JANE'S
       LEMONADE              $ 10.00
                        1   0    KG. SUGAR
              CENTS     2   0    L. CONC.
                        3   0    CUPS

                        WEATHER:

                        PREDICTION FOR
                        SATURDAY:

                        TEMPERATURES IN
                        THE 70'S AND
                        LOW HUMIDITY
1 SUGAR IS $ 1.06 FOR A 5-KILO BAG.
HOW MANY BAGS DO YOU WANT TO BUY?

        HELP - PRESS   H <RETURN>
```

The screen at right is part of an educational program that simulates real-life difficulties, then lets you work through them. In this case, the program presents the problems of running a lemonade stand and requires you to make choices about purchasing supplies and pricing your product. It then displays the results of your decisions and points out how you could have done better.



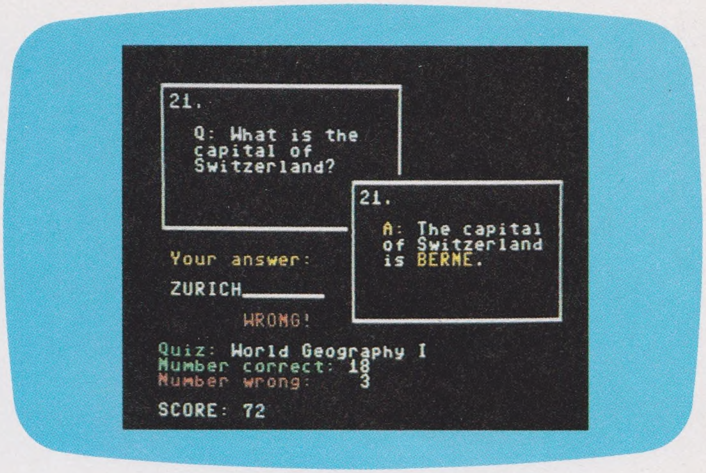
A tutorial program teaches by instruction rather than by interaction. The screen at right, displaying an aerial view of a nuclear power plant, is part of a program that uses text and graphics to teach the particulars of specific subjects. The program explains how nuclear power plants produce electricity — from the generation of heat by fission to the operation of huge generators.

The Commodore 64 has many of the attributes of a good teacher: It can call upon large amounts of knowledge, pose questions and score responses, keep track of progress and repeat teaching sequences with infinite patience. When you choose the right software to go with it, the 64 can be an excellent tool for computer-assisted instruction.

Good educational software holds the attention with some of the same devices that make computer games appealing — good sound and graphics, and routines that call for frequent interaction. The element of enjoyment is as important in computer learning as it is in the classroom.

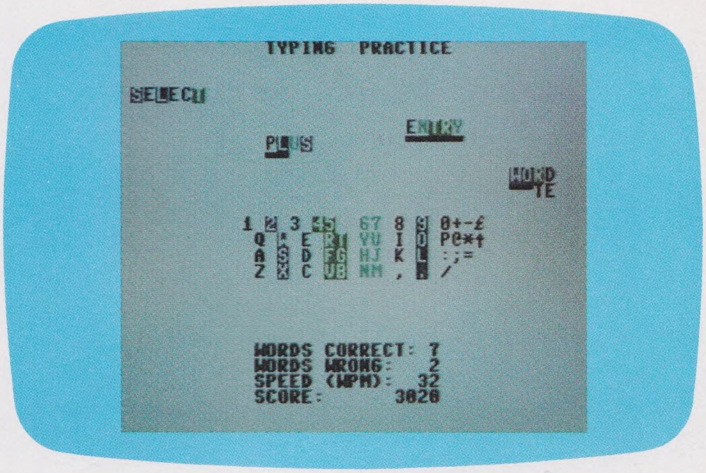
Several types of educational software are available for the 64. The most common leads you through practice drills, generally on information you have learned elsewhere, such as arithmetic, spelling or a language. Some drill programs keep a record of your correct and incorrect

The flash-card program at right tests knowledge of geography. First a question appears; then you type in an answer. The screen then displays the correct answer and the computer keeps track of your score. Some programs are set up to go back over the questions you get wrong.



```
21.

 Q: What is the
 capital of
 Switzerland?
                         21.

                          A: The capital
                          of Switzerland
                          is BERNE.
 Your answer:

 ZURICH______

       WRONG!

Quiz: World Geography I
Number correct: 18
Number wrong:    3

SCORE: 72
```

At right is a screen from a program that makes learning to touch-type into a game. The keys are displayed on the screen in colors — keys of the same color are pressed by the same finger. Words randomly appear at the upper left, and you must type them before they disappear at right. Underlining signifies correctly-typed letters; red letters beneath words show the keys you have pressed by mistake. Your score and speed are recorded at the bottom.



```
          TYPING  PRACTICE

SELECT

                          ENTRY
         PLUS

                                      WORD
                                        TE

       1 2 3 45 67 8 9 0+-£
        Q W E RT YU I O P@*↑
        A S D FG HJ K L :;=
        Z X C VB NM , . /



        WORDS CORRECT: 7
        WORDS WRONG:    2
        SPEED (WPM):   32
        SCORE:       3020
```

answers, to let you keep track of your progress. Some allow you to make up your own questions and answers for later review.

A second type of program does more active teaching. These tutorial programs present a sequence of textual information. Some periodically test your understanding of the concepts, and if you answer correctly, they advance and present more information. Tutorial programs can teach the fundamentals of such disciplines as algebra, geography, astronomy or even BASIC, the computer language that your 64 uses.

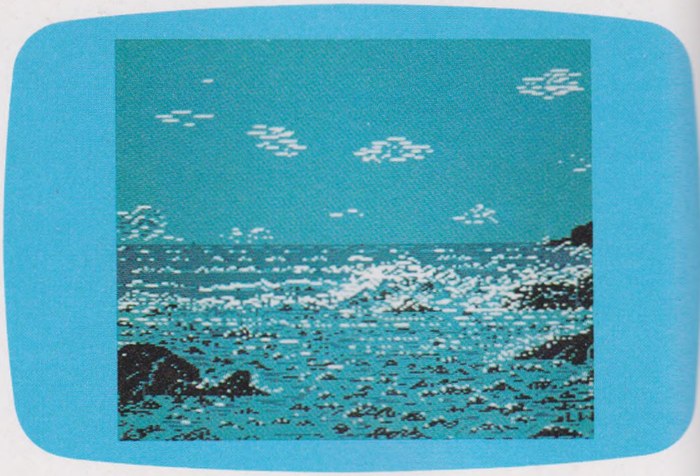
Another, more sophisticated type of program simulates real situations and concomitant problems; you learn by working through the difficulties. This type of software might present the purchasing and pricing problems of a small business or, for children, the problems of distributing equal volumes of liquids into odd-shaped containers.

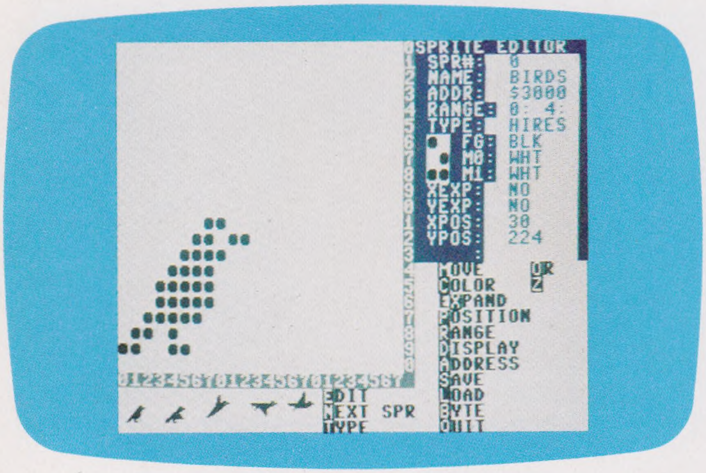
Many good educational programs have been placed in the public domain by their authors. This means they may be copied without explicit permission; many user groups can provide a disk with several programs on it for a nominal fee.

The hallmarks of good educational software, wherever you find it, are clear on-screen instructions, commands that use keys logically, and the graphics, color and sound effects that keep the learner interested.

Creating Art on a Color Monitor

This seascape is an example of the graphics you can create on your Commodore 64 with a multicolor drawing program. A joystick or trackball is used to move the cursor around on the screen; pressing the "fire" button makes the cursor leave a trail of color, using any four of the Commodore's 16 colors. You can even save your artwork on disk or cassette to work on later.





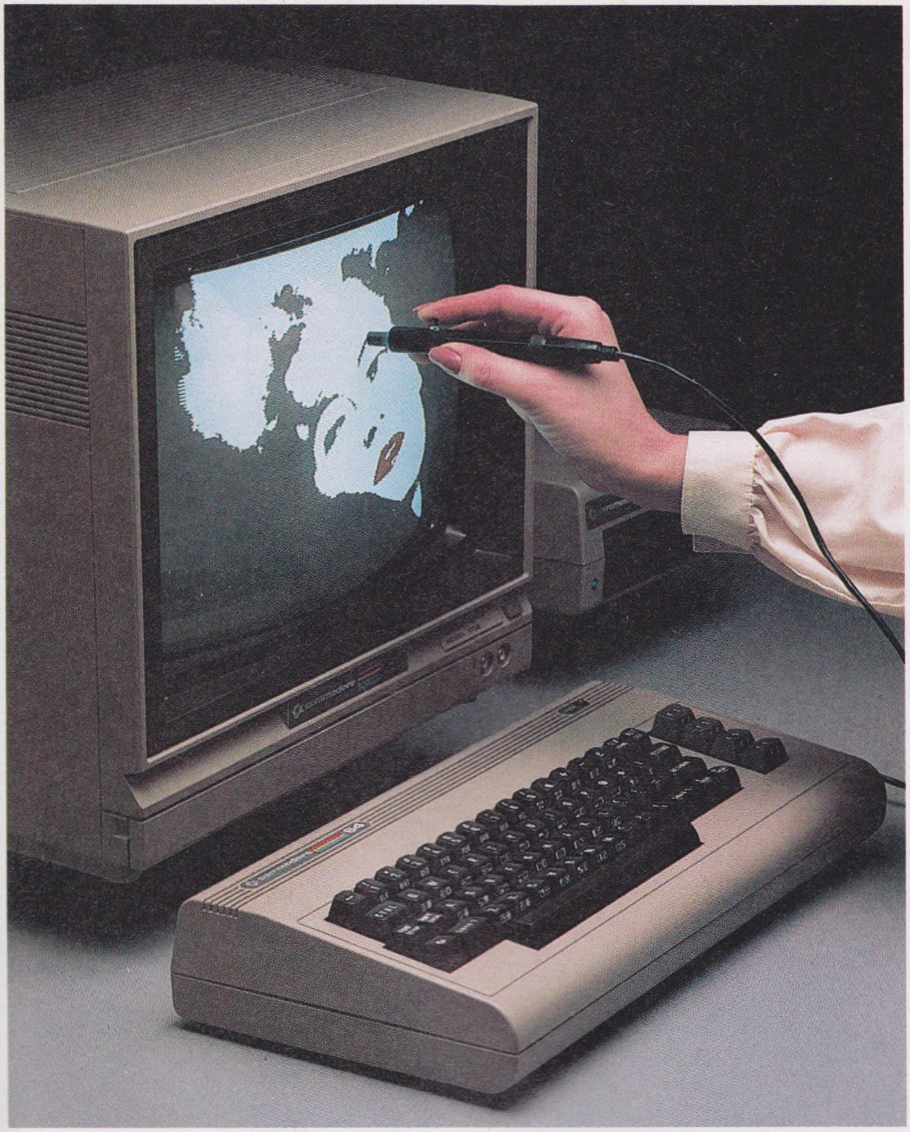
```
SPRITE EDITOR
 SPR#:   0
 NAME:   BIRDS
 ADDR:   $3000
 RANGE:  0: 4:
 TYPE:   HIRES
   FG:   BLK
   M0:   WHT
   M1:   WHT
 XEXP:   NO
 YEXP:   NO
 XPOS:   30
 YPOS:   224
```

A sprite editor provides a working screen similar to the one shown at left. The sprite being edited is shown greatly enlarged; the program lets you shape it by coloring in one pixel — one dot — at a time. The menu of commands is at lower right on the screen, and information on the present set of sprites is at the upper right. Here, a sequence of five sprites is shown at lower left. Switched on and off in sequence, these images make simple animation possible, simulating the takeoff of a bird.

Sixteen colors to choose from, finely detailed ("high-resolution") images, moving figures called sprites, character shapes that can be redefined — these are some of the Commodore 64's powerful graphics features. The features are rooted in the computer's ability to pick out and change the color of a single point on the monitor screen. Each of these points is called a pixel (short for "picture element"); there are 64,000 pixels on a color monitor. The Commodore uses a special color processor called the VIC-II (Video Interface Controller) chip to do the work of translating data inside the computer into images on the screen.

The standard BASIC commands used to instruct the VIC-II chip are quite complicated; they require that you tell the computer to insert numeric data into specific memory locations. Because this is very time-consuming, a considerable variety of software has been devised that does much of the work for you. Images you create with any of this software can be printed out with a dot matrix printer, or saved for reuse or for incorporation into other programs.

The majority of graphics utilities, as these programs are called, are expressly designed for creating images to be used in other programs. With a programmable-character editor, you can create accented letters needed in foreign words, or little "people" for use in a game. A



Some of the newest drawing programs give you the option of doing artwork on the monitor screen using a light pen instead of a joystick or trackball. Interacting with the screen through a light-sensitive cell in its tip, a light pen can put a dot on the screen at any point you desire. This lets you produce finely detailed graphics without spending hours programming your computer.

sprite — a figure about eight times the size of a character — can be created, colored and even animated with the help of a sprite editor.

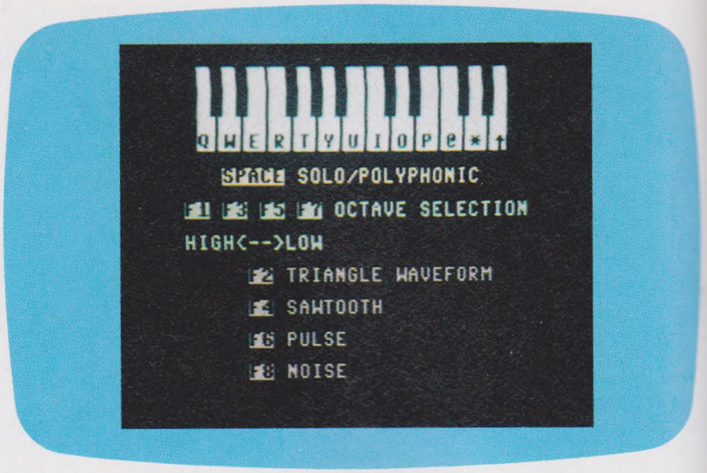
Some programs provide extensions to the BASIC language in the 64. Such an improved BASIC lets you write programs using simplified commands such as **PLOT, DRAW, ROTATE,** and **COLOR;** when you run the program, the graphic image appears on the screen.

Less sophisticated graphics-oriented languages such as LOGO have picture-making capabilities, but their main purpose is educational, helping children learn programming concepts and abstract thinking, as well as geometry.

Drawing and sketchpad programs are the computer's nearest approach to drawing on a piece of paper. This type of software lets you see the results of your actions immediately; they are most effective when used with a control device, such as a joystick or a trackball, that makes it easy to use the cursor like a paintbrush on the screen. Other versatile drawing tools are the light pen *(above)* and the graphics tablet, a flat pad that relays to the computer shapes drawn with a stylus.

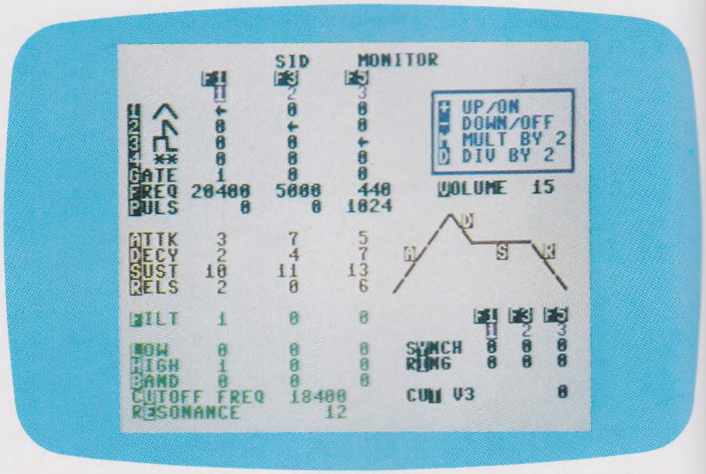
Making Music with the 64

This display is generated by a keyboard program similar to the one provided in the back of the *Commodore 64 User's Guide.* The letters on the white piano keys indicate the keys you press on your computer keyboard to hear the corresponding musical notes, which sound immediately. Other such programs allow you to store your sequence of notes for later playback as well.



```
Q W E R T Y U I O P @ * ↑

     SPACE SOLO/POLYPHONIC

F1 F3 F5 F7 OCTAVE SELECTION

HIGH<-->LOW

          F2 TRIANGLE WAVEFORM

          F4 SAWTOOTH

          F6 PULSE

          F8 NOISE
```

Special-effects generators, such as a SID monitor *(right),* let you take direct control of the Commodore 64's sound-producing chip. By changing the numerical values for each of the three voices and for the sound pattern, or envelope, displayed on the screen, you can tailor musical notes and sound effects for use in your own programs.

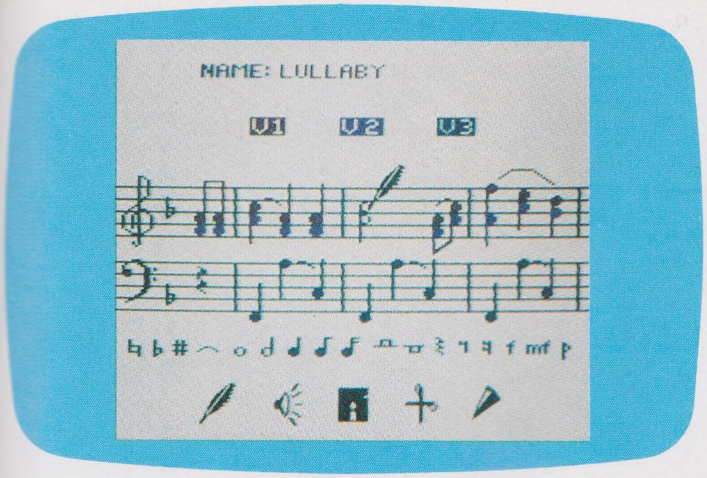


```
                SID      MONITOR
          F1       F3       F5
          1        2        3
1 ^       ←        0        0          + UP/ON
2         0        ←        0          - DOWN/OFF
3         0        0        ←          * MULT BY 2
4  **     0        0        0          D DIV BY 2
GATE      1        0        0
FREQ  20400     5000      440          VOLUME   15
PULS      0        0     1024

ATTK      3        7        5
DECY      2        4        7
SUST     10       11       13
RELS      2        0        6

FILT      1        0        0                  F1  F3  F5
                                               1   2   3
LOW       0        0        0          SYNCH   0   0   0
HIGH      1        0        0          RING    0   0   0
BAND      0        0        0
CUTOFF FREQ    18400                   CUT V3          0
RESONANCE         12
```

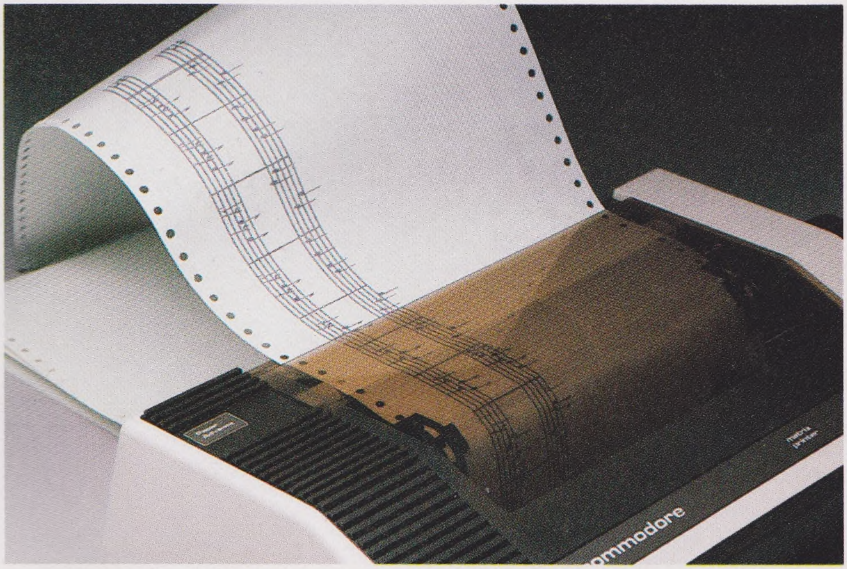
Sound, like graphics, is an area where the Commodore 64 excels. You can compose tunes, imitate the sounds of a wide variety of musical instruments, add background music to a game program or synthesize your own sound effects.

The 64's sound output, like the graphics, is controlled by a special chip, this one called SID (Sound Interface Device). The SID chip has three separate sound generators, so the computer can produce three different "voices." It also lets you choose from among four different waveforms — the patterns of sound vibrations. By combining these attributes, you can use the 64 as a music synthesizer of great versatility.

Like the VIC-II chip that controls video output, the SID chip requires complex instructions to do its work; these instructions are difficult to program in BASIC, the 64's native language. But a growing body of software makes programming sound



A typical music composition program lets you compose music — or copy a piece of sheet music — by placing musical notation on the staff. The notation is represented by small symbols called icons, which you manipulate with a joystick. The program interprets the notation and plays the composition back; it may also scroll the music across the screen as it plays.



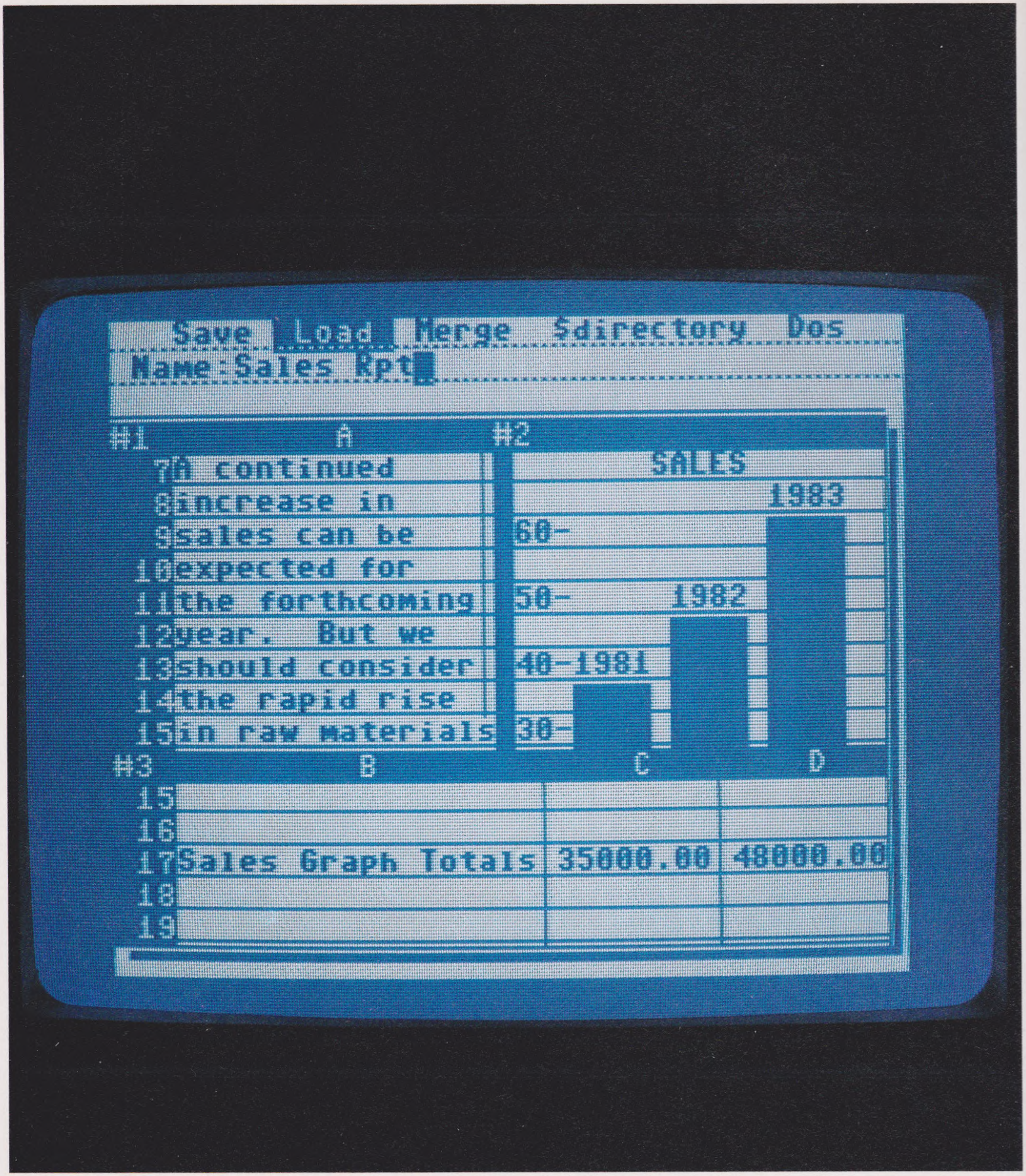
The best music composition packages allow you to use a dot matrix printer to produce hard copies of your musical scores, as well as saving your music on disk. You can thus print multiple copies of a particular piece of music to play later.

and music on the 64 easier, for novice and expert alike.

Much of the software designed to let you compose music offers a variety of features. Some programs make the process relatively simple, letting you compose by moving notes onto a musical staff on the screen; you manipulate the notes and other musical notation with keyboard commands, joystick or light pen. More difficult to use but more powerful are packages using a special music-programming language; you write a detailed statement for each note of the music, as if you were writing a BASIC program.

Other music programs turn your computer into a piano; notes sound when you press keys on the computer keyboard *(opposite, top)*. You can play notes in eight octaves, choosing from among four waveforms.

Sound-programming utilities give you direct control over every characteristic — waveform, frequency and many others — of a sound you want to produce. You can experiment not only with musical sounds but also with the nonmusical: sirens, thunder, explosions. When you have the right sound, you record the numerical value of each of the sound's characteristics and save the synthesized sound for use in games or other programs.



```
Name:Sales Rpt
```

```
#1          A                  #2
 7A continued                              SALES
 8increase in                                     1983
 9sales can be
10expected for
11the forthcoming
12year.  But we
13should consider
14the rapid rise
15in raw materials
```

```
#3             B                C           D
15
16
17Sales Graph Totals  35000.00  48000.00
18
19
```

Business Software: Efficiency in the Office

Well-designed application software packages are the tools that make your Commodore 64 the centerpiece of a highly efficient small office. Word processing programs can turn your computer into a supertypewriter and you into a speedier typist, a versatile editor and a champion speller. Electronic spreadsheets — programs patterned after the ruled ledger sheets used by accountants — let you ask "what-if" financial questions. Once you have entered the mathematical relationships between various financial elements into the spreadsheet, you can experiment with alternate figures: The program will make all the recalculations for you. Some spreadsheet programs also let you turn numbers into graphics, cutting the time needed to prepare artist-quality charts and graphs from days and weeks down to hours or even minutes. Data base management systems can help you store and keep track of entire file cabinets full of information — and find any given item in a trice.

Such software is enormously popular, and new offerings regularly appear on the market. For each type of application, products can range from the basic to the elaborate — with price tags to match. Finding the software that is best suited to your needs takes a bit of research on your part.

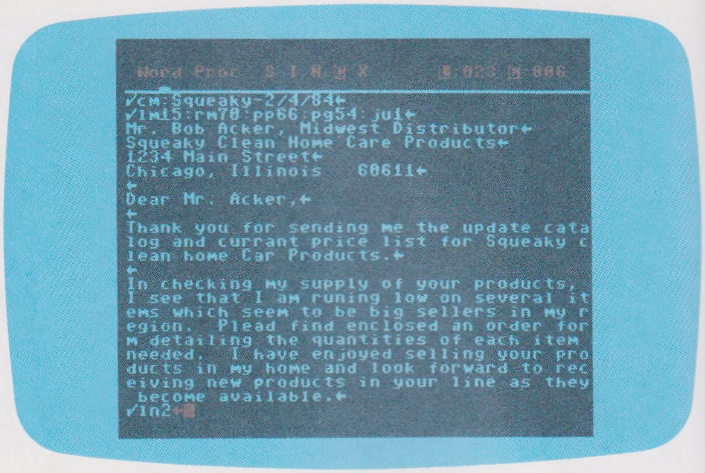
One of the more important aspects of shopping for software is determining how a potential purchase will interact with other application programs. There are, for example, word processing programs that will let you insert information from your data base or spreadsheet, and vice versa. There are also multifunction programs (called integrated software) that offer several applications within one package, each application interactive with the others. But beware of the inflated claims that accompany some programs; sophisticated features that are part of an individual program may be omitted to save memory space in an integrated package.

Equally important is whether the software is easy to learn and use. Pay close attention to the documentation, or written material, that comes with the program, and to on-screen aids, if any. Good software requires thousands of hours of programming time to create, but you should not have to spend thousands of hours mastering its commands and options. See the software demonstrated and try it out yourself before purchase.

The integrated software package shown on the screen at left allows the user to pause while doing financial calculations to call up data files; the different jobs appear in separate windows on the monitor. This package also incorporates a graphics function that simultaneously displays worksheet figures as line or bar graphs.

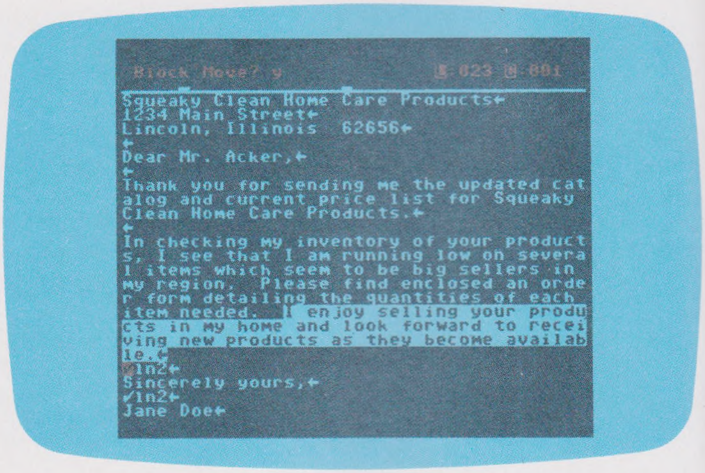
Writing and Editing Electronically

The screen at right illustrates some of the features of a simple word processing program, used here to compose a letter. The top line on the screen indicates the mode in which the program is working, and the position of the cursor. A check mark at the beginning of a line indicates formatting codes for the printer. Words that are broken at the ends of lines will appear intact when the letter is printed. You correct spelling and punctuation errors on the screen without retyping the whole letter.



```
Word Proc  S I N ■ X          ■:023 ■:006
✓cm:Squeaky-2/4/84←
✓lm15:rm70:pp66:pg54:jui←
Mr. Bob Acker, Midwest Distributor←
Squeaky Clean Home Care Products←
1234 Main Street←
Chicago, Illinois   60611←
←
Dear Mr. Acker,←
←
Thank you for sending me the update cata
log and currant price list for Squeaky c
lean home Car Products.←
←
In checking my supply of your products,
I see that I am runing low on several it
ems which seem to be big sellers in my r
egion.  Plead find enclosed an order for
m detailing the quantities of each item
needed.  I have enjoyed selling your pro
ducts in my home and look forward to rec
eiving new products in your line as they
 become available.←
✓ln2←
```

In the example at right, the letter has been moved up on the screen and spelling errors have been corrected. The block of text highlighted in white has been marked for moving, using a sequence of commands entered at the keyboard. The block will be moved to the position occupied by the cursor at the bottom of the screen.



```
Block Move? y                 ■:023 ■:001
Squeaky Clean Home Care Products←
1234 Main Street←
Lincoln, Illinois  62656←
←
Dear Mr. Acker,←
←
Thank you for sending me the updated cat
alog and current price list for Squeaky
Clean Home Care Products.←
←
In checking my inventory of your product
s, I see that I am running low on severa
l items which seem to be big sellers in
my region.  Please find enclosed an orde
r form detailing the quantities of each
item needed.  I enjoy selling your produ
cts in my home and look forward to recei
ving new products as they become availab
le.←
■ln2←
Sincerely yours,←
✓ln2←
Jane Doe←
```

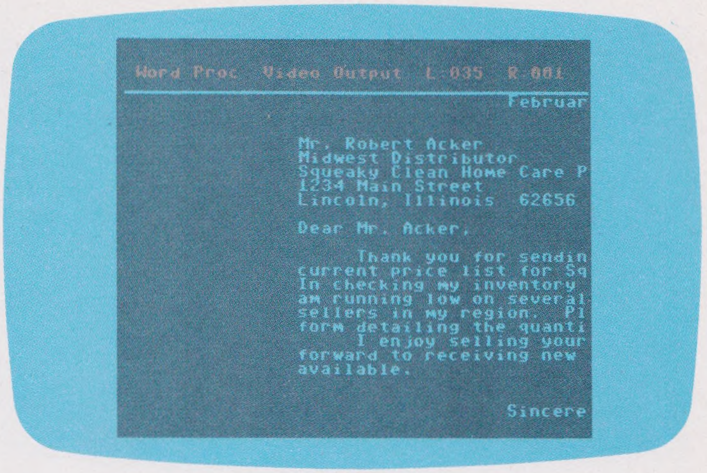
One of the most common office-related uses for the Commodore 64 is as an electronic typewriter. You can combine your 64 with word processing software and a printer to create documents as simple as a shopping list, as complex as a business report, or as long as a book.

The simplest programs for writing and changing documents are called text editors. While they are easy to learn and use, they are limited: You can make changes on only one line of text at a time. More sophisticated programs, known as word processors, treat an entire document — letter or book chapter — as a unit, ana let you make complex changes such as moving paragraphs or pages.

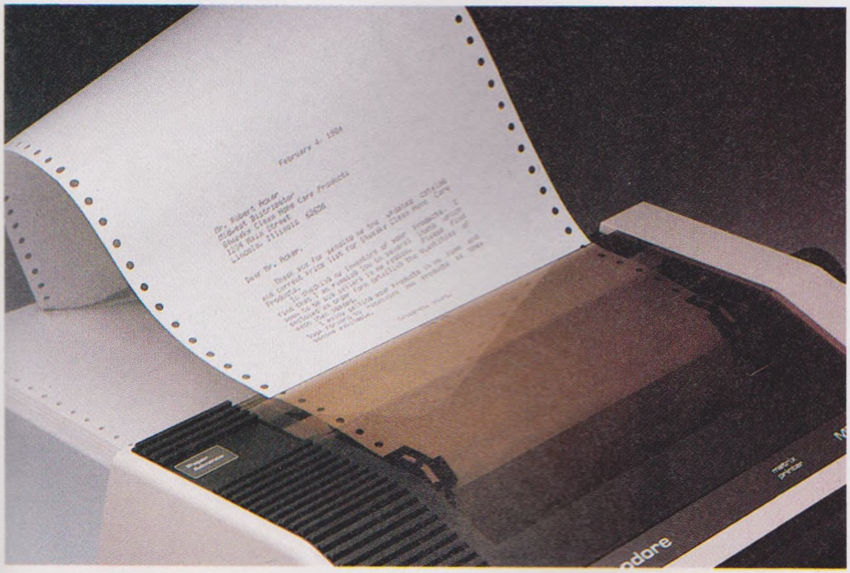
If you will be writing only a few documents, and they will be simple ones such as memos, you can save money with a no-frills text editing program. Most such programs have the basic features of a word processor, including word-wrap, an automatic carriage return that moves long words to the next line. Even the most basic program should allow you to do simple editing (inserting and deleting material anywhere on the screen), formatting (automatically setting margins, justification and tabs), and storing and retrieving text. Be sure the program leaves enough memory space for the amount of text you normally work with.

Somewhat more complex programs let you delete any part of the

The screen at right shows part of the letter as it will appear when printed on the page after all changes are made. Because the Commodore 64 can ordinarily display only 40 characters on a line, this "Video Output" preview can show only part of the wider letter at a time; a few keystrokes will move, or scroll, the letter sideways across the screen, bringing the right side into view.



```
Word Proc  Video Output  L:035  R:001
──────────────────────────────────────
                               Februar

                 Mr. Robert Acker
                 Midwest Distributor
                 Squeaky Clean Home Care P
                 1234 Main Street
                 Lincoln, Illinois  62656

                 Dear Mr. Acker,

                      Thank you for sendin
                 current price list for Sq
                 In checking my inventory
                 am running low on several
                 sellers in my region.  Pl
                 form detailing the quanti
                      I enjoy selling your
                 forward to receiving new
                 available.

                                  Sincere
```



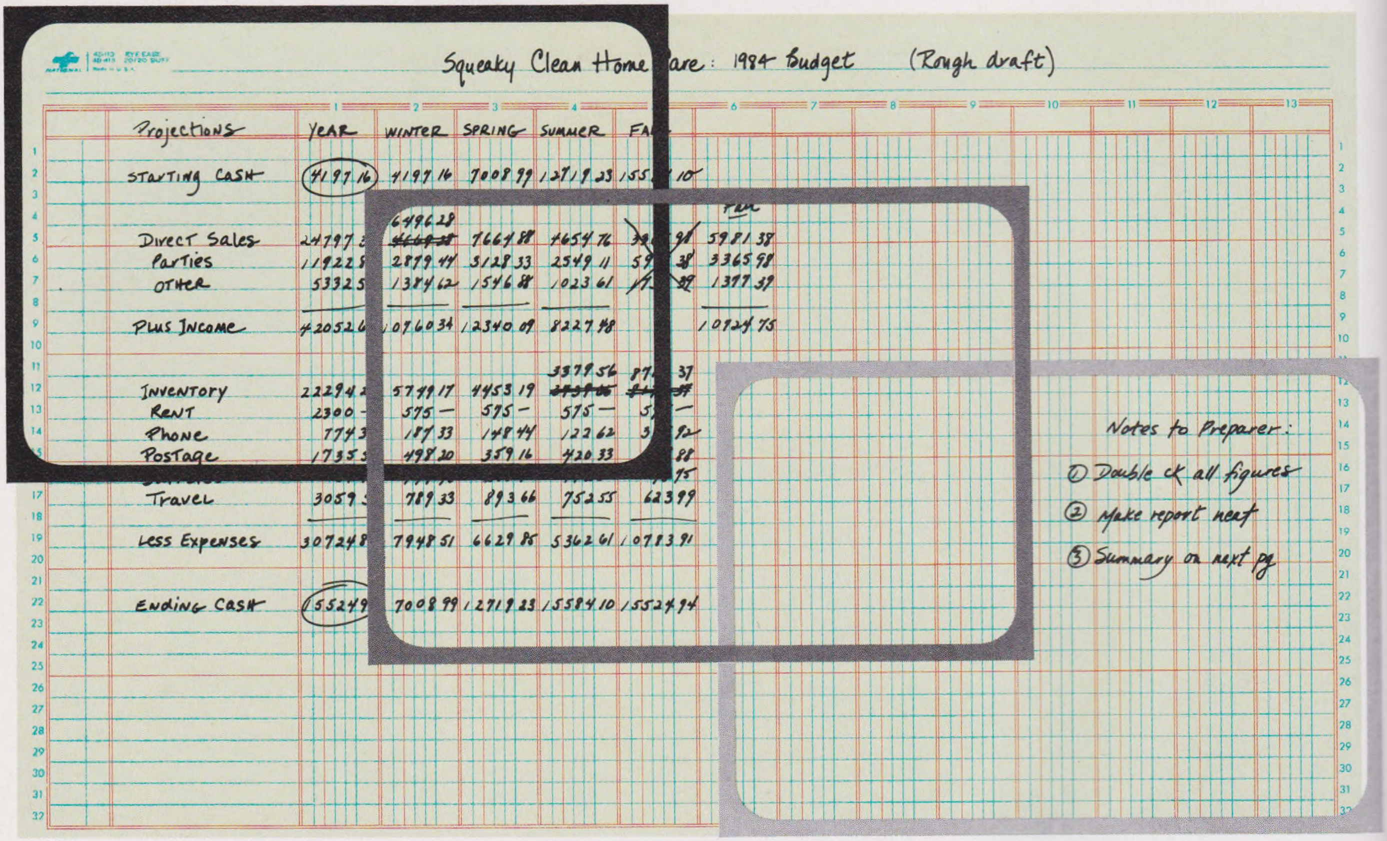
The final draft of the letter emerges from the printer, all its elements proofread and correctly placed. Some printers can produce documents on individual sheets of paper, so you can use your printed letterhead; they can also print on small objects such as envelopes, so you can write and address a letter to an entire mailing list without typing each one individually.

text, or move it to another place in the document. Some programs can check your spelling, search an entire document for a word or phrase and replace one word with another wherever it appears.

To turn out sophisticated reports complete with page headings, footnotes and aligned tables, you will require a more advanced program. The best programs also let you send form letters by merging an address list with a letter file. If you plan to use your 64 regularly for word processing, you can overcome the 40-column screen limit by buying software with an 80-column option, or by equipping your 64 with a special cartridge called an 80-column board. Either way, an optional monochrome monitor provides the clearest image of the smaller characters. If format is important, look for a program that lets you see on the monitor exactly what will be printed.

Keep in mind that elaborate programs may be difficult to master. When you choose a program, look for a clearly written manual, and notice the on-screen aids or menus; these can be so clear that they practically lead you by the hand — or they can be cryptic and confusing. Look for a program that combines the features you need with simplicity and ease of understanding. And finally, be sure that the word processing program you choose will work with your printer and printer interface.

Financial Wizardry with Spreadsheets



Squeaky Clean Home are: 1984 Budget (Rough draft)

Projections	YEAR	WINTER	SPRING	SUMMER	FA	Fall
STARTING CASH	4197 16	4197 16	7008 99	12719 23	155 10	
Direct Sales	24797 3	~~4469 28~~ 6496 28	7664 88	4654 76	39 98	5981 38
Parties	11922 8	2879 44	3128 33	2549 11	59 38	3365 98
OTHER	53325	1384 62	1546 88	1023 61	1 39	1377 39
PLUS INCOME	42052 6	10760 34	12340 09	8227 48		10724 75
INVENTORY	22294 2	5749 17	4453 19	~~2939 05~~ 3379 56	~~8 37~~ 87 37	
RENT	2300 -	575 —	575 -	575 —	5 —	
Phone	774 3	187 33	148 44	122 62	3 92	
Postage	1735 5	498 20	359 16	420 33	88	
					75	
Travel	3059	789 33	893 66	752 55	623 99	
Less Expenses	30724 8	7948 51	6629 85	5362 61	10783 91	
Ending Cash	15524 9	7008 99	12719 23	15584 10	15524 94	

Notes to Preparer:

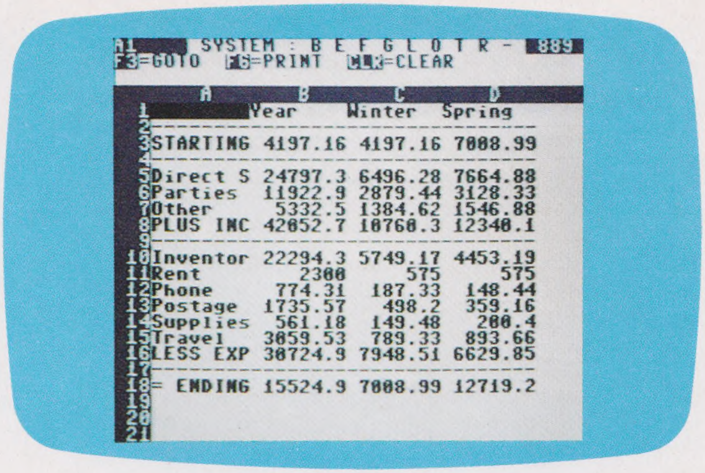
1. Double ck all figures
2. Make report neat
3. Summary on next pg

Computer screens overlaid on a paper worksheet *(above)* illustrate the window effect of a spreadsheet program. Unlike a paper version, the electronic model can have hundreds of columns and rows without growing impossibly unwieldy. By scrolling or by using direct **GOTO** commands, you can call any section of the worksheet into view on your screen.

A spreadsheet program lets you build a model of the accountant's traditional worksheet on your computer screen. You can use such a program to prepare budgets, to record business and household expenses, to compute taxes, to manage your stock portfolio and to assist you in financial planning.

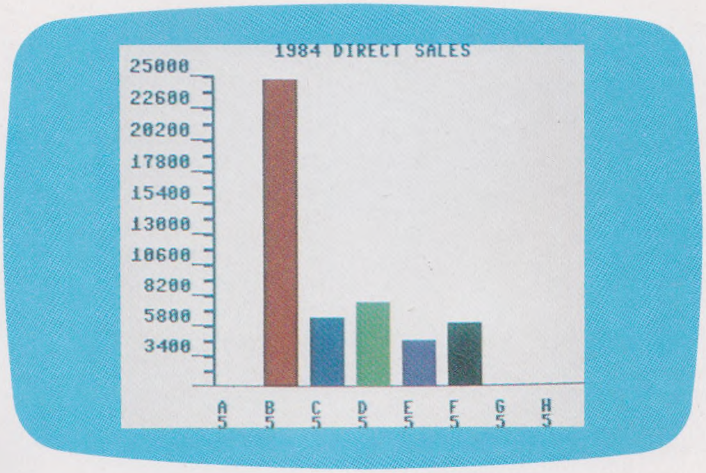
Like its paper counterpart, the electronic model is essentially a blank page divided into rows (usually numbered, from top to bottom) and columns (usually lettered, from left to right). Columns and rows intersect to form cells, which are identified by their coordinates. Because it isn't limited by the size of a page, an electronic spreadsheet may have more than 10,000 separate cells. The cells can hold different types of information: In some, you will put text, such as labels for columns and rows; in others, numbers; and in still others, instructions, in the form of mathematical formulas.

A typical spreadsheet screen *(right)* uses one or more lines at the top for display of housekeeping information. Here the top line identifies the cell position of the cursor, and lists letters standing for menu choices. The second line and the left column are letters and numbers used to pinpoint cell locations: For instance, direct-sales figures for winter are in cell C5. The cursor highlights the cell where it is positioned.



```
A1           SYSTEM : B E F G L O T R -  889
F3=GOTO  F6=PRINT  CLR=CLEAR

         A          B         C          D
 1                Year     Winter    Spring
 2 ----------------------------------------
 3STARTING  4197.16  4197.16  7008.99
 4 ----------------------------------------
 5Direct S  24797.3  6496.28  7664.88
 6Parties   11922.9  2879.44  3128.33
 7Other      5332.5  1384.62  1546.88
 8PLUS INC  42052.7  10760.3  12340.1
 9 ----------------------------------------
10Inventor  22294.3  5749.17  4453.19
11Rent         2300      575      575
12Phone      774.31   187.33   148.44
13Postage   1735.57    498.2   359.16
14Supplies   561.18   149.48    200.4
15Travel    3059.53   789.33   893.66
16LESS EXP  30724.9  7948.51  6629.85
17 ----------------------------------------
18= ENDING  15524.9  7008.99  12719.2
19
20
21
```

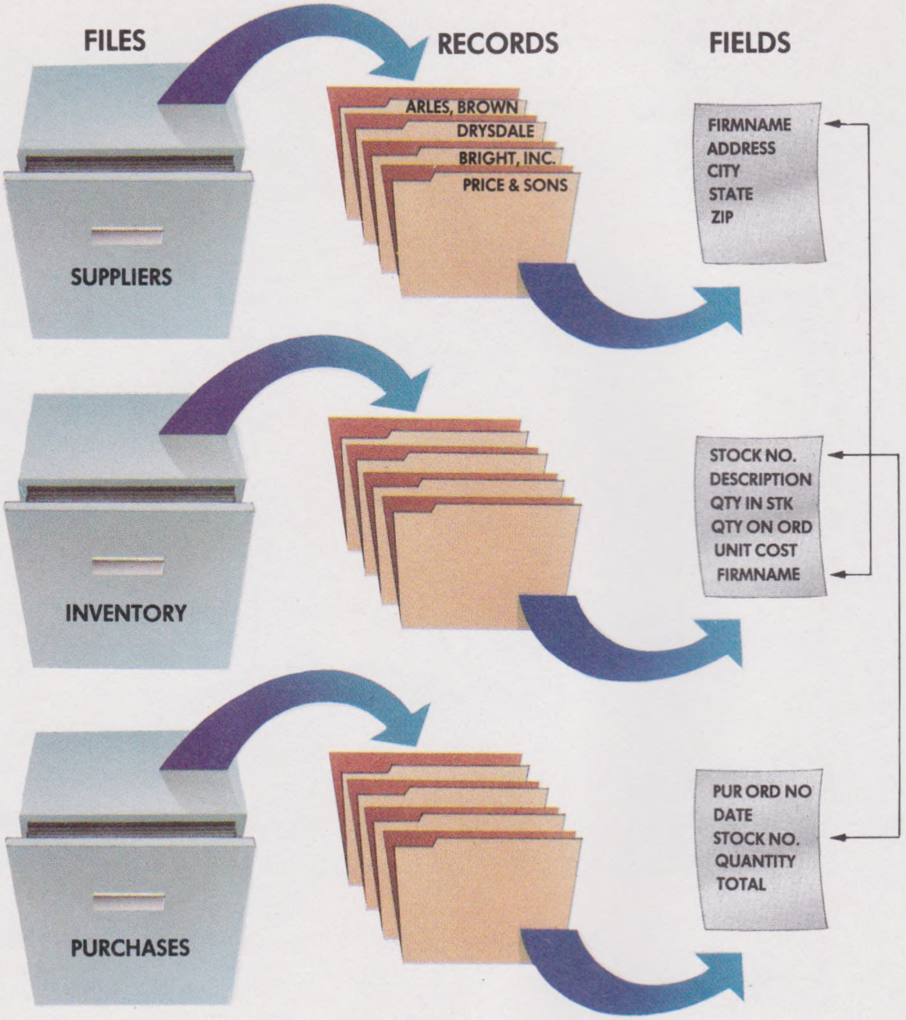
If a spreadsheet package incorporates a graphics program, you can present spreadsheet information visually with bar graphs like the one shown here. When you change entries on the spreadsheet, the associated charts will be adjusted automatically.



Formulas are the magic wands of spreadsheet programs. They can be simple, such as an instruction to add all the numbers in a certain range of cells and put the total in another cell; or complex, such as an instruction to perform a given calculation *if* a specified condition is present. Once the formula is entered, any changes in the cells it uses in its computations automatically produce changes in the values calculated by the formula. With a spreadsheet program, such a change takes only a few seconds, instead of the hours that might be required if the sheet recalculations were done by hand. This means you can try different "what-if" situations, to see how variations in one area, such as unit price or monthly income, would affect the numbers in the rest of the spreadsheet.

A spreadsheet can be used for more than budgeting and planning; it is also a kind of visible data base, since you can view a wide variety of data quickly and easily, by simply scrolling through the data entered in its cells. Some spreadsheet packages may include simple graphics programs as well, to give you output that is more easily understood than rows and rows of numbers.

The Organizing Power of Data Base Systems



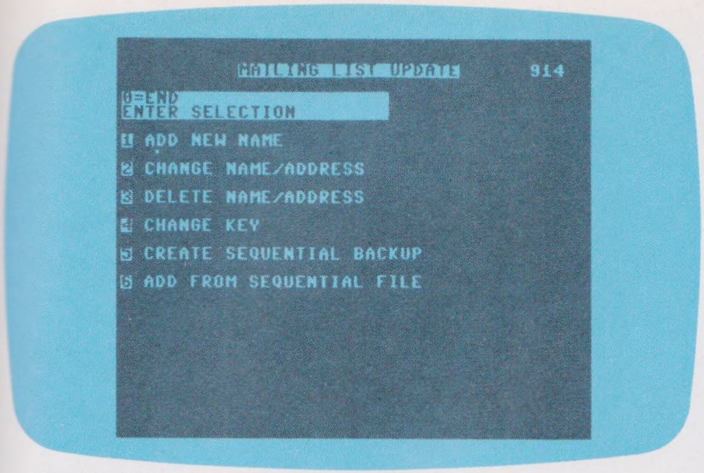
Data base managers help you organize large quantities of information for storage on, and quick retrieval from, cassette or disk. The information is stored in data files, subdivided into records and fields. A record is the electronic equivalent of a file folder, holding several items — or fields — of related information. All records in a file have the same fields, holding different information. By designating one field as the key, you can arrange records in alphabetical or numerical order, or tell the computer to search other files for records containing an identical field, as illustrated.

Data base management programs, sometimes called data managers, turn your 64 into an electronic filing cabinet. A data manager lets you store large quantities of information — such as mailing lists, personnel records or inventory data — on cassette or disk. You can then retrieve information easily and organize it in any order you choose.

As the starting point for the use of a data manager, you must enter the information in short elements called fields. Fields are arranged into records; the program then does its work by sorting through the contents of fields and using what it finds there to put records into a given order within a file. Ordinary data managers have limits of 80 characters in each field and no more than 254 characters in each record.

Data base managers can be distinguished by file type, sequential or relative. The simplest programs use sequential files, storing data in a specified order. While sequential files are easy to handle, information retrieval is slow, since the computer must sift through the entire file until it finds the record you need. Sequential files can be stored on a cassette and are best suited to small files — fewer than 100 records.

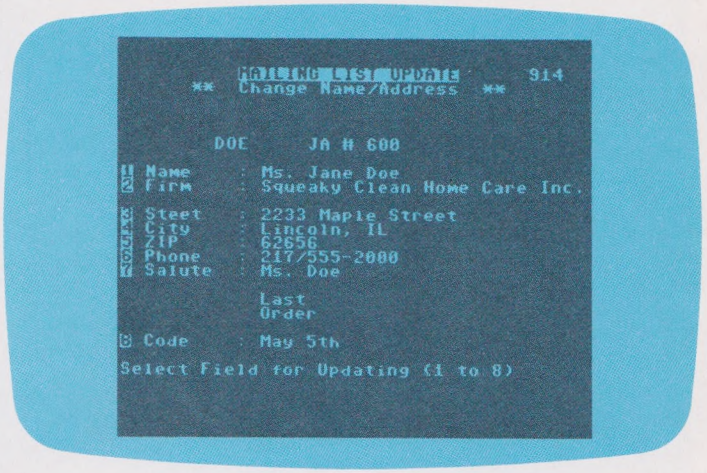
Relative files are better suited to larger data bases, and require at least one disk drive for storage. Every record in a relative file is the same length, up to 254 characters (al-



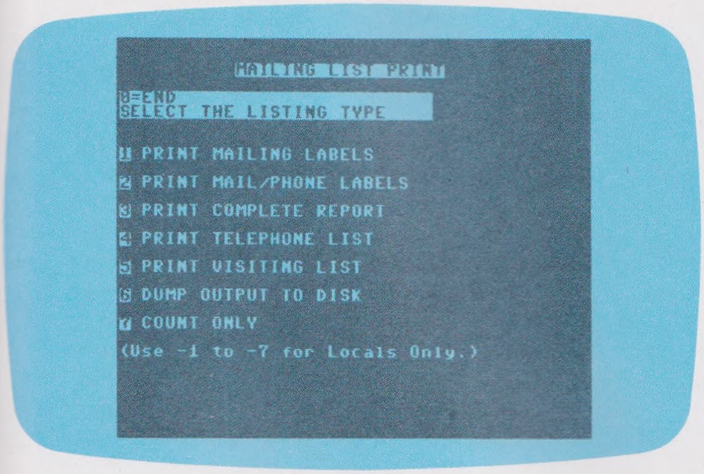
```
          MAILING LIST UPDATE        914
0=END
ENTER SELECTION
1 ADD NEW NAME
2 CHANGE NAME/ADDRESS
3 DELETE NAME/ADDRESS
4 CHANGE KEY
5 CREATE SEQUENTIAL BACKUP
6 ADD FROM SEQUENTIAL FILE
```

The first three options on this data base menu allow changes to records and fields; the fourth changes the field that the program looks at when sorting records. The last two options let you store data in its current sequence or add an item from another file.

This sample record from a mailing list includes eight fields, each with a specific type of information about an individual. Use of separate fields allows the file to be sorted and reorganized easily; for instance, a zip code field allows records to be placed with their zip codes in numerical order.



```
           MAILING LIST UPDATE      914
      **  Change Name/Address  **

        DOE      JA # 600

1 Name     : Ms. Jane Doe
2 Firm     : Squeaky Clean Home Care Inc.

3 Steet    : 2233 Maple Street
4 City     : Lincoln, IL
5 ZIP      : 62656
6 Phone    : 217/555-2000
7 Salute   : Ms. Doe

             Last
             Order
8 Code     : May 5th

Select Field for Updating (1 to 8)
```



```
          MAILING LIST PRINT
0=END
SELECT THE LISTING TYPE

1 PRINT MAILING LABELS
2 PRINT MAIL/PHONE LABELS
3 PRINT COMPLETE REPORT
4 PRINT TELEPHONE LIST
5 PRINT VISITING LIST
6 DUMP OUTPUT TO DISK
7 COUNT ONLY
(Use -1 to -7 for Locals Only.)
```

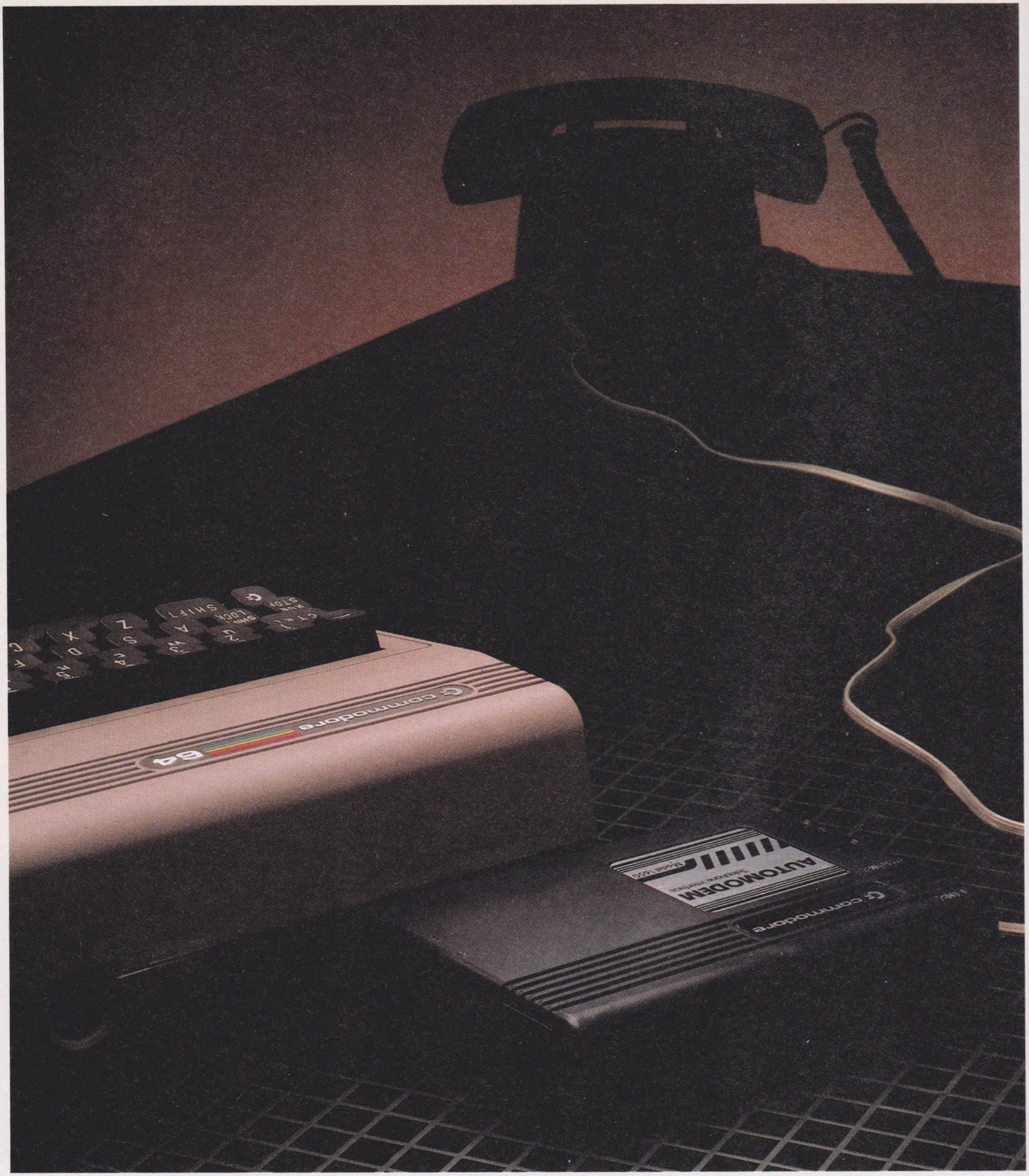
A good data base manager allows you to retrieve and print information in a variety of ways: as tractor-fed address labels, individual envelopes or index cards, or as reports or lists on single sheets of paper. A built-in word processor, or compatibility with a separate word processor, lets you format the data base output in any manner you desire.

though field length can vary). This design allows quick retrieval: Rather than sorting through the entire file to find a particular record, a program using relative files lets you go straight to that record.

Perhaps the most important feature in a data manager is the ability to restructure files. No matter how carefully you design a file, you will eventually want to add, delete, move or change a record or field in some way. Good systems make this simple to do; some don't allow for it at all.

The best data base managers have unlimited field and record lengths, and can store data on more than one disk. These programs may also allow you to use data from two or more files simultaneously. There are other features to look for: the ability to handle lower-case letters and punctuation marks, and to do rapid sorting and searching.

A less vital but very useful feature is programmability — which lets you automate a frequently used command. Some packages allow you to preprogram an entire search or sort routine, so that jobs you do regularly can be implemented with a few keystrokes, rather than an entire set of complicated commands.


commodore
64
AUTOMODEM
Model 1600
commodore

Computer Communications

The potential of your 64 is greatly increased by its ability to "talk" to other computers, using the same telephone network that you ordinarily use for voice communications. Development of the technology that adapts telephone lines for rapid data transfer has paralleled the spread of computer use.

In practice, this means that your 64 can be linked to many different types of computers, perhaps thousands of miles away. And the 64 can draw on the special capabilities of those remote computers to do jobs it could not do on its own. For instance, it can communicate with large mainframe computers that control huge data bases, which you can read and use for your own purposes. You can do research in a specialized data base such as a computerized law library or an electronic newspaper.

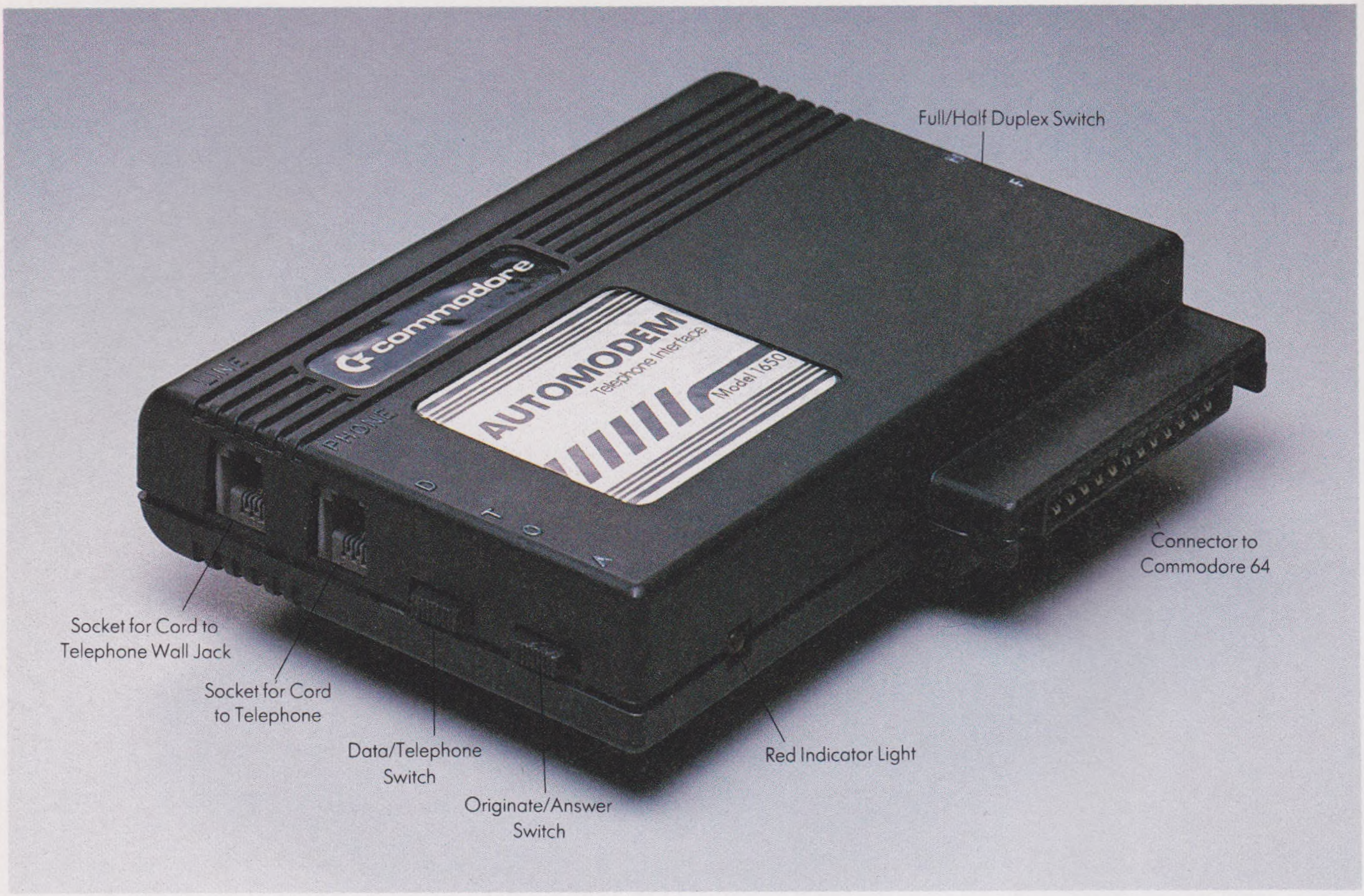
An easy way to get access to a variety of data bases is to subscribe to a commercial information service, a computer network that draws on a number of specialized data bases. After making the phone link with the service, you can electronically leaf through lists of options until you reach the subject you want to work on, then track down the specific items that interest you. You can even set up your computer to record this information, as hard copy from your printer or as a disk file.

Research data bases aren't the only offerings of an information service. Some have a wide variety of features, from movie reviews to constantly updated news services. In some cities, you may even be able to find restaurant menus and theater schedules through an information service. And several services have facilities that let subscribers communicate with each other, by notices posted on electronic "bulletin boards," by individually addressed messages, or even by direct conversations, where words entered at the keyboard appear on a monitor hundreds or thousands of miles away.

But you don't need an information service to conduct direct communications. Any two personal computers can be set up to talk to each other over the telephone lines; you can send reports from office to office, or electronic mail from city to city. Some computer owners even use their equipment to set up small-scale bulletin boards. Whatever the aim of your communications, you will find a vast pool of computerized information available to you — quickly and easily.

A modem like the one shown at left is essential to the process of communicating with another computer. Electronic impulses from the computer are channeled through the modem and translated into tones that can be transmitted over the telephone lines.

Translating with a Modem



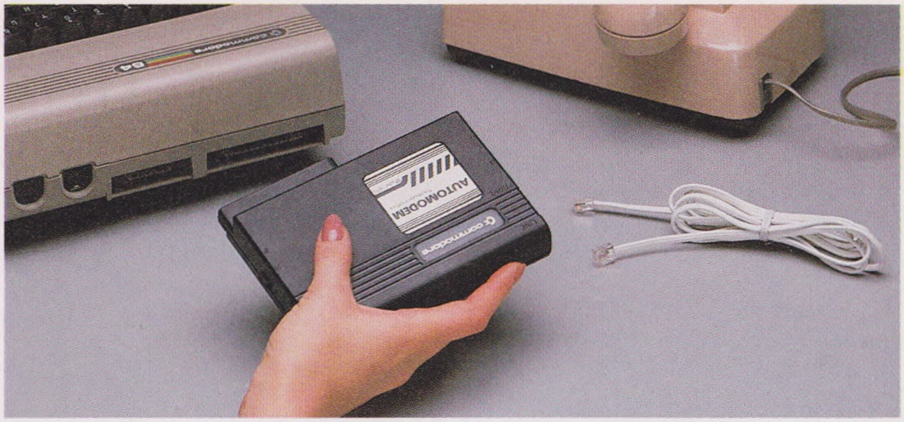
Switches on the Commodore 1650 Automodem *(above)* allow you to set it up for different uses. The DATA/TELEPHONE switch lets you use the phone line for data transfer or for calls without disconnecting the modem. The DUPLEX switch selects data flow in one direction at a time (half duplex) or in both directions simultaneously (full duplex). The ANSWER/ORIGINATE switch sets the modem to place or receive a call. The red light indicates that a link is established.

A conversation between two people is based on unspoken rules that make it possible for them to understand each other. The rules include using speech as their means of communication, speaking in the same language, and taking turns doing the talking and listening.

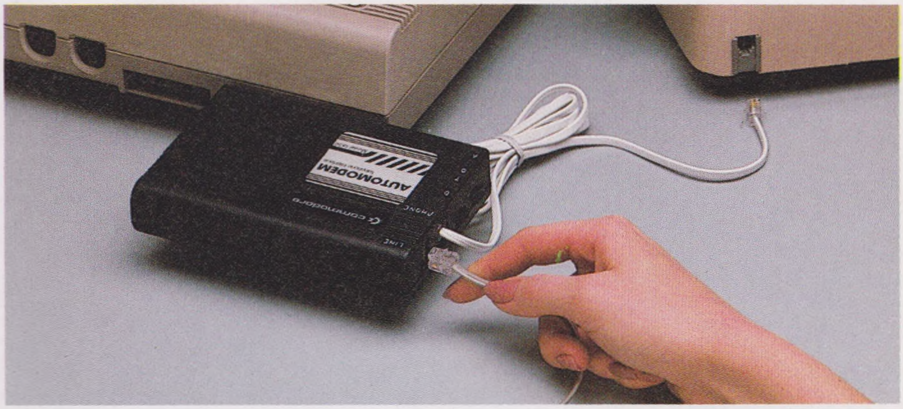
Computers must make use of similar rules and a common language to be able to communicate. But because computers are much less flexible than people are, their rules must be more precise; computers cannot use slang or gestures to get a point across. So computer manufacturers have developed a set of rigid standards, governing everything from the meanings of electrical codes to the shapes of cable connectors, to ensure that data transfer can be conducted without any garbling of messages or damage to equipment.

The C-64 does not come equipped to be hooked into telephone lines. For that, you must add a piece of equipment called a modem (short for "**mo**dulator/**dem**odulator"); it does the necessary translation between the electronic pulses understood by the computer and the signals understood by the telephone system. Modems made by manufacturers other than Commodore require an extra item of hardware called an interface to make them compatible with the C-64.

Your 64 can use an acoustic modem or a direct-connect modem.



Plug the 1650 Automodem into the user port of the Commodore 64 *(above)*. Through this connection, the computer provides electrical power to the modem and also transfers data; the data is converted by the modem into signals that can be sent through the telephone lines.



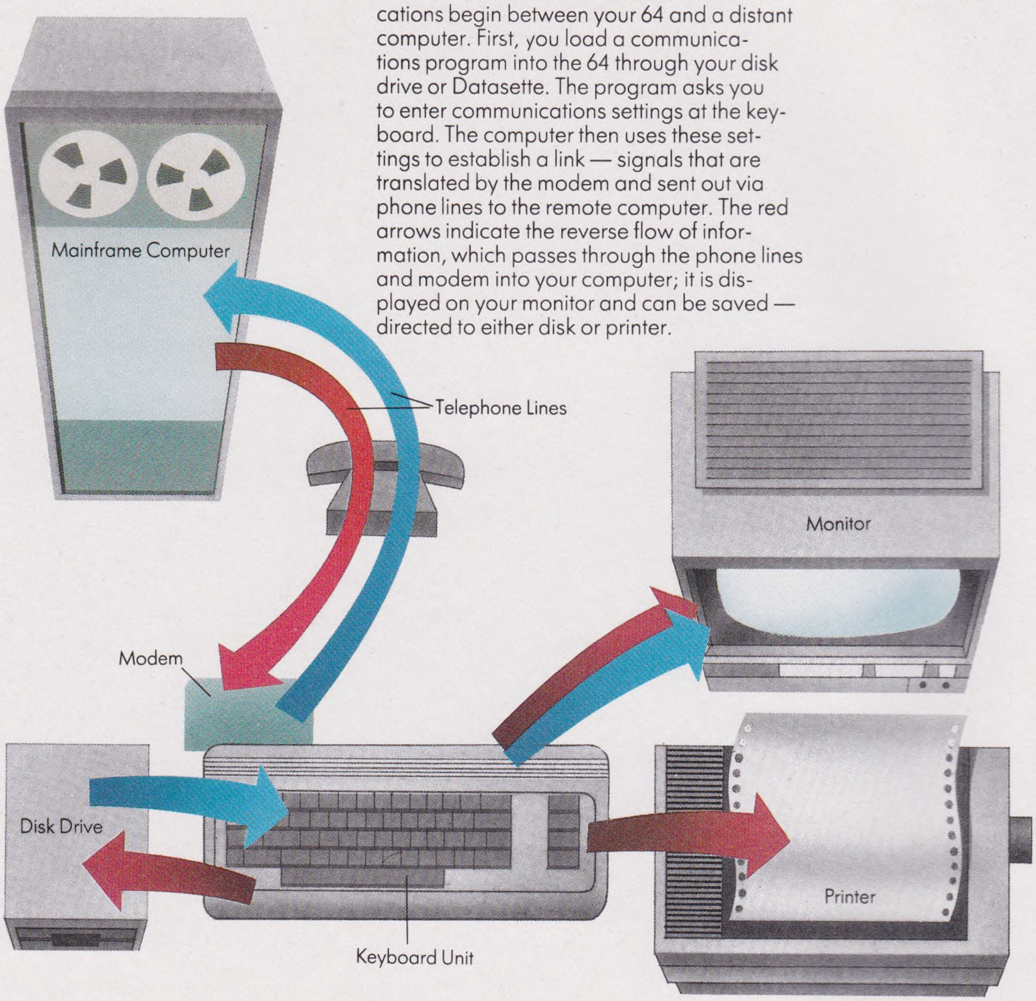
Find the cord that connects the telephone and the wall jack; unplug it from the back of the phone and connect that end to the socket on the modem marked LINE. Take the cord supplied with the modem and plug one end into the back of the phone; plug the other end into the socket on the modem marked PHONE.

Acoustic modems use the telephone handset as part of the communications link; the earpiece and mouthpiece fit tightly into rubber cups on the modem. Direct-connect modems skip the handset and plug directly into the body of the telephone or the wall jack. They are generally more expensive than acoustic modems, but they are also more versatile; many have such features as automatic dialing and answering. Direct-connect modems also eliminate the possibility of room noise interfering with the transfer of data.

The modem Commodore manufactures for the 64 is a direct-connect model, the Commodore 1650 Automodem, shown above. It plugs directly into the user port (no interface is needed), and comes with the cord required to connect it to the telephone system. This modem is capable of dialing or answering the telephone automatically.

You will have to set some switches on your modem before you begin communications. These switches vary from type to type; be sure to check the manufacturer's instructions to determine the settings appropriate to your use.

Making the Link with Software



The blue arrows here show how communications begin between your 64 and a distant computer. First, you load a communications program into the 64 through your disk drive or Datasette. The program asks you to enter communications settings at the keyboard. The computer then uses these settings to establish a link — signals that are translated by the modem and sent out via phone lines to the remote computer. The red arrows indicate the reverse flow of information, which passes through the phone lines and modem into your computer; it is displayed on your monitor and can be saved — directed to either disk or printer.

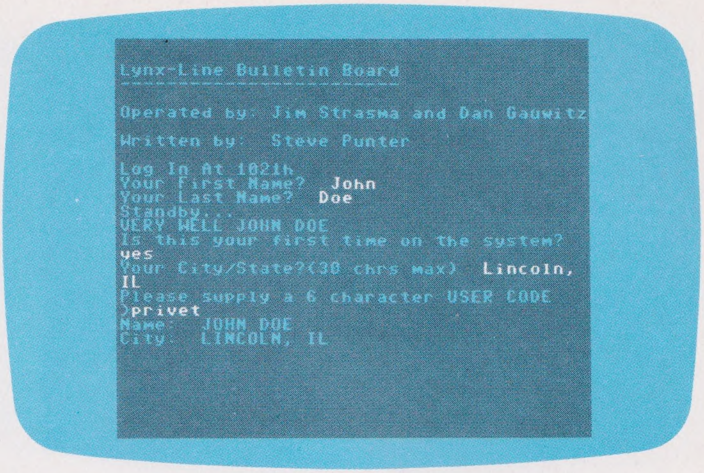
Even with the modem in place, your 64 can communicate with other computers only when it has the proper instructions, contained in a communications program, sometimes called terminal software. This software tells the 64 how to organize its internal operations and how to set up an external link. It effectively turns your 64 into a terminal for passing information to and from a distant computer, and directs the communications process.

The communications software enables your 64 to play a variety of roles. The computer can send and receive messages, either through a bulletin board or via a direct link with another computer. You can have the computer store incoming information, such as a program, on a disk for later printing or other use; this process is called downloading. You can also upload, sending information you have stored on a disk or a cassette to a remote system. Electronic information transfer can be faster than the mail and more economical than messenger services. And by gaining access to the huge memory of a mainframe computer, you can use programs — and games — too large for the memory of your 64.

In order to communicate with another computer, your communications program must determine and set certain parameters. Many communications programs take care of these details for you; some may re-

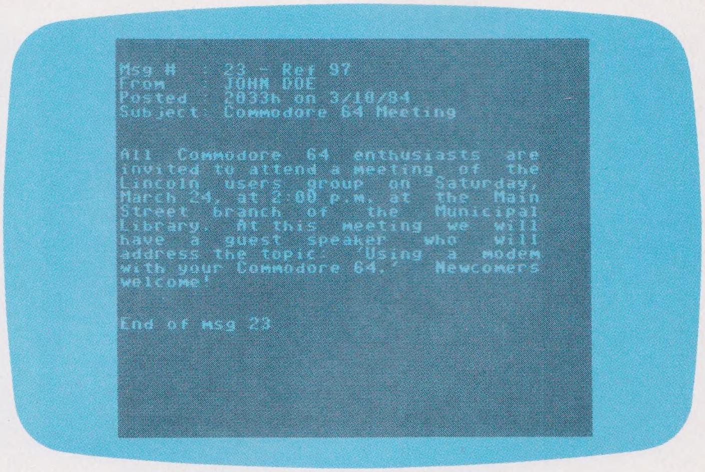
The Rules of Data Exchange

The screen at right shows the procedure for connecting to a typical electronic bulletin board — a process called logging on. Gaining access to this local bulletin board requires a code word (in this case, **privet).** Once you have identified yourself, you can leave messages for all users to read, or send or receive a private message.



```
Lynx-Line Bulletin Board

Operated by: Jim Strasma and Dan Gauwitz

Written by:  Steve Punter

Log In At 1021h
Your First Name?  John
Your Last Name?  Doe
Standby...
VERY WELL JOHN DOE
Is this your first time on the system?
yes
Your City/State?(30 chrs max)  Lincoln,
IL
Please supply a 6 character USER CODE
>privet
Name:   JOHN DOE
City:   LINCOLN, IL
```

The screen at right — announcing a meeting of Commodore users — shows how a message might appear on a bulletin board. Other messages might contain information about software updates, new hardware, or computer classes. Or you might leave a message for a friend asking for help with a software problem.



```
Msg #   : 23 - Ref 97
From    : JOHN DOE
Posted  : 2033h on 3/18/84
Subject: Commodore 64 Meeting


All  Commodore  64  enthusiasts  are
invited to attend a meeting  of  the
Lincoln  users  group  on  Saturday,
March 24, at 2:00 p.m. at  the  Main
Street  branch  of   the  Municipal
Library.  At this  meeting  we  will
have  a  guest  speaker  who   will
address the topic:  'Using  a  modem
with your Commodore 64.'  Newcomers
welcome!


End of msg 23
```

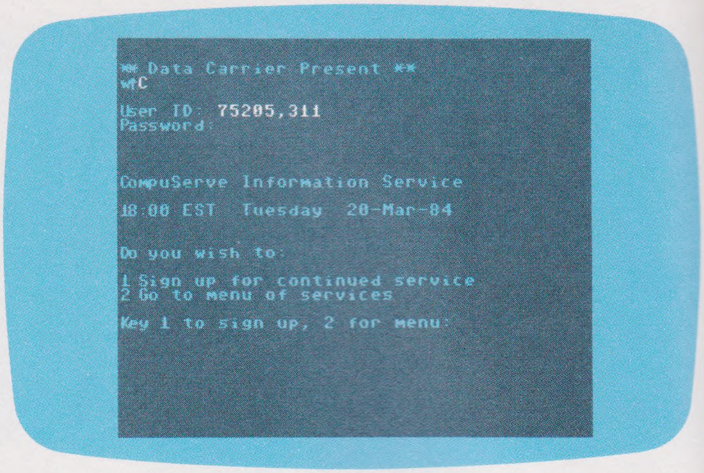
quire that you answer questions about what settings to use. You can answer these questions without a detailed understanding of the terms; it is essential, however, that both computers use compatible settings. These include baud rate (the speed at which the computer will send and receive data), the number of data bits per character, the number of stop bits, whether you will be transmitting in full duplex (allowing information to go in both directions at once) or half duplex (one direction at a time), and the type of parity checking to use in error detection.

Only the most sophisticated programs can send incoming information directly to the printer or disk; most terminal programs for the 64 copy the incoming information into the computer's memory only. You must then save the data, or print it out in a separate step. If you forget to do this, you may lose it.

If you plan to send or receive BASIC programs, you will need to get communications software that is up to the job. Many communications programs for the 64 cannot send or receive BASIC programs properly, since these processes require a sophisticated level of programming.

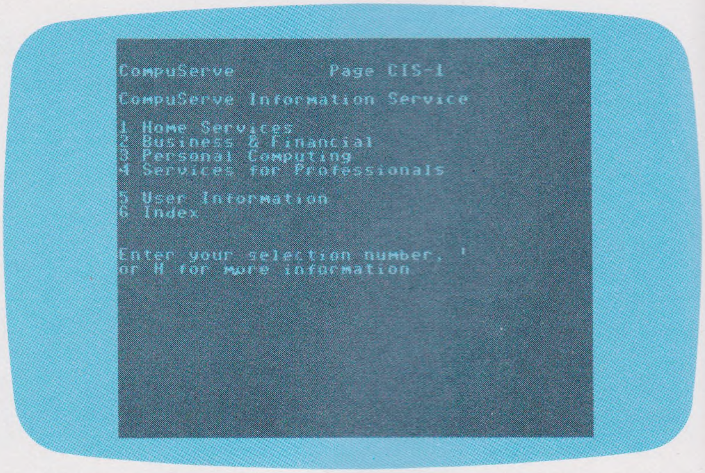
The Convenience of Commercial Data Bases

The screen at right shows a typical log-on procedure for CompuServe. First dial the CompuServe number provided for your area. Once connected, hold down the **CONTROL** key and press **C**. You will then be asked for your ID number and password. To ensure that your password remains private, CompuServe keeps the word from showing on the screen when you type it.



```
** Data Carrier Present **
↑C

User ID: 75205,311
Password:


CompuServe Information Service

18:00 EST  Tuesday  20-Mar-84


Do you wish to:

1 Sign up for continued service
2 Go to menu of services

Key 1 to sign up, 2 for menu:
```

Once you are connected to CompuServe, a main menu like that at right lists a wide array of services. Each menu entry leads to another, specialized menu; entering **2**, for example, brings up a list of financial and business services. To go directly to Commodore's Information Network, type **go cbm**.



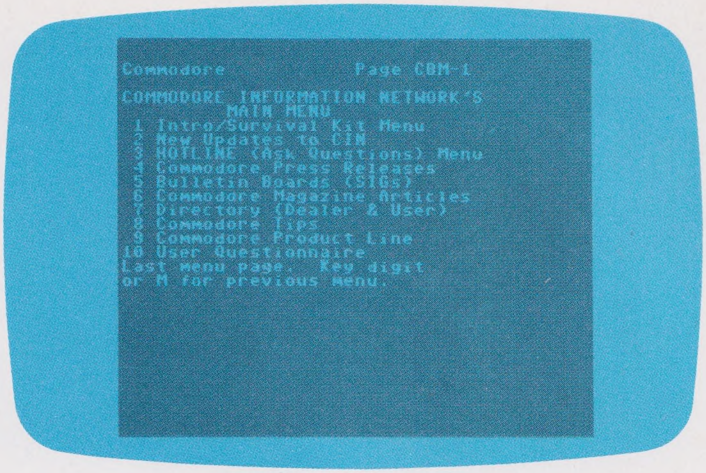
```
CompuServe         Page CIS-1

CompuServe Information Service

1 Home Services
2 Business & Financial
3 Personal Computing
4 Services for Professionals

5 User Information
6 Index


Enter your selection number, !
or H for more information
```

Among the simplest and most fruitful uses of your Commodore 64 in communications is linking up with a commercial information service, such as The Source or CompuServe. Designed by their operators to be easy to use, these services place a huge amount of data at your disposal.

Most information services work in similar ways. You open an account (applications are available through computer dealers and by mail) and pay an initial fee. The service assigns you an identification number and a password, and you use these to gain access to the system. The service bills you for the amount of time you are connected; most services vary the rates according to the time of day. And of course, you pay the telephone company for the time you use on the phone lines.

Even if there is no direct telephone number in your area for the information service you want, you may still be able to avoid long-distance rates. Most services can tell you about specialized communications networks, which connect you to a distant service with only a local call.

CompuServe has a feature of particular interest to Commodore users. Commodore has chosen that service to carry its special information network. Through this network you can talk via computer to other 64 users, gain access to a Commodore newsletter, and get questions answered quickly through the Hotline.



```
Commodore              Page CBM-1

COMMODORE INFORMATION NETWORK'S
         MAIN MENU
 1 Intro/Survival Kit Menu
 2 New Updates to CIN
 3 HOTLINE (Ask Questions) Menu
 4 Commodore Press Releases
 5 Bulletin Boards (SIGs)
 6 Commodore Magazine Articles
 7 Directory (Dealer & User)
 8 Commodore Tips
 9 Commodore Product Line
10 User Questionnaire
Last menu page.  Key digit
or M for previous menu.
```

The menu for the Commodore network shows the variety of Commodore-related information available. The Hotline (Option 3) offers advice about knotty computer problems; Option 9 leads you to announcements about new products. And you can gain access to Commodore's user bulletin board by choosing Option 5.

An Assortment of Services

Commercial information utilities provide easy access to an enormous variety of information and a vast array of services. Networks such as CompuServe and The Source offer an assortment of services through one centralized facility. Other utilities are more specialized, providing such services as bibliographic searches of professional journals, or detailed information on commodity prices.

These services can save you substantial time and money by eliminating transportation costs and reducing the hours you would otherwise spend in libraries. But keep in mind that user fees can quickly add up. To keep your costs at a reasonable level, familiarize yourself with user guides before you log on, and have a good idea about what part of the service you want to use and what information you are seeking. Plan to use the service during evenings or on weekends, when the hourly access charges are lower.

The following is a brief guide to the range of information and services available to you through your modem and your Commodore 64.

NEWS AND WEATHER: Electronic newspapers range from up-to-the-minute headline services to complete newspapers, including everything from weather and sports to astrology.

FINANCIAL INFORMATION: You can survey constantly changing stock and commodity prices, review data on thousands of corporations, or research current economic indicators.

ON-LINE SEARCHES: On-line reference services let you leaf electronically through reference materials. You can search the *New York Times* index, or a whole encyclopedia, or hundreds of professional and trade journals.

ELECTRONIC PUBLISHING: You can call up special editions of your favorite magazines, or electronically publish your own poems, articles, newsletters and books.

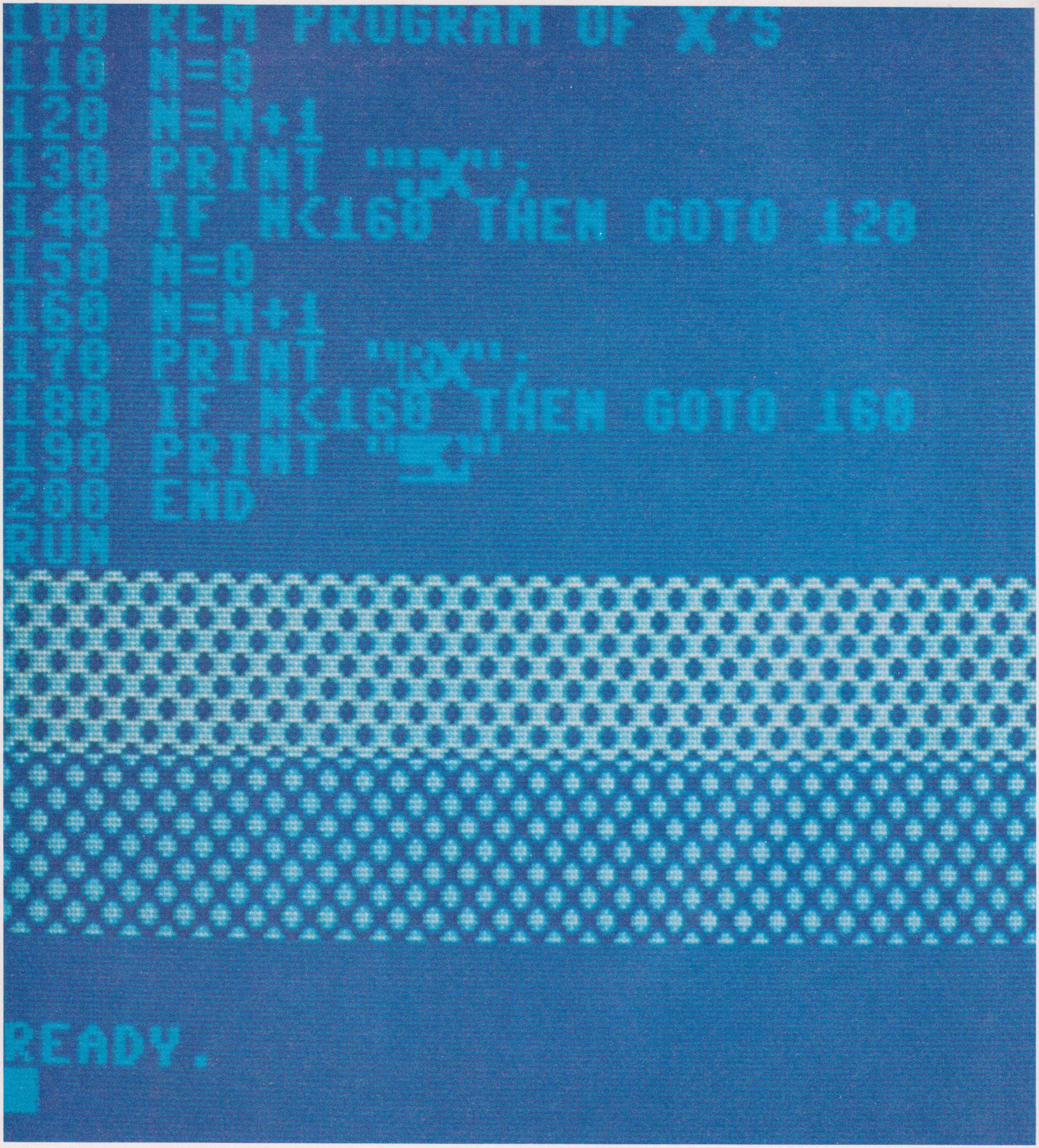
SHOP-AT-HOME SERVICES: Electronic catalogues let you browse, place an order and charge it to your credit card number.

BANK-AT-HOME SERVICES: Computerized banking speeds transactions, letting you transfer funds between accounts or pay bills from your account.

TRAVEL INFORMATION: You can check airline schedules, make reservations, and order and charge your tickets in much the same way a travel agent does.

ENTERTAINMENT: Special listings give you ticket-ordering facilities, movie and theater reviews, and restaurant menus. Or you can play multi-user strategy games, chat with friends on a computer version of CB radio, read your horoscope, or use a matchmaking service to line up a Saturday night date.

ELECTRONIC MAIL: These services let you save time and avoid messenger costs by transferring memos, contracts, or letters directly from computer to computer.



```
100 REM PROGRAM OF X'S
110 N=0
120 N=N+1
130 PRINT "X";
140 IF N<160 THEN GOTO 120
150 N=0
160 N=N+1
170 PRINT "X";
180 IF N<160 THEN GOTO 160
190 PRINT "■"
200 END
RUN
```

```
READY.
█
```

Programming in BASIC: Home-grown Software

The job of programming — writing the software that makes your Commodore 64 such a versatile tool — is often regarded as an arcane task best left to highly trained professionals. But BASIC, the language that you use to communicate with the 64, makes it relatively easy for you to write your own programs, and learn about programming in the process.

BASIC, like most programming languages for non-experts, is what is called a higher-level language. These languages are themselves programs that translate easy-to-remember commands, called source code, into the machine code that the computer understands. Designed as a teaching tool, BASIC lets you try out different ways of programming, and see the results, quickly and easily.

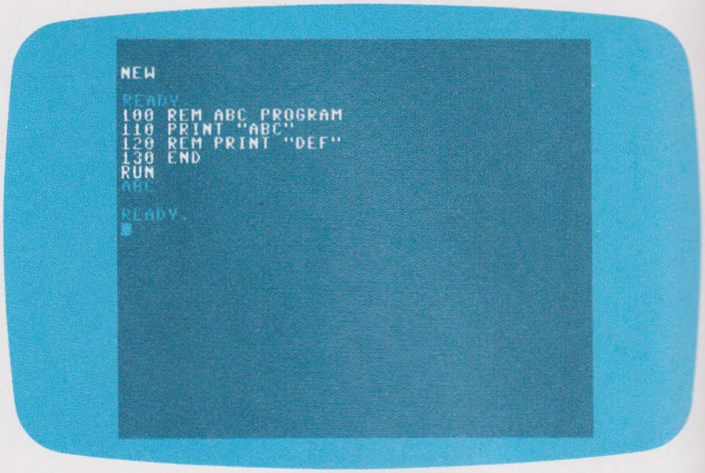
The version of BASIC installed in your 64 is an interpretive language: Each line of instructions is interpreted into machine code whenever the program is run, but the original source code is always preserved. This means that as you are developing a program, you can run a line or group of lines to see whether it is doing what you want, and revise the source code if necessary.

Like most interaction with a computer, programming in BASIC requires that you be very precise in formulating your instructions. Each BASIC program line is made up of two kinds of information — the key words and special characters that have specified meanings in BASIC, and the ordinary letters and numbers of the data the program processes. One feature of BASIC that makes it easy to use is that key words are usually self-explanatory. All the program elements must be assembled according to the system (called syntax) that BASIC understands. When BASIC encounters instructions that do not make sense within its syntax, it responds with the message **SYNTAX ERROR**. If your instructions do make sense to BASIC, but you have misused a key word, the line will be executed — but with unpredictable results. Be sure to check each line carefully as you type it.

In this chapter you will meet some of the most fundamental elements of BASIC, which let you write simple programs. You should try the examples on your computer, and try out variations of the examples as well, to get an understanding of the concepts they illustrate. Don't worry about mistakes; you can learn programming only by writing programs. BASIC helps make it easy.

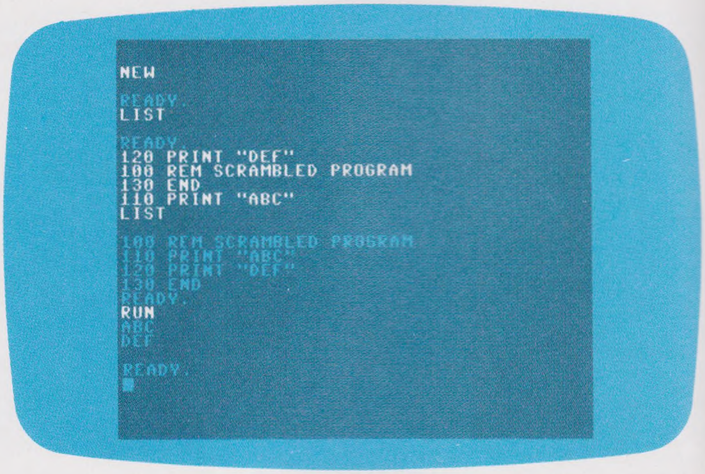
A few lines of BASIC program code make the Commodore 64 display this colorful pattern of the character *X* on the monitor. The numbers at the beginning of each line are the backbone of the program; BASIC reads the lines in numerical order, executing the instructions on each line before going on to the next. When this program runs, the shapes at the bottom of the screen are determined by commands in lines 130 and 170.

Simple Commands for Working in BASIC



```
NEW

READY.
100 REM ABC PROGRAM
110 PRINT "ABC"
120 REM PRINT "DEF"
130 END
RUN
ABC

READY.
█
```

To prepare for a new BASIC program, hold down **SHIFT** and press **CLEAR/HOME**; type **NEW** and press **RETURN** to clear the memory. After you enter each line of a program — in this case, a simple alphabetical sequence — press **RETURN** to go to the next line. If you make a mistake, use the **DELETE** key to erase it, then enter the correct version. Enter the **RUN** command; note that BASIC displays the characters in the *PRINT* statement on line 110, but not those on line 120. This is because the **REM** on line 120 tells BASIC that the remainder of the line is a programmer's remark, not to be executed.



```
NEW

READY.
LIST

READY.
120 PRINT "DEF"
100 REM SCRAMBLED PROGRAM
130 END
110 PRINT "ABC"
LIST

100 REM SCRAMBLED PROGRAM
110 PRINT "ABC"
120 PRINT "DEF"
130 END
READY.
RUN
ABC
DEF

READY.
█
```

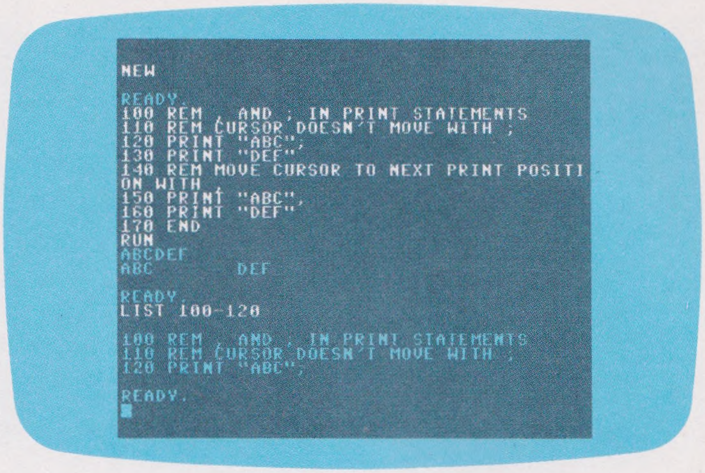
Enter **NEW** to clear the previous program from memory. You can then confirm that the memory has been cleared; enter **LIST** and see that BASIC displays only a blank line. Type each line as shown, pressing **RETURN** after each. When you again enter **LIST**, BASIC displays the lines you entered, but arranged in numerical order. If you run the program you will see that it prints **ABC** on one line and **DEF** on the next; after a *PRINT* statement, the cursor normally moves to the beginning of the next screen line.

BASIC executes the lines of a program one at a time, in numerical order. You can use any whole number from 0 to 63999 as a line number. Although you could write a program starting with line 1 and continuing in increments of 1, it would be inconvenient in practice, leaving no room between lines for later additions. One numbering system is to start at 100 and continue in increments of 10. This leaves room at the beginning and between lines for changes.

It is good practice to enter a **NEW** command every time you start a new program. **NEW** clears the part of memory the 64 uses to store BASIC programs, to ensure that no elements of an old program will enter your new work. But save the previous program before you type **NEW**; you might otherwise undo a lot of work.

At any point in program writing, you can review your work by entering **LIST**. You can see how the program works so far with a **RUN** command. **RUN** tells BASIC to execute the stored program statements, starting at the lowest-numbered line.

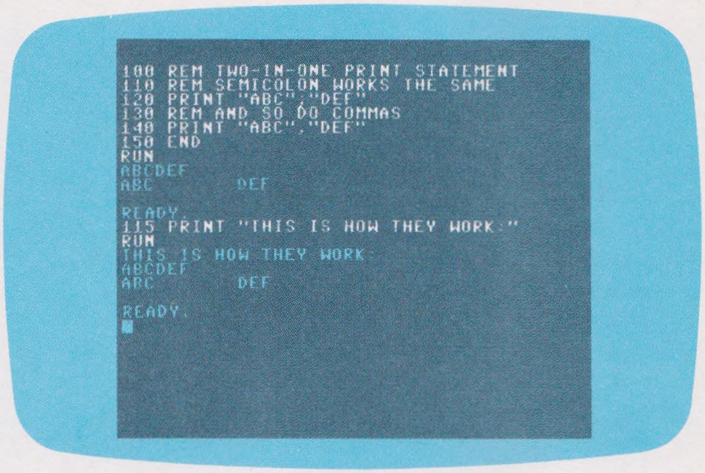
As you write a program, you can leave notes in it to remind yourself what certain parts of the program are intended to do. The *REM* statement (short for REMark) is a kind of memo pad; when BASIC encounters **REM**, it knows that the rest of the program line is not to be executed. Other uses for **REM** are to list the program title, author and date. One feature of



```
NEW

READY.
100 REM , AND ; IN PRINT STATEMENTS
110 REM CURSOR DOESN'T MOVE WITH ;
120 PRINT "ABC";
130 PRINT "DEF"
140 REM MOVE CURSOR TO NEXT PRINT POSITI
ON WITH
150 PRINT "ABC",
160 PRINT "DEF"
170 END
RUN
ABCDEF
ABC       DEF

READY.
LIST 100-120

100 REM , AND ; IN PRINT STATEMENTS
110 REM CURSOR DOESN'T MOVE WITH ;
120 PRINT "ABC";

READY.
█
```

The semicolon at the end of the *PRINT* statement in line 120 keeps the cursor from advancing, so the characters from line 130 are displayed immediately to the right of those from line 130. The comma in line 150 advances the cursor to the next *PRINT* position before the characters from line 160 are displayed.

The **LIST** command followed by two numbers that are separated by a hyphen causes BASIC to display the contents of the lines starting with the first number and ending with the second number.



```
100 REM TWO-IN-ONE PRINT STATEMENT
110 REM SEMICOLON WORKS THE SAME
120 PRINT "ABC";"DEF"
130 REM AND SO DO COMMAS
140 PRINT "ABC","DEF"
150 END
RUN
ABCDEF
ABC       DEF

READY.
115 PRINT "THIS IS HOW THEY WORK:"
RUN
THIS IS HOW THEY WORK:
ABCDEF
ABC       DEF

READY.
█
```

A single *PRINT* statement may contain more than one group of characters within quotes, as shown here, if the groups are separated by semicolons or commas. This program produces the same result as the one shown above, but requires fewer statements. Note the addition of line 115 after the first run of the program. In the second run, line 115 is executed between lines 110 and 120; regardless of the order in which lines are written, BASIC executes them in the order of their line numbers. For the program at left, and subsequent programs, **NEW** was entered before the screen was cleared.

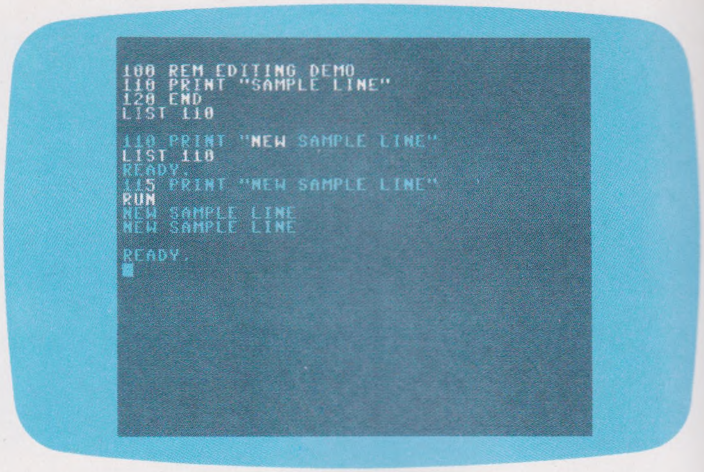
a well-planned program is an abundance of *REM* statements. These are useful when any user needs to examine a program; though the program makes sense when you write it, it may not later.

The messages that a program displays on the monitor as it runs are controlled by *PRINT* statements. When you run the program, the cursor disappears, but it is still moving invisibly across the screen, telling the computer where to put the next character. The cursor movements are governed by well-defined rules, and it's a good idea to know them so you can control the cursor readily.

When BASIC encounters a *PRINT* statement in a program, it displays all the characters that appear between quotation marks in the statement, positioning the first character wherever the cursor is located. Normally, the cursor advances to the beginning of the next screen line after the characters in quotes are displayed. But if the quotation marks are followed by a semicolon, the cursor stays where it is. A comma after the quotes causes the cursor to advance to the next preset **PRINT** position. The computer has 40 columns, numbered 0 to 39; column positions 0, 10, 20 and 30 are preset **PRINT** positions. A comma or semicolon has the same effect whether it is used at the end of one *PRINT* statement followed by another, or between items in a single *PRINT* statement.

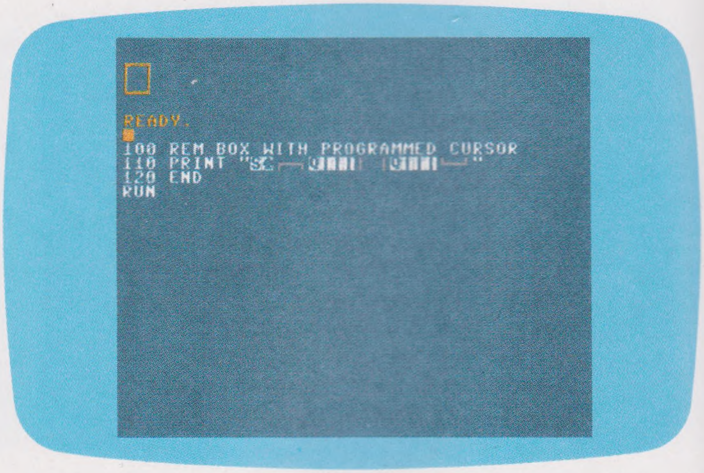
Editing BASIC Program Lines

Enter the three-line program at the top of the screen at right; then enter the next line, a *LIST* statement that causes BASIC to display line 110. Move the cursor onto the **S** in **SAMPLE** and depress the **SHIFT** key as you press the **INSERT/DELETE** key four times. A space is created each time to the right of the cursor. Type **NEW** in the first three spaces and press **RETURN**. When you list line 110 again, it will have changed. Move the cursor onto the *0* in *110* and type *5* and press **RETURN**. You have now added a line 115 that is identical, except for the number, to line 110 — which remains unchanged.



```
100 REM EDITING DEMO
110 PRINT "SAMPLE LINE"
120 END
LIST 110

110 PRINT "NEW SAMPLE LINE"
LIST 110
READY.
115 PRINT "NEW SAMPLE LINE"
RUN
NEW SAMPLE LINE
NEW SAMPLE LINE

READY.
█
```

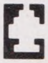
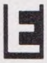
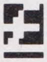
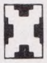
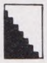
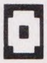
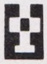
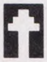
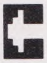
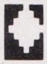
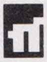
The program at right creates the large box shown on page 33, using cursor-control and graphics characters in a *PRINT* statement. Move the cursor down the screen six lines before typing in the program. The characters in line 110 in dark blue surrounded by white appear when you press cursor-control and color-control keys following a quotation mark *(chart, opposite)*. The characters shown in gray are the graphics characters that form the box. When the program is run, the cursor moves to the upper left corner of the screen and prints the box.



```
□

READY.
█
100 REM BOX WITH PROGRAMMED CURSOR
110 PRINT "SC┌─▮▮▮▮│ │▮▮▮▮└─┘"
120 END
RUN
```

Instructions in BASIC programs must be precisely typed, or you will get unpredictable results. Fortunately, BASIC has features that make it easy for you to edit lines, either to correct errors or to change the way a program works. You can change a line on the screen by moving the cursor to the line, then typing over characters, or adding or removing characters with the **INSERT/DELETE** key. Changes made on the screen take effect only when you press **RETURN** while the cursor is on the altered line. If you use **SHIFT** and **RETURN** together, the line will not be changed in memory. If you change the line number of a line listed on the screen, that line will be entered in memory under the new number, but the original line will also remain in memory.

The *PRINT* statement is a versatile tool in BASIC programming. It can be used to display text messages, move the cursor, display graphics characters, and change the color of the cursor and printed characters. A *PRINT* statement performs cursor movements and color changes in the order in which the instructions are entered after the statement's opening quotation mark. When you type cursor and color instructions in a *PRINT* statement, you are working in "quote mode." In quote mode each cursor and color operation is represented onscreen as a reversed character *(charts, opposite);* no cursor movements or color changes occur

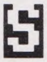
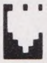
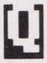
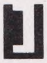
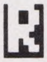
COLOR	KEYS	SYMBOL	COLOR	KEYS	SYMBOL
BLACK	CTRL 1		ORANGE	C= 1	
WHITE	CTRL 2		BROWN	C= 2	
RED	CTRL 3		LIGHT RED	C= 3	
CYAN	CTRL 4		DARK GREY	C= 4	
PURPLE	CTRL 5		MEDIUM GREY	C= 5	
GREEN	CTRL 6		LIGHT GREEN	C= 6	
BLUE	CTRL 7		LIGHT BLUE	C= 7	
YELLOW	CTRL 8		LIGHT GREY	C= 8	

QUOTE-MODE COLOR CONTROLS

When the color-change keys for the cursor or printed characters are used in quote mode, they generate the symbols shown above. When the program is run, the cursor or the character it prints will change color.

QUOTE-MODE CURSOR CONTROLS

When cursor-control keys are pressed in quote mode, the symbols appear on the screen as shown at right. When you run the program, the symbols will not appear and the cursor will follow your instructions.

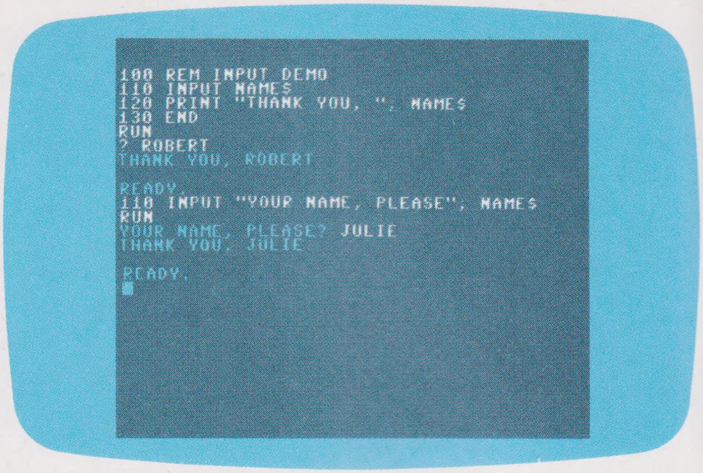
KEY FUNCTION	KEYS	SYMBOL
CURSOR HOME	CLR HOME	
CLEAR SCREEN	SHIFT CLR HOME	
CURSOR DOWN	⇑ CRSR ⇓	
CURSOR UP	SHIFT ⇑ CRSR ⇓	
CURSOR RIGHT	⇐ CRSR ⇒	
CURSOR LEFT	SHIFT ⇐ CRSR ⇒	
REVERSE ON	CTRL RVS ON	
REVERSE OFF	CTRL RVS OFF	

until the program is run. Quote mode stays in effect until you type another quotation mark, or press **RETURN** or **SHIFT** and **RETURN**.

Learning to use quote mode for cursor movement and color changes opens the door to building complex graphic figures. Each *PRINT* statement can hold many separate operations and characters in combination; only 19 keystrokes are required to build the box shown in the example on the bottom screen opposite.

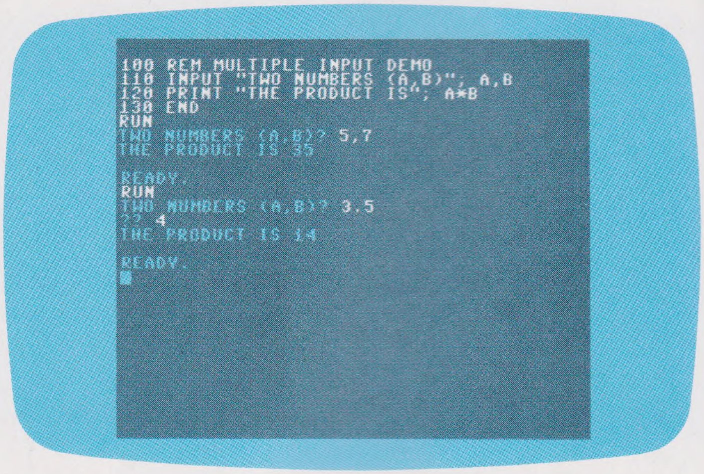
Getting Data from the Keyboard

The *INPUT* statement on line 110 of this program tells BASIC to take information typed at the keyboard and store it in the variable **NAME$**. In the next line, the content of **NAME$** is used as part of the message displayed by the *PRINT* statement. A second version of line 110 includes a prompt as part of the *INPUT* statement, to tell the user what information the program requires.



```
100 REM INPUT DEMO
110 INPUT NAME$
120 PRINT "THANK YOU, "; NAME$
130 END
RUN
? ROBERT
THANK YOU, ROBERT

READY.
110 INPUT "YOUR NAME, PLEASE", NAME$
RUN
YOUR NAME, PLEASE? JULIE
THANK YOU, JULIE

READY.
■
```

The *INPUT* statement in line 110 calls for two data items. The variables are separated by commas, and the user's responses must also be separated by commas. In this example the two variables are the numeric variables **A** and **B**. The values stored in these variables are multiplied and printed as the product incorporated in the message displayed by the *PRINT* statement. If the user fails to provide enough data items, as in the second case, the computer displays **??**, indicating that more items are required. When the user enters a second number — **4** — the program continues.



```
100 REM MULTIPLE INPUT DEMO
110 INPUT "TWO NUMBERS (A,B)"; A,B
120 PRINT "THE PRODUCT IS"; A*B
130 END
RUN
TWO NUMBERS (A,B)? 5,7
THE PRODUCT IS 35

READY.
RUN
TWO NUMBERS (A,B)? 3.5
?? 4
THE PRODUCT IS 14

READY.
■
```

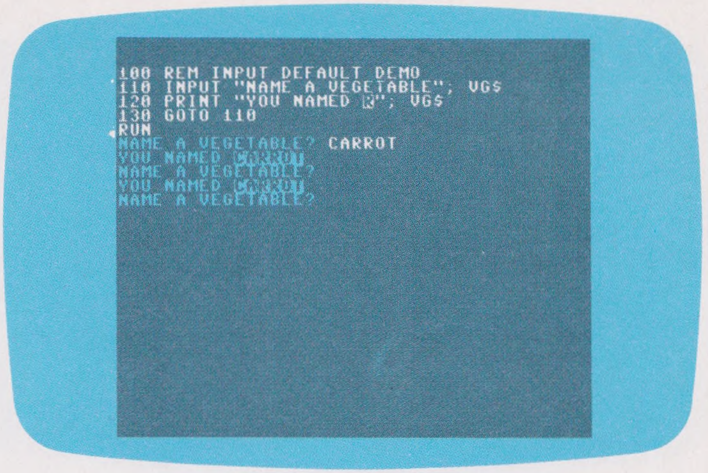
A variable is one of BASIC's most useful tools. BASIC uses variables as if they were file folders, each with a name. You can store any value in the variable. To manipulate that value, you instruct BASIC to perform an operation on the variable itself.

One way to assign values to variables is to use an *INPUT* statement, which lets BASIC read data from the keyboard while a program is running. When BASIC encounters an *INPUT* statement, it puts a question mark on the screen and flashes the cursor. When you type something and press **RETURN**, BASIC stores the typed material in the variable named in the *INPUT* statement, then goes on to the next program line.

With an *INPUT* statement you can also display a prompt before the question mark. As with a *PRINT* statement, the prompt must be enclosed in quotes and separated from the variable name by a semicolon.

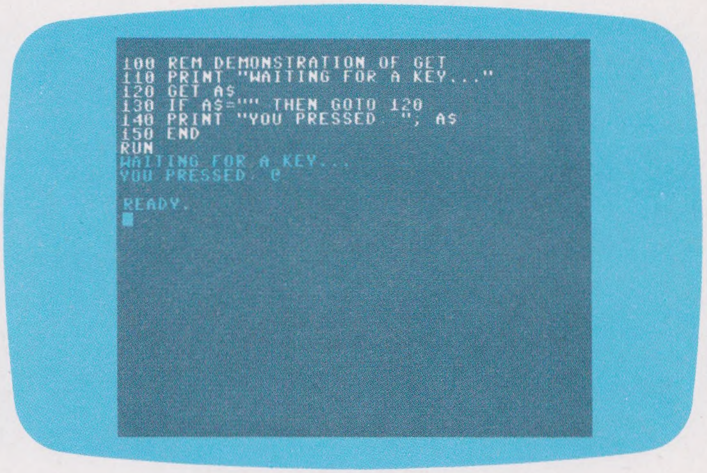
An *INPUT* statement may request several pieces of data. The variables in the statement must be separated by commas; so must the response items from the keyboard. If you type more items than required, the computer will say **?EXTRA IGNORED**. If too few items are typed, the computer will ask for more by printing two question marks on a new line.

BASIC distinguishes between two types of variable, called string and numeric variables. A numeric variable can hold only numbers; a string



```
100 REM INPUT DEFAULT DEMO
110 INPUT "NAME A VEGETABLE"; VG$
120 PRINT "YOU NAMED R"; VG$
130 GOTO 110
RUN
NAME A VEGETABLE? CARROT
YOU NAMED CARROT
NAME A VEGETABLE?
YOU NAMED CARROT
NAME A VEGETABLE?
```

When only the **RETURN** key is pressed in response to an *INPUT* statement, BASIC automatically keeps the previous value stored in the variable specified in the statement. In this example, the word *CARROT* was typed as the first response to the *INPUT* statement. The next time, however, the user simply pressed **RETURN**. Rather than changing the content of **VG$**, BASIC kept its previous content — *CARROT.*



```
100 REM DEMONSTRATION OF GET
110 PRINT "WAITING FOR A KEY..."
120 GET A$
130 IF A$="" THEN GOTO 120
140 PRINT "YOU PRESSED: "; A$
150 END
RUN
WAITING FOR A KEY...
YOU PRESSED: @

READY.
█
```

The *GET* statement in line 120 automatically fills its variable with the value of the key that is pressed when BASIC executes the *GET* statement. If no key is pressed, the variable is filled with a null string, and the program goes on to the next line. The *IF* statement on line 130 tells BASIC to go back to line 120 if **A$** holds a null string, indicated by empty quotes. This construction is commonly used to make a program wait for a response; the program continues the loop until a key is pressed. In this case the value of the key is stored in **A$**, and the program continues with line 140.

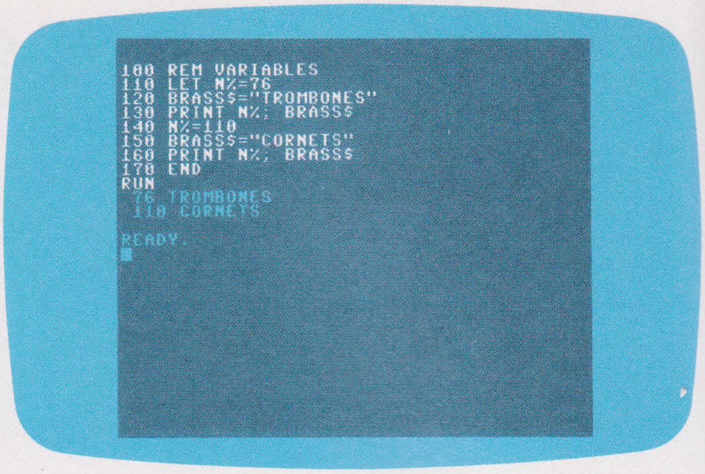
variable, which must have a name ending with **$**, can hold any combination of letters and numbers. You can store numeric data in a string variable, but BASIC won't be able to use it in mathematical computations, since BASIC does not see string variables as numbers. If you try to put nonnumeric characters in a numeric variable, BASIC will respond with the message **?REDO FROM START**. Then you must retype your input correctly.

Another way to get keyboard data into a program is with a *GET* statement, which takes single-character responses and does not wait for you to press **RETURN**. BASIC reads whatever key is pressed, stores that value in a string variable, and goes to the next program line. If no key is pressed, BASIC fills the variable with a "null string" — one with no characters — and goes on. The null string is expressed in BASIC as two sets of quotation marks ("") with no space between them. You can make the program wait for a key to be pressed: Follow the *GET* statement with an *IF . . . THEN* statement *(line 130, screen immediately above)* that holds the program in a loop as long as the variable contains a null string.

A *GET* statement provides no prompt or flashing cursor, but you can use a *PRINT* statement with it to display a prompt.

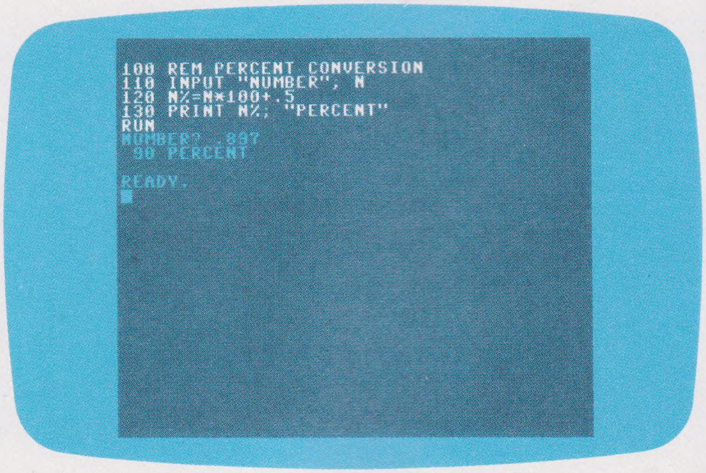
Putting Values into Variables

This program uses two types of simple variables. **N%** is an integer variable, first assigned a value of 76. **BRASS$** is a string variable, first assigned the character string *TROMBONES*. The *PRINT* statement in line 130 displays both these variables in sequence. Note the semicolon, which keeps the cursor in place after printing the first element. **N%** in line 140 is reassigned a value of 110 and in line 150 **BRASS$** is reassigned the characters *CORNETS*.



```
100 REM VARIABLES
110 LET N%=76
120 BRASS$="TROMBONES"
130 PRINT N%; BRASS$
140 N%=110
150 BRASS$="CORNETS"
160 PRINT N%; BRASS$
170 END
RUN
 76 TROMBONES
 110 CORNETS

READY.
■
```

This program takes a typed decimal number and converts it to the nearest whole percentage. A floating-point variable named **N**, which can hold a number that has decimal places, is used in the *INPUT* statement in line 110. Line 120 computes the per cent equivalent of the decimal fraction, then adds .5, and assigns the result to a new, integer variable named **N%**. Since this variable can hold only integers, it drops any numbers to the right of the decimal point; the *PRINT* statement on line 130 thus displays only a whole number, no matter what decimal number is initially typed.



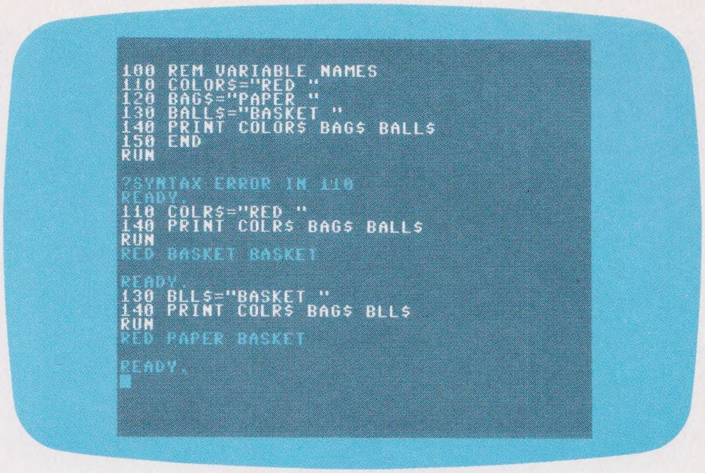
```
100 REM PERCENT CONVERSION
110 INPUT "NUMBER", N
120 N%=M*100+.5
130 PRINT N%; "PERCENT"
RUN
NUMBER? .897
 90 PERCENT

READY.
■
```

You can assign values to a variable from within a program, using a *LET* statement. This statement allows you to take full advantage of the 64's computing power, since the value you store in the variable need not be only a single value, but can be the result of a set of a computations. Thus the value of a variable might be the result of computations involving other variables. The expressions within a *LET* statement can be as complex as your program requires, provided they are written according to the rules required by BASIC.

You can omit the word *LET* in a *LET* statement *(see lines 110, 120 and 130, top screen, opposite)*; the essentials are a variable name followed by an equal sign and the expression or value to be assigned to the variable.

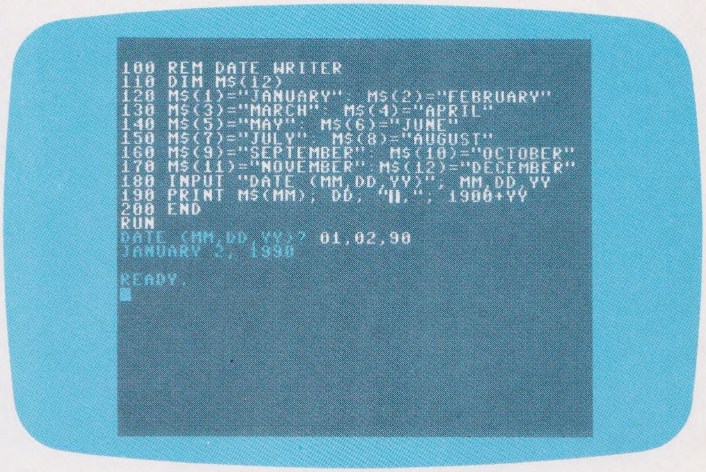
BASIC recognizes two types of numeric variables. Floating-point variables can hold numbers with decimal points, up to nine digits; integer variables can hold only whole numbers. Integer-variable names must always end with **%**, just as string variable names must end with **$**. A variable name must begin with a letter of the alphabet; subsequent characters may be letters or numerals. BASIC handles variables by only the first two characters of each name, so each of your variables must start with a different two-letter combination *(screen, top right)*. Be careful when you are using different types of variables together; if you try to use a

Long variable names help make a program understandable. However, there are two potential problems. A variable name must not include key words reserved in BASIC, such as **PRINT**, **GO**, **ON**, **OR**, **IF** and **NOT**. Such a variable would generate a **?SYNTAX ERROR** message. Also, each variable name must begin with a different two-letter combination; if two names begin with the same pair of letters, BASIC will refer to the most recent occurrence of the combination when it needs the value assigned to the variable. Both errors are included in this program, then corrected after they show up.



```
100 REM VARIABLE NAMES
110 COLOR$="RED "
120 BAG$="PAPER "
130 BALL$="BASKET "
140 PRINT COLOR$ BAG$ BALL$
150 END
RUN

?SYNTAX ERROR IN 110
READY.
110 COLR$="RED "
140 PRINT COLR$ BAG$ BALL$
RUN
RED BASKET BASKET

READY.
130 BLL$="BASKET "
140 PRINT COLR$ BAG$ BLL$
RUN
RED PAPER BASKET

READY.
█
```

This program uses an array to store names of the months of the year. Line 110 "dimensions" the array, indicating that it can hold 13 elements, numbered 0 through 12. When the program is run, the user types in the date in numeric form, and the program returns the date spelled out. BASIC assigns 01 to the variable **MM**, then uses this variable as a subscript to the array variable **M$**. The *PRINT* statement displays the value of **M$(1)** — *JANUARY*. The cursor-left in line 190 positions the comma immediately after the date.



```
100 REM DATE WRITER
110 DIM M$(12)
120 M$(1)="JANUARY": M$(2)="FEBRUARY"
130 M$(3)="MARCH": M$(4)="APRIL"
140 M$(5)="MAY": M$(6)="JUNE"
150 M$(7)="JULY": M$(8)="AUGUST"
160 M$(9)="SEPTEMBER": M$(10)="OCTOBER"
170 M$(11)="NOVEMBER":M$(12)="DECEMBER"
180 INPUT "DATE (MM,DD,YY)"; MM,DD,YY
190 PRINT M$(MM); DD; "▌,"; 1900+YY
200 END
RUN
DATE (MM,DD,YY)? 01,02,90
JANUARY 2, 1990

READY.
█
```

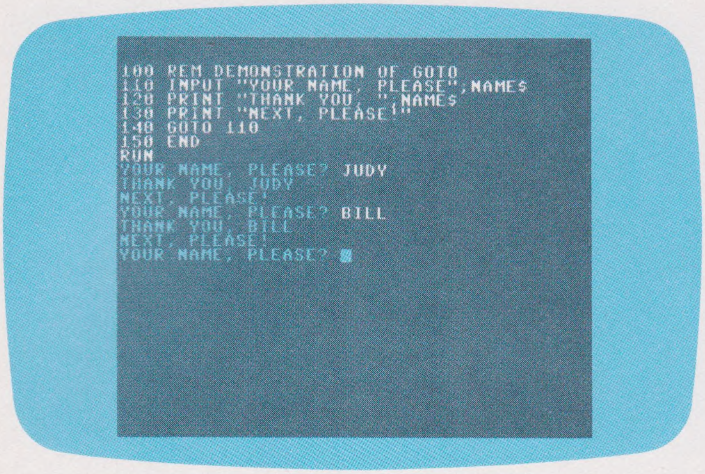
variable incorrectly — for instance, if you try to multiply with a string variable — BASIC will respond with the message **?TYPE MISMATCH ERROR**.

When you have a number of related values that you need to store in variables, you can use a device called an array. In an array, several variables are grouped together using the same name; they are differentiated by numeric subscripts in parentheses. The program uses the subscripts to tell BASIC which variable to use. Usually the variables in an array are arranged according to a numeric relationship, as in the example immediately above.

To set up an array of variables, you need to tell BASIC to reserve space for it, using a *DIM* (for dimension) statement. The *DIM* statement *(above, bottom)* contains a variable name followed by a number in parentheses; it tells BASIC to reserve space for elements numbered from zero through that number. The total number of elements made available is thus one greater than the number in parentheses.

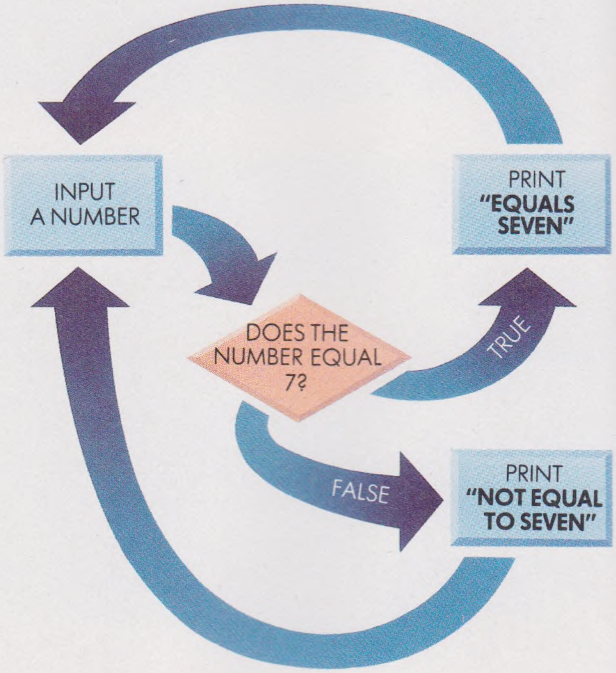
If you use a subscript higher than the number specified in the *DIM* statement, BASIC will display a **?BAD SUBSCRIPT ERROR** message. A **?REDIM'D ARRAY ERROR** will occur if the *DIM* value for any array is changed, or if the *DIM* statement appears in a program line following the first reference to one of the variables in the array.

Making Decisions with IF . . . THEN



```
100 REM DEMONSTRATION OF GOTO
110 INPUT "YOUR NAME, PLEASE",NAME$
120 PRINT "THANK YOU, ";NAME$
130 PRINT "NEXT, PLEASE!"
140 GOTO 110
150 END
RUN
YOUR NAME, PLEASE? JUDY
THANK YOU, JUDY
NEXT, PLEASE!
YOUR NAME, PLEASE? BILL
THANK YOU, BILL
NEXT, PLEASE!
YOUR NAME, PLEASE? █
```

The *GOTO* statement in line 140 causes this program to repeat indefinitely, simulating a polite receptionist who asks for the name of each person in a waiting room *(line 110)*, thanks each person by name *(line 120)*, and goes on to the next person. This process continues until you press the **RUN/STOP** and **RESTORE** keys together.

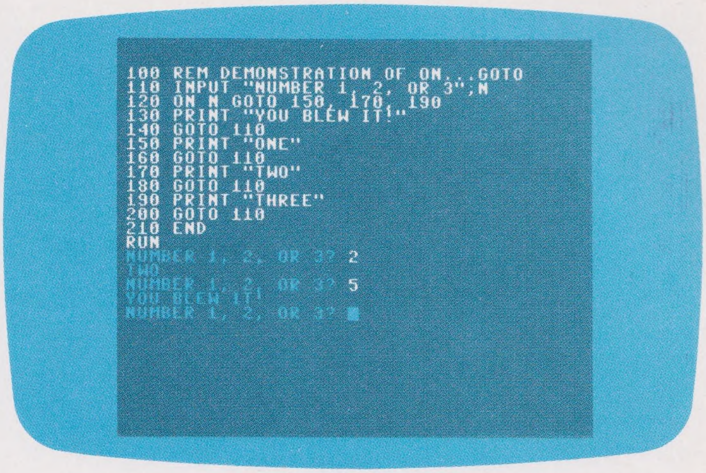


This flow chart illustrates the working of the *IF . . . THEN* statement in the program on the bottom screen, opposite. The starting point is an *INPUT* statement that takes a number from the user. The diamond-shaped box is where a decision is made: BASIC tests the truth of a specified expression. If the expression is true, BASIC executes the *THEN* statement. If false, the *IF* statement results in no action and the next BASIC line is executed. In either case, the program then goes back to the beginning and starts over.

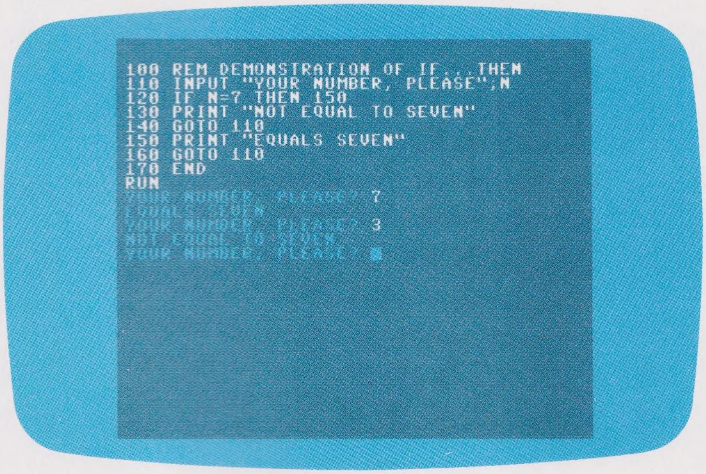
Much of the work done by a program consists of comparing values and performing actions based on the results of each comparison. The instruction that directs BASIC to do this is the *IF* statement, which usually includes the word *THEN*. The use of *IF . . . THEN* in BASIC is similar to its use in English; for example, you might tell a child, "If the sky is blue, then go outside to play," or tell BASIC, "If *B* is equal to 3, then print *B*." But unlike a child, BASIC does nothing if the expression proves false; in this analogy, if the sky were not blue, a child might still go outside, but if *B* did not equal 3, BASIC would simply go on to the next instruction.

Equality is only one of several conditions you can test by the use of an *IF . . . THEN* statement. Other symbols, known as logical operators, can be used in place of the equal sign, with different effects. The logical operators are > (greater than), >= (greater than or equal to), < (less

This program uses an *ON . . . GOTO* statement *(line 120)* to branch to one of three specified line numbers, depending on the value assigned to the variable **N** by the *INPUT* statement on line 110. If the value of **N** is 1, the program branches to the first specified line, number 150; if **N** is 2, the branch is to the second specified line, number 170, and so on. If the value of **N** is greater than the number of lines specified, BASIC simply continues executing lines in sequence, in this case displaying the message on line number 130. Each of the alternatives loops back to line 110 for another input.



```
100 REM DEMONSTRATION OF ON...GOTO
110 INPUT "NUMBER 1, 2, OR 3";N
120 ON N GOTO 150, 170, 190
130 PRINT "YOU BLEW IT!"
140 GOTO 110
150 PRINT "ONE"
160 GOTO 110
170 PRINT "TWO"
180 GOTO 110
190 PRINT "THREE"
200 GOTO 110
210 END
RUN
NUMBER 1, 2, OR 3? 2
TWO
NUMBER 1, 2, OR 3? 5
YOU BLEW IT!
NUMBER 1, 2, OR 3? █
```



```
100 REM DEMONSTRATION OF IF...THEN
110 INPUT "YOUR NUMBER, PLEASE";N
120 IF N=7 THEN 150
130 PRINT "NOT EQUAL TO SEVEN"
140 GOTO 110
150 PRINT "EQUALS SEVEN"
160 GOTO 110
170 END
RUN
YOUR NUMBER, PLEASE? 7
EQUALS SEVEN
YOUR NUMBER, PLEASE? 3
NOT EQUAL TO SEVEN
YOUR NUMBER, PLEASE? █
```

This program uses an *IF . . . THEN* statement to test the content of a variable assigned through an *INPUT* statement in line 110. At line 120 BASIC examines **N**; if **N** is equal to 7, then BASIC executes the rest of the line; it branches to line 150, where a *PRINT* statement displays the message **EQUALS SEVEN**. If the expression at line 120 is false, BASIC ignores the rest of that line and goes on to the next, and the *PRINT* statement on line 130 displays the message **NOT EQUAL TO SEVEN**. Lines 140 and 160 return the program to line 110 for another input.

than), <= (less than or equal to) and <> (not equal to).

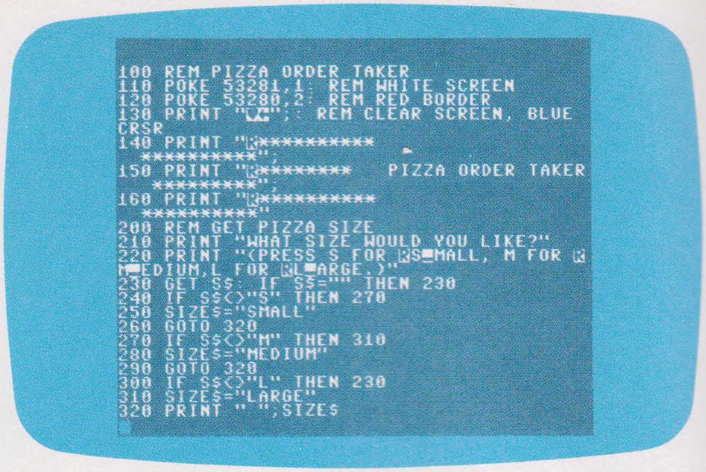
You can substitute *GOTO* for *THEN* in the *IF* statement *(line 120 on the screen immediately above)*, directing BASIC to branch to a specified line number if an expression is true. After branching as directed, BASIC executes the specified line and all subsequent lines. *GOTO* is a useful statement in its own right; it can be placed anywhere in the program without an **IF** statemenf, to branch out of the normal sequence of line numbers to any other point in the program.

You can also use *GOTO* in an *ON . . . GOTO* statement *(line 120, top screen, above)*, which causes BASIC to branch to one of several line numbers specified in the statement. This construction is often used to direct the program to a different set of procedures, as determined by the user's response to an *INPUT* statement.

As you begin to write programs that branch out of the normal sequence of line numbers, you may find it useful to outline the procedures you have in mind before you try to type the program lines themselves. A useful tool for this kind of program development is a flow chart, which describes with shapes and arrows the processes and flow of a program. Such a chart, along with ample internal documentation (using the *REM* statement), will help you and others understand why and how the finished program works as it does.

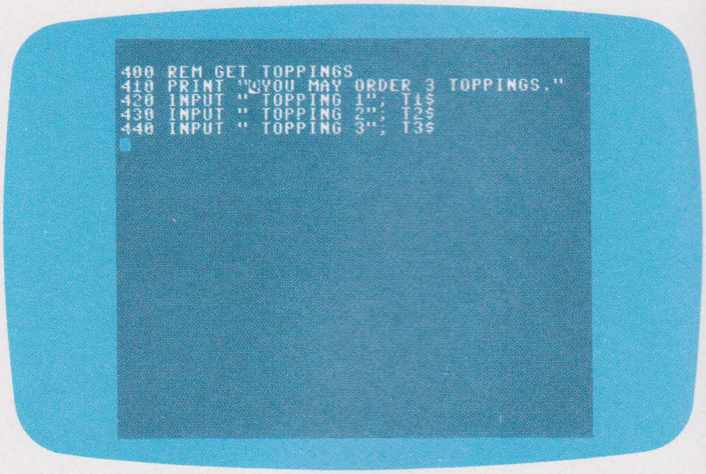
Putting Together a Simple Program

The program lines on this screen and the two following screens make up a **PIZZA ORDER TAKER** program. Lines 100 through 160 establish color settings and display the program title. Lines 200 through 320 take user input to establish the size of the pizza. The *GET* statement on line 230 takes the user's choice among the sizes on line 220; an *IF . . . THEN* statement tells BASIC to stay at line 230 until a choice is made. A sequence of *IF . . . THEN* statements sorts the input, assigning a value to **SIZE$** or branching to line 230 if the choice is not S, M or L. Line 320 displays the chosen size.



```
100 REM PIZZA ORDER TAKER
110 POKE 53281,1: REM WHITE SCREEN
120 POKE 53280,2: REM RED BORDER
130 PRINT "■■";: REM CLEAR SCREEN, BLUE
CRSR
140 PRINT "■***********
  ***********";
150 PRINT "■*********   PIZZA ORDER TAKER
  **********";
160 PRINT "■***********
  ***********"
200 REM GET PIZZA SIZE
210 PRINT "WHAT SIZE WOULD YOU LIKE?"
220 PRINT "(PRESS S FOR ■S■MALL, M FOR ■
M■EDIUM,L FOR ■L■ARGE.)"
230 GET S$: IF S$="" THEN 230
240 IF S$<>"S" THEN 270
250 SIZE$="SMALL"
260 GOTO 320
270 IF S$<>"M" THEN 310
280 SIZE$="MEDIUM"
290 GOTO 320
300 IF S$<>"L" THEN 230
310 SIZE$="LARGE"
320 PRINT " ";SIZE$
```

Lines 400 through 440 take the user's choice of toppings. Unlike the *GET* statement of line 230, the *INPUT* statements of this segment can take more than one character from the keyboard. Each *INPUT* statement assigns the user's input to a string variable, to be used later in the program when the complete order is displayed.



```
400 REM GET TOPPINGS
410 PRINT "■YOU MAY ORDER 3 TOPPINGS."
420 INPUT " TOPPING 1"; T1$
430 INPUT " TOPPING 2"; T2$
440 INPUT " TOPPING 3"; T3$
```

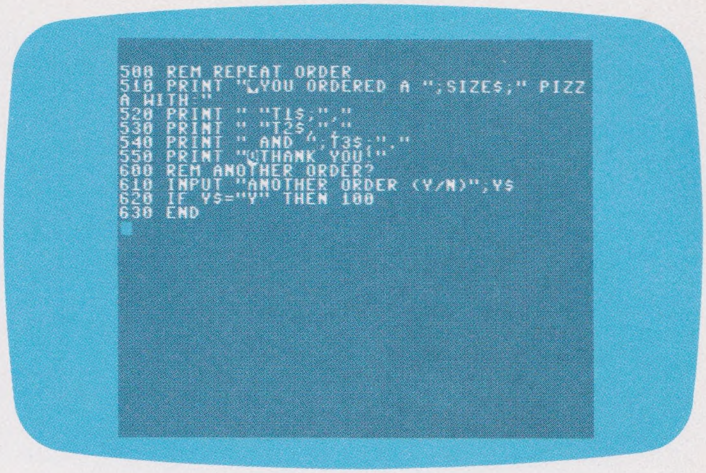
A BASIC program that can do real jobs for you is simply a combination of elements like those you have learned in this chapter. As you begin to develop simple programs of your own, you should follow a few guidelines to ensure that your programs are easy to understand, easy to modify and easy to use.

Before you begin to enter any program lines, take time to define the different jobs that need to be done within the program, then outline the major segments of the program that will do each of these jobs. This way you can write, test and correct each part of the program as you add it on; you will have far less trouble finding the bugs in a short segment than in an entire program that may be hundreds of lines long.

Building a program in segments also helps you keep a logical program flow, which makes the program easier for you and others to understand. And if each part is largely self-contained, you can more easily modify one part of the program without affecting the rest.

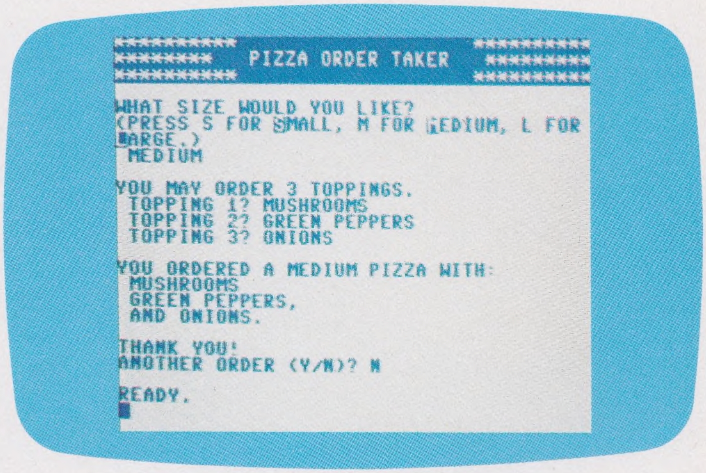
Leave room for the changes by starting the program with line 100, and numbering subsequent lines in increments of 10. For ease of reference, start each major segment on an even hundred. And to make sure that the program is easy to read, use *REM* statements to insert explanatory remarks before each segment and before any complicated operation.

The *PRINT* statements of lines 500 through 550 display the user's order, using the values assigned to variables in preceding lines. Lines 600 through 630 give the user a chance to place another order. If the response to the *INPUT* statement on line 610 is **Y**, the *IF* statement on line 620 causes BASIC to branch to the beginning of the program; any other response has no effect, and line 630 is executed next, ending the program.



```
500 REM REPEAT ORDER
510 PRINT "↓YOU ORDERED A ";SIZE$;" PIZZ
A WITH:"
520 PRINT " "T1$;","
530 PRINT " "T2$;","
540 PRINT " AND ";T3$;"."
550 PRINT "↓THANK YOU!"
600 REM ANOTHER ORDER?
610 INPUT "ANOTHER ORDER (Y/N)";Y$
620 IF Y$="Y" THEN 100
630 END
█
```

An actual run of the program is shown at right. Initiated by the **RUN** command, the program displays the title banner and then the prompt asking the user to choose a size. Note that even if the whole word is typed, the *GET* statement on line 230 assigns only its first letter to the variable **S$**. Next the program prompts the user for toppings; when they have been entered, the complete order is displayed. The user gets a chance to place another order; in this case, the response is **N**, and the program ends.

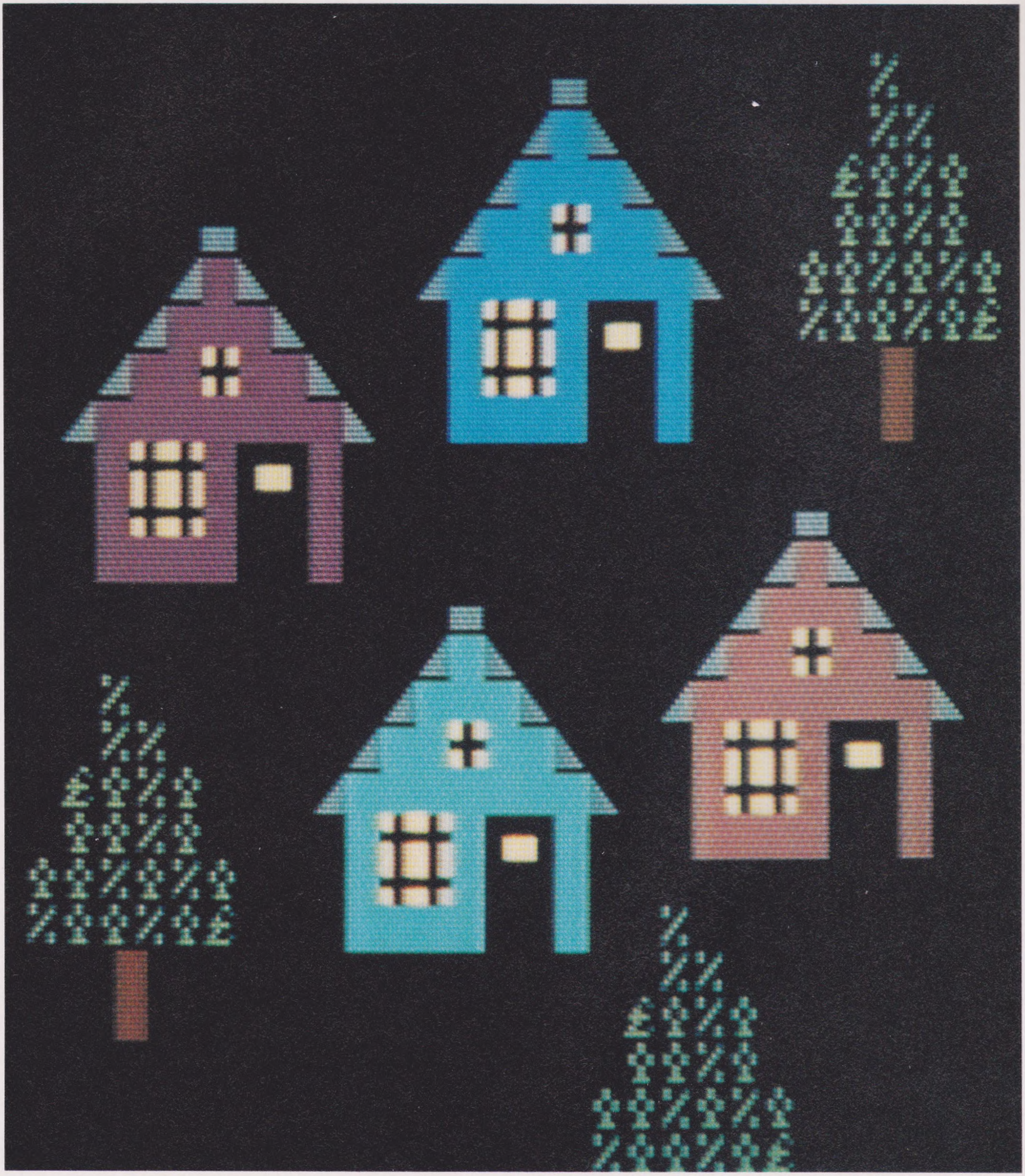


```
***********                 ***********
*********   PIZZA ORDER TAKER   *********
***********                 ***********

WHAT SIZE WOULD YOU LIKE?
(PRESS S FOR SMALL, M FOR MEDIUM, L FOR
LARGE.)
 MEDIUM

YOU MAY ORDER 3 TOPPINGS.
 TOPPING 1? MUSHROOMS
 TOPPING 2? GREEN PEPPERS
 TOPPING 3? ONIONS

YOU ORDERED A MEDIUM PIZZA WITH:
 MUSHROOMS
 GREEN PEPPERS,
 AND ONIONS.

THANK YOU!
ANOTHER ORDER (Y/N)? N

READY.
█
```

Try to keep the program simple, with its steps proceeding as much as possible in numeric sequence; too many jumps out of sequence make a program hard to read and correct, and slow to run. And until you are proficient in BASIC, keep program lines uncluttered. You can put two or more statements on one line if you separate them with colons, but you will have more trouble reading and correcting the program if you do.

The program illustrated above can be entered in segments, and you can run it at each of the stages shown, to see how it works up to that point. The program simulates a simple order-taking operation at a pizza parlor. It incorporates only one element of BASIC not previously discussed in this chapter — a *POKE* statement. *POKE* lets you put new information directly into a specified location in memory; it is used here to set colors for the screen border and background. The *POKE* statement is discussed in greater detail on pages 94 and 95. Type it in carefully, since a mistake can put your new data into the wrong memory address, erasing the material that should be there and leading to serious flaws in the working of your program.

When you use the **LIST** command to display the lines of a long program, it may scroll up the monitor screen too quickly for you to read. You can slow the scrolling down by depressing the **CONTROL** key.



Advanced BASIC: Building Programming Skills

The preceding chapter introduced you to enough elements of BASIC to let you begin developing simple programs of your own. In this chapter you will learn about a number of BASIC elements designed for the tasks your computer does best — processing large amounts of data and performing repetitive tasks.

The BASIC elements that make this possible, called loops and subroutines, are subdivisions that act as smaller programs within your program. In addition to the power they bring to bear on your computing jobs, they can help you write orderly, easily understood programs.

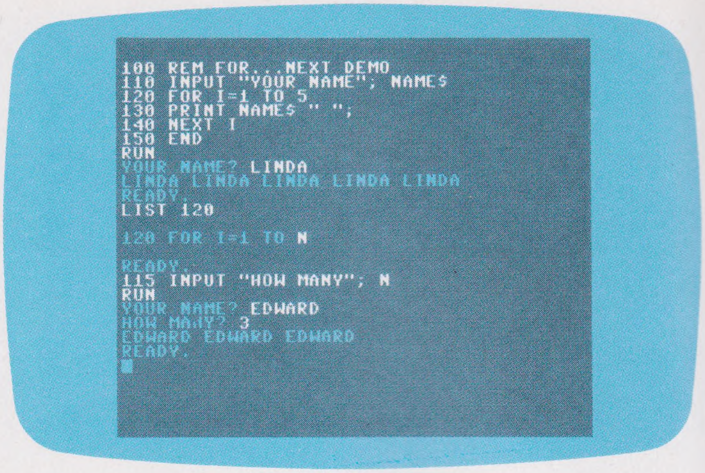
The best way to write a BASIC program is one piece at a time. First define the tasks that need to be done within the program, then write the subroutines, loops and lists of statements to do each of the jobs. This way you can test and correct each part of the program independently before all the pieces are together; you will have far less trouble finding the bugs in a short segment than in an entire program that may be hundreds of lines long.

Building a program in segments also helps you keep a logical program flow, which makes the program easier for you and others to understand. And if each part is largely self-contained, you can more easily modify one part of the program without affecting the rest.

With the elements of BASIC explained in this book, you can begin to develop programs for specialized jobs, or to write simple game programs. You will even be able to add sound and color to your programs. Keep in mind, however, that BASIC is a rich and diverse realm. There are scores of commands, statements and functions in the syntax of the language, each meant for a particular kind of operation. It takes time to become familiar with BASIC and its potential. You can learn a lot by reading other people's programs, trying to follow the flow of control and determine the outcome of the processes. But the best route to knowledge and mastery is through writing programs of your own, constantly trying out new ways of doing things. Practice makes programmers.

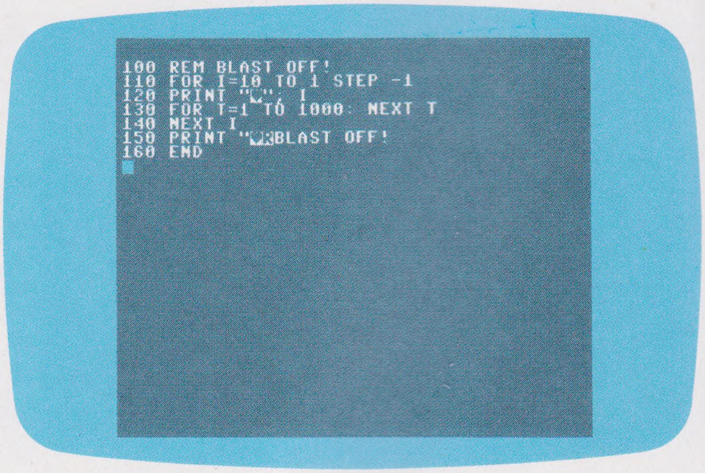
The program that produces this design uses one segment to create a house shape, another to create a tree shape. These segments, called subroutines, can generate as many of the shapes as required; you change the colors by assigning new values to variables used within the subroutines.

A Loop to Repeat a Set of Procedures



```
100 REM FOR...NEXT DEMO
110 INPUT "YOUR NAME"; NAME$
120 FOR I=1 TO 5
130 PRINT NAME$ " ";
140 NEXT I
150 END
RUN
YOUR NAME? LINDA
LINDA LINDA LINDA LINDA LINDA
READY.
LIST 120

120 FOR I=1 TO N

READY.
115 INPUT "HOW MANY"; N
RUN
YOUR NAME? EDWARD
HOW MANY? 3
EDWARD EDWARD EDWARD
READY.
█
```

The *FOR* statement in line 120 and the *NEXT* statement in line 140 cause the *PRINT* statement on line 130 to be repeated five times, displaying the name of the user with each repetition. By changing the *FOR* statement on line 120 so that the end value of the loop is set by the variable **N**, then adding line 115 to assign a value to **N**, you change the loop so that it repeats only as many times as you want it to.



```
100 REM BLAST OFF!
110 FOR I=10 TO 1 STEP -1
120 PRINT "◣"; I
130 FOR T=1 TO 1000: NEXT T
140 NEXT I
150 PRINT "◣◣BLAST OFF!
160 END
█
```

You can increase or decrease the variable in a *FOR . . . NEXT* loop by using a *STEP* expression, as shown in line 110; in this case, the variable decreases by 1 on each pass through the loop. Line 130 has another loop "nested" within the first; here both statements of the inner loop are on one line, separated by a colon. This loop creates a time delay of about a second, as the computer counts to 1,000. Always be sure that the *NEXT* statement of an inner loop precedes that of the outer loop.

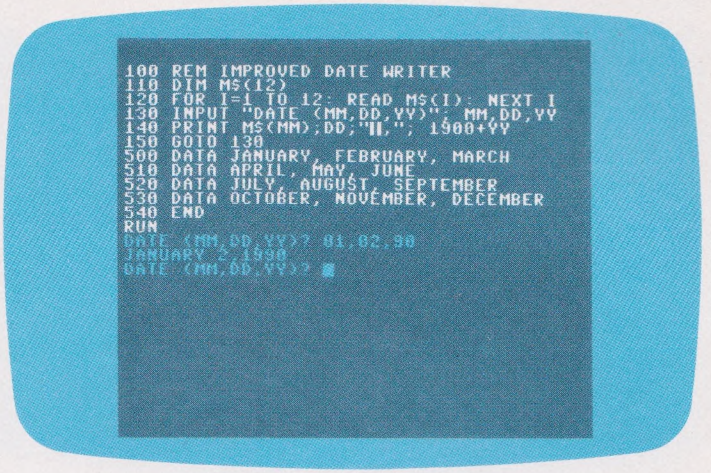
A computer excels at jobs that call for repeating a process many times. You can make use of this power in BASIC by bracketing a procedure between two special statements — *FOR* and *NEXT* — that make BASIC execute a loop, carrying out the procedure as many times as you specify.

A *FOR* statement tells BASIC that it is to do a job a given number of times, as specified by the values assigned to a variable in the statement. The variable in a *FOR* statement is used as a counter; BASIC stores the initial value of the variable in memory, then executes the subsequent instructions until it reaches a *NEXT* statement. BASIC then branches back to the *FOR* statement, adds the specified amount to the counter variable, and checks to see if the value exceeds the specified final value. BASIC repeats the loop until the counter exceeds the final value, then branches directly to the statement that follows the *NEXT* statement.

The power of the *FOR . . . NEXT* loop lies in the way it can process a different set of values at each repetition. A good example of its use is in the preparation of a long chart. The loop might contain complex instructions for calculating and displaying data, but you would need to write the instructions only once, no matter how many lines were required to complete the chart.

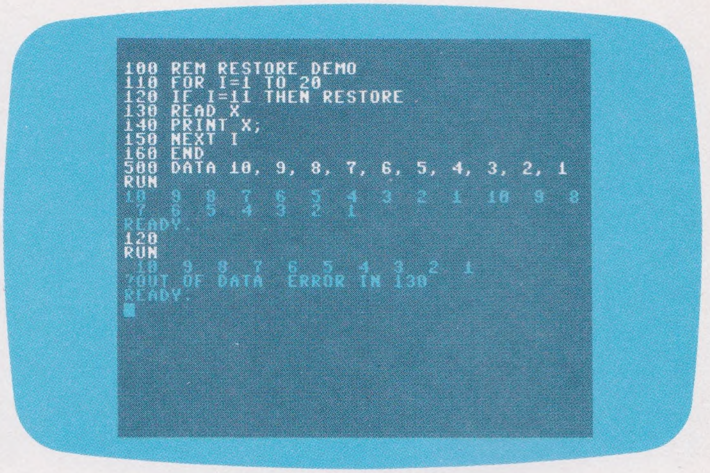
Storing and Retrieving Data

The *READ* statement in line 120 causes BASIC to look for the first *DATA* statement (line 500) and assign the first value it finds there to the array variable **M$**. Because the *READ* statement is in a *FOR . . . NEXT* loop, it is executed 12 times, each time reading the next item in the *DATA* statement. The remainder of the program is the same as the *Date Writer* program shown on page 81.



```
100 REM IMPROVED DATE WRITER
110 DIM M$(12)
120 FOR I=1 TO 12: READ M$(I): NEXT I
130 INPUT "DATE (MM,DD,YY)"; MM,DD,YY
140 PRINT M$(MM);DD;"█,"; 1900+YY
150 GOTO 130
500 DATA JANUARY, FEBRUARY, MARCH
510 DATA APRIL, MAY, JUNE
520 DATA JULY, AUGUST, SEPTEMBER
530 DATA OCTOBER, NOVEMBER, DECEMBER
540 END
RUN
DATE (MM,DD,YY)? 01,02,90
JANUARY 2,1990
DATE (MM,DD,YY)? █
```

You can reuse the values in a *DATA* statement by using a *RESTORE* statement, as shown in line 120. The *RESTORE* statement resets the pointer to the first *DATA* element in the program, in this case after 10 elements have been read. Removing the *RESTORE* statement by eliminating line 120, as shown here, results in an **?OUT OF DATA ERROR** when the BASIC tries to read past the end of the *DATA* statement.



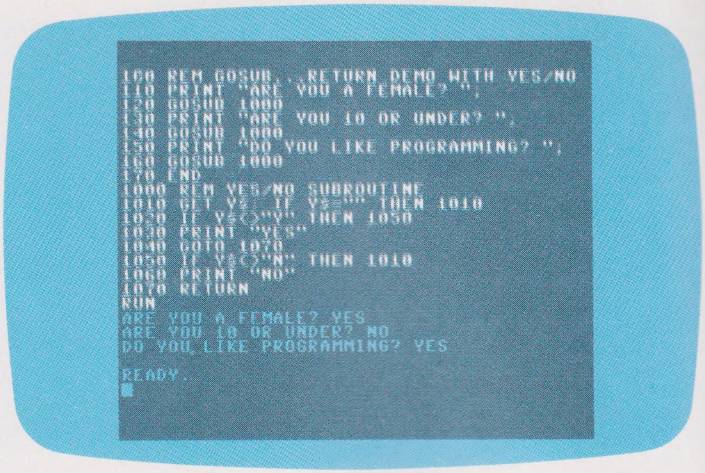
```
100 REM RESTORE DEMO
110 FOR I=1 TO 20
120 IF I=11 THEN RESTORE
130 READ X
140 PRINT X;
150 NEXT I
160 END
500 DATA 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
RUN
 10   9   8   7   6   5   4   3   2   1   10   9   8
 7   6   5   4   3   2   1
READY.
120
RUN
 10   9   8   7   6   5   4   3   2   1
?OUT OF DATA  ERROR IN 130
READY.
█
```

Many programs process information that does not change from one run to the next. It is useful to keep this data permanently attached to the program. You could assign the data values to variables, using *LET* statements, but this would be a cumbersome procedure if you had more than a few items. BASIC provides a much easier way to deal with this situation, using a pair of statements called *READ* and *DATA*.

A *READ* statement tells BASIC to establish one or more variables, and then to fill them with values taken sequentially from a *DATA* statement. The first *READ* statement in a program draws on the first *DATA* statement, wherever it may appear. BASIC takes as many values from the *DATA* statement as are required to fill the variables of the *READ* statement. By leaving a pointer in the *DATA* statement, BASIC keeps track of the values that have already been used; the next time a *READ* statement looks at the *DATA* statement, it starts with the first unused values.

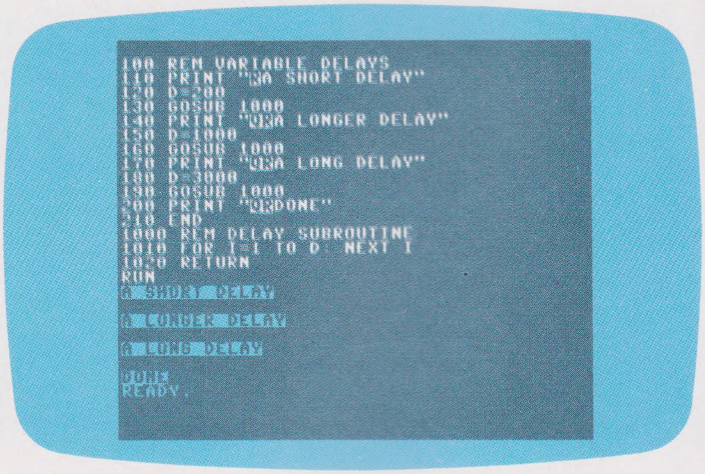
When BASIC comes to the end of one *DATA* statement and still needs values to fill the variables of a *READ* statement, it moves to the next *DATA* statement and continues to read. If it runs out of data before all variables are filled, BASIC stops running the program and displays an error message. One way to avoid this problem is shown in the example in the screen immediately above.

Subroutines to Be Called When Needed



```
100 REM GOSUB...RETURN DEMO WITH YES/NO
110 PRINT "ARE YOU A FEMALE? ";
120 GOSUB 1000
130 PRINT "ARE YOU 10 OR UNDER? ",
140 GOSUB 1000
150 PRINT "DO YOU LIKE PROGRAMMING? ";
160 GOSUB 1000
170 END
1000 REM YES/NO SUBROUTINE
1010 GET Y$: IF Y$="" THEN 1010
1020 IF Y$<>"Y" THEN 1050
1030 PRINT "YES"
1040 GOTO 1070
1050 IF Y$<>"N" THEN 1010
1060 PRINT "NO"
1070 RETURN
RUN
ARE YOU A FEMALE? YES
ARE YOU 10 OR UNDER? NO
DO YOU LIKE PROGRAMMING? YES

READY.
█
```

The subroutine that begins at line 1000 takes a one-letter user response from the keyboard and displays the corresponding word, **YES** or **NO**. It is called three times from the body of the program, by the *GOSUB* statements on lines 120, 140 and 160. Each time the subroutine has completed its job, the *RETURN* statement on line 1070 branches the program back to the statement after the *GOSUB* that called the subroutine.



```
100 REM VARIABLE DELAYS
110 PRINT "▨A SHORT DELAY"
120 D=200
130 GOSUB 1000
140 PRINT "▨▨A LONGER DELAY"
150 D=1000
160 GOSUB 1000
170 PRINT "▨▨A LONG DELAY"
180 D=3000
190 GOSUB 1000
200 PRINT "▨▨DONE"
210 END
1000 REM DELAY SUBROUTINE
1010 FOR I=1 TO D: NEXT I
1020 RETURN
RUN
A SHORT DELAY

A LONGER DELAY

A LONG DELAY

DONE
READY.
```

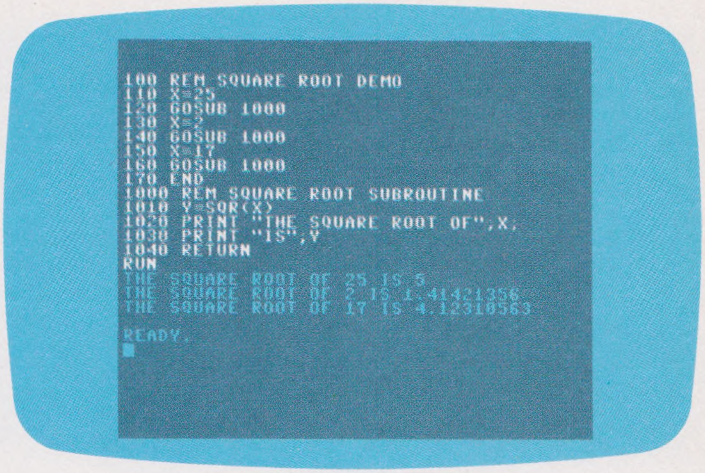
The subroutine that begins at line 1000 causes a delay in program execution. The length of each delay is set by the value you assign to the variable **D** before the subroutine is called by a *GOSUB* statement in the body of the program. Assigning a larger number to the variable **D** makes the computer go through the *FOR . . . NEXT* loop in the subroutine more times, thus increasing the duration of the pause.

Many programming situations require that you repeat a particular set of steps at several different points within the program. Instead of writing the same set of instructions again and again, you can write the code once in a sort of miniature program called a subroutine, then tell BASIC to branch to the subroutine each time you need it.

This procedure, known as calling a subroutine, is made possible by the *GOSUB* and *RETURN* statements. A *GOSUB* tells BASIC to branch to the subroutine beginning at a specified line, then execute all subsequent lines until it encounters a *RETURN* statement. The *RETURN* tells BASIC to branch back to the statement following the *GOSUB* that called the subroutine. You can call a subroutine any number of times, from anywhere in the program.

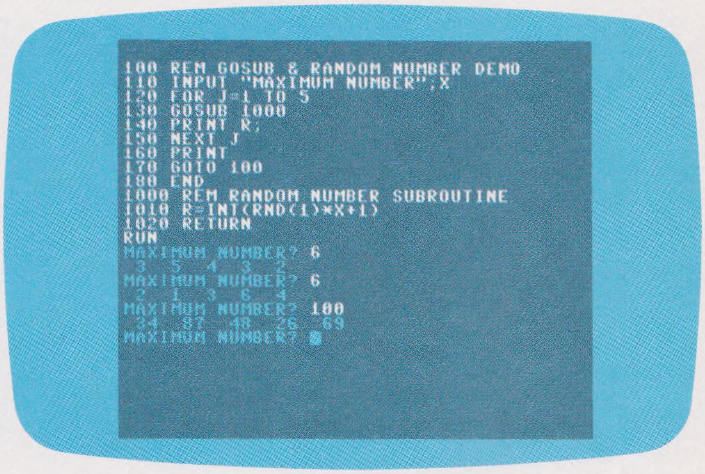
You must place any subroutine after the *END* statement of a program; otherwise BASIC would execute the

In this program the subroutine first calculates and then displays the square root of a number assigned to the variable **X**. The numeric function *SQR* on line 1010 computes the square root and assigns it to the variable **Y**, which is displayed by the *PRINT* statement on line 1030. A subroutine can perform operations on variables received from the body of the program, as in this case; it can also pass variables back to the body of the program for further processing.



```
100 REM SQUARE ROOT DEMO
110 X=25
120 GOSUB 1000
130 X=2
140 GOSUB 1000
150 X=17
160 GOSUB 1000
170 END
1000 REM SQUARE ROOT SUBROUTINE
1010 Y=SQR(X)
1020 PRINT "THE SQUARE ROOT OF";X;
1030 PRINT "IS";Y
1040 RETURN
RUN
THE SQUARE ROOT OF 25 IS 5
THE SQUARE ROOT OF 2 IS 1.41421356
THE SQUARE ROOT OF 17 IS 4.12310563

READY.
█
```

Here a subroutine uses the variable **X** as the limit for a routine that produces a random whole number. The number is assigned to a variable (**R**) that is passed back to the body of the program to be displayed by the print statement on line 140. The randomizing routine on line 1010 uses two numeric functions, *INT* and *RND*, to assign a value to **R**. The expression **RND(1)** generates a random decimal fraction between 0 and 1. This is multiplied by the value of **X** and then increased by 1, to ensure that the result will be greater than 1. The *INT* function then produces a whole number based upon the product of the calculation.



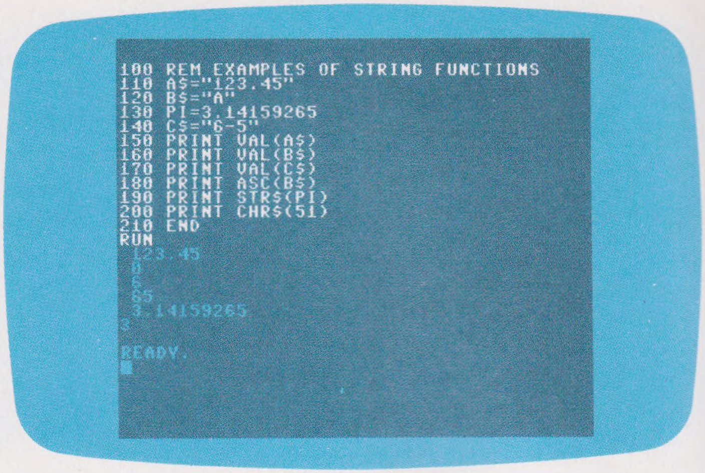
```
100 REM GOSUB & RANDOM NUMBER DEMO
110 INPUT "MAXIMUM NUMBER";X
120 FOR J=1 TO 5
130 GOSUB 1000
140 PRINT R;
150 NEXT J
160 PRINT
170 GOTO 100
180 END
1000 REM RANDOM NUMBER SUBROUTINE
1010 R=INT(RND(1)*X+1)
1020 RETURN
RUN
MAXIMUM NUMBER? 6
 3  5  4  3  2
MAXIMUM NUMBER? 6
 2  1  3  6  4
MAXIMUM NUMBER? 100
 34  87  48  26  69
MAXIMUM NUMBER? █
```

subroutine when going sequentially through the numbered lines of the program. If BASIC encounters a *RETURN* statement without a preceding *GOSUB,* it will stop the program and display an error message.

Using subroutines whenever possible can make programming much easier for you. Each subroutine can be tested and modified in isolation before it is added to the main program. And you can keep your program structure easy to understand by numbering the subroutines so that each starts with a line number that is a multiple of 1,000.

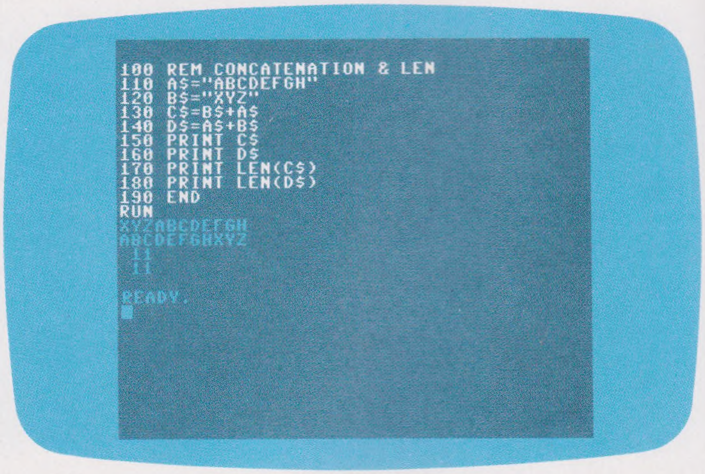
The examples shown on the two screens on this page incorporate special BASIC expressions called numeric functions. These functions are used to perform common mathematical operations; each is a short program in its own right, and it does the same job each time it is called. The *SQR* function simply produces the square root of a specified value. The *INT* function produces the largest integer that is less than or equal to a specified value. The *RND* function tells BASIC to generate a random number between 0 and 1; it is often used in game programs that require unpredictable actions. BASIC also has functions that perform trigonometric operations and others that do tasks common in accounting work. An appendix to the *Commodore 64 User's Guide* explains the functions and their uses.

Using Strings to Handle Characters



```
100 REM EXAMPLES OF STRING FUNCTIONS
110 A$="123.45"
120 B$="A"
130 PI=3.14159265
140 C$="6-5"
150 PRINT VAL(A$)
160 PRINT VAL(B$)
170 PRINT VAL(C$)
180 PRINT ASC(B$)
190 PRINT STR$(PI)
200 PRINT CHR$(51)
210 END
RUN
 123.45
 0
 6
 65
 3.14159265
3

READY.
█
```

The *PRINT* statements on lines 150 through 170 display the results of *VAL* functions that convert designated string values into numeric values. The numeric value of **A$** is displayed as **123.45**; the numeric value of **B$** is displayed as **0**, since **B$** contains no numbers. BASIC displays the numeric value of **C$** as **6**, since the *VAL* function does no computation and cannot convert characters such as the minus sign. The *STR$* function on line 190 returns a string with the numerals in the numeric variable **PI**. The *CHR$* function on line 200 returns the numeral **3** — the character represented by ASCII code 51.



```
100 REM CONCATENATION & LEN
110 A$="ABCDEFGH"
120 B$="XYZ"
130 C$=B$+A$
140 D$=A$+B$
150 PRINT C$
160 PRINT D$
170 PRINT LEN(C$)
180 PRINT LEN(D$)
190 END
RUN
XYZABCDEFGH
ABCDEFGHXYZ
 11
 11

READY.
█
```

The statements on lines 130 and 140 tell BASIC to concatenate string values, by adding the second one named to the end of the first. The results of these concatenations are displayed by *PRINT* statements on lines 150 and 160. On lines 170 and 180, *PRINT* statements display the results returned by the *LEN* functions, which count the characters in the designated string values.

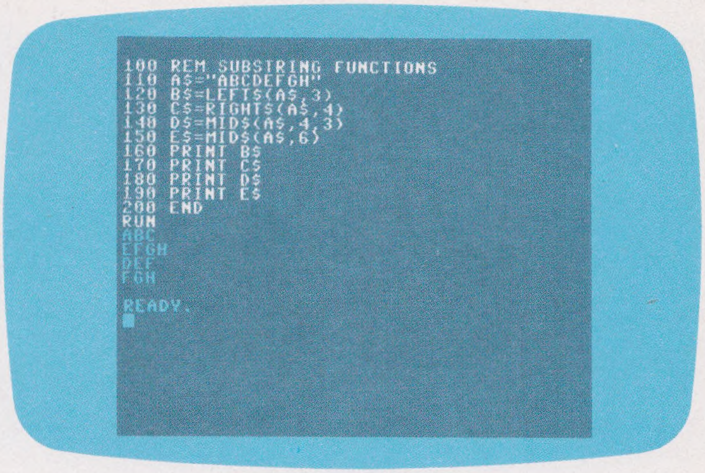
Many programming situations require BASIC to perform operations on strings of alphanumeric characters — for example, extracting the first letter of a name, or counting the letters in a word. Other situations call for similar operations on characters that are not ordinarily visible on the monitor screen, such as cursor- and color-control characters. BASIC provides a number of built-in string functions to do these operations. Like a numeric function *(page 9)*, each string function is a small program that does the same job — called returning a result — whenever it is used.

One string can be added to the end of another by a process called concatenation: You simply put a plus sign between the two strings. If the content of a string is numbers, remember that string concatenation is different from numeric addition, which also uses the plus sign. Concatenation of the strings **123** and **45** returns **12345**, not **168**.

You can find out the length of a character string by using the *LEN* function. *LEN* returns the number of characters in the string, including those not ordinarily visible on the screen when the program is run.

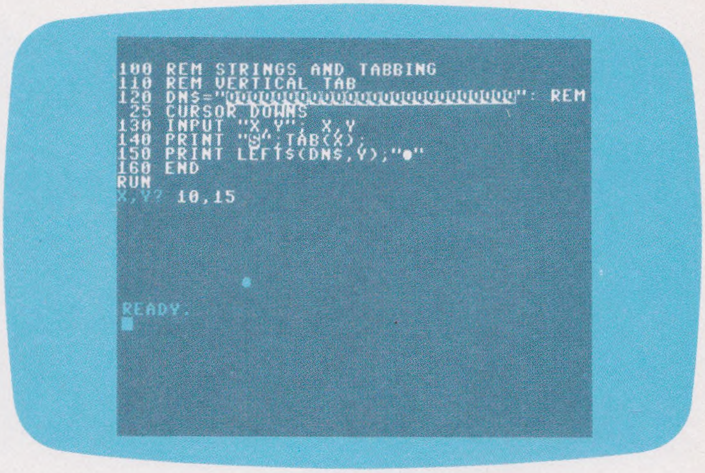
Three substring functions allow you to extract different parts — called substrings — from specified strings for use in other parts of the program. These functions — *LEFT$, MID$* and *RIGHT$ (see top screen, opposite)* — copy the designated elements with-

The *PRINT* statements on lines 160 through 190 display the results of substring functions on lines 120 through 150. The *LEFT$* function on line 120 acts on the string value **A$**, counting three characters from the left and assigning them to **B$**. The *RIGHT$* function on line 130 counts four characters from the right of **A$**, assigning them to **C$**. The first *MID$* function takes characters from the middle of **A$** by assigning three characters to **D$**, starting with the fourth character in the string **A$**. The final *MID$* function, with no instruction on number of characters, starts with the sixth and uses all the remaining characters of the string **A$**.



```
100 REM SUBSTRING FUNCTIONS
110 A$="ABCDEFGH"
120 B$=LEFT$(A$,3)
130 C$=RIGHT$(A$,4)
140 D$=MID$(A$,4,3)
150 E$=MID$(A$,6)
160 PRINT B$
170 PRINT C$
180 PRINT D$
190 PRINT E$
200 END
RUN
ABC
EFGH
DEF
FGH

READY.
█
```

You can overcome the C-64's lack of a vertical tabulating function with a program like this one, which moves the cursor to the designated location. A string called **DN$** on line 120 contains 25 "cursor-down" characters. The *INPUT* statement on line 130 takes the designation of cursor placement by column and line. Line 140 sets the horizontal position using the *TAB* function. Line 150 sets the vertical position by applying a *LEFT$* function to the string **DN$** and printing the number of cursor-down characters specified by the variable **Y**. When the program runs, the cursor moves to the designated point and then prints the shape.



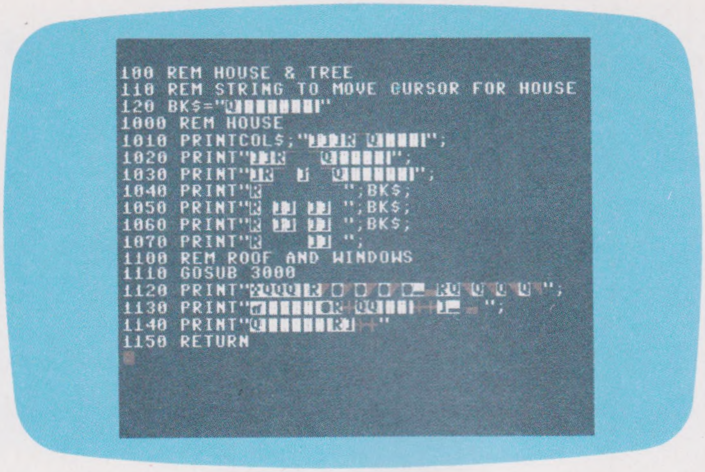
```
100 REM STRINGS AND TABBING
110 REM VERTICAL TAB
120 DN$="QQQQQQQQQQQQQQQQQQQQQQQQQ": REM
 25 CURSOR DOWNS
130 INPUT "X,Y"; X,Y
140 PRINT "S";TAB(X);
150 PRINT LEFT$(DN$,Y);"●"
160 END
RUN
X,Y? 10,15





         ●

READY.
█
```

out changing the strings themselves; they cannot be used to insert new material into the original strings.

Two other functions serve to convert string values to numeric values, and vice versa. The *STR$* function turns a numeric expression into a string of numeric characters; the *VAL* function returns the numeric value of the characters in a string expression. If the first character in the specified expression is not a numeral or a plus or minus sign, the *VAL* function returns a numeric value of zero.

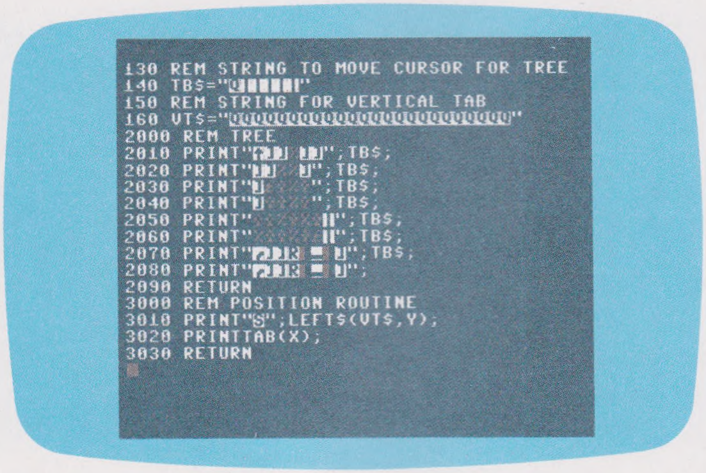
Each character used by the C-64 is also known by a unique number, which is part of a numbering system called ASCII (for American Standard Code for Information Interchange, and pronounced "AS-key"). These ASCII codes range from zero to 255. Two BASIC functions allow you to convert characters to their ASCII values, and vice versa. The *ASC* function returns the ASCII code for a designated character; the *CHR$* function returns the character represented by a designated ASCII code. Some characters, such as *delete* or *carriage return,* do not appear as images on the monitor screen; you can include them in strings only by using a *CHR$* expression. The ASCII codes for the characters of the C-64 can be found in an appendix to the *Commodore 64 User's Guide.*

Building Graphics from the Keyboard



```
100 REM HOUSE & TREE
110 REM STRING TO MOVE CURSOR FOR HOUSE
120 BK$="Q▌▌▌▌▌▌▌▌"
1000 REM HOUSE
1010 PRINTCOL$;"]]]R Q▌▌▌▌▌";
1020 PRINT"]]R    Q▌▌▌▌▌";
1030 PRINT"]R  ]  Q▌▌▌▌▌▌▌";
1040 PRINT"R          ";BK$;
1050 PRINT"R ]] ]] ";BK$;
1060 PRINT"R ]] ]] ";BK$;
1070 PRINT"R     ]] ";
1100 REM ROOF AND WINDOWS
1110 GOSUB 3000
1120 PRINT"£QQQR▨▨▨▨_RQ▨Q▨Q▨Q▨";
1130 PRINT"▌▌▌▌▌▌▌QR┼QQ▌▌▌┼┼]_▬ ";
1140 PRINT"Q▌▌▌▌▌▌▌▌R]┼┼"
1150 RETURN
```

The first part of a program that draws a simple landscape *(lower screen, opposite)* consists of cursor controls *(line 120)* and a subroutine at line 1000 that builds the house. Graphic-symbol characters are shown here in gray; the other characters represent cursor and color controls. The subroutine called by line 1110 puts the cursor at its starting point to begin work on the roof.



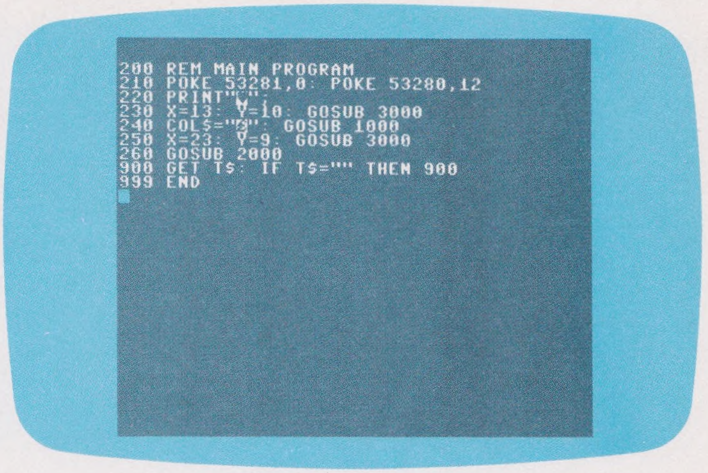
```
130 REM STRING TO MOVE CURSOR FOR TREE
140 TB$="Q▌▌▌▌▌▌"
150 REM STRING FOR VERTICAL TAB
160 VT$="QQQQQQQQQQQQQQQQQQQQQQQQQ"
2000 REM TREE
2010 PRINT"£]] ]]";TB$;
2020 PRINT"]]▒▒]";TB$;
2030 PRINT"]▒▒▒▒";TB$;
2040 PRINT"]▒▒▒▒";TB$;
2050 PRINT"▒▒▒▒▒▌▌";TB$;
2060 PRINT"▒▒▒▒▒▌▌";TB$;
2070 PRINT"£]R▒▬▒]";TB$;
2080 PRINT"£]R▒▬▒]";
2090 RETURN
3000 REM POSITION ROUTINE
3010 PRINT"S";LEFT$(VT$,Y);
3020 PRINTTAB(X);
3030 RETURN
```

Two more subroutines provide the *PRINT* statements that draw the tree and reposition the cursor. Lines 140 and 160, like line 120 above, assign cursor-control characters to variables, to be used for moving the cursor in the subroutines *(pages 92-93)*.

The ability to produce colorful graphics is one of the 64's strong suits, and BASIC gives you several ways to create your own images on the screen. The easiest way is to use the character-graphics mode, as shown in the sample program on the screens above. It lets you use the C-64's extensive set of characters and graphic symbols as building blocks, assembling them in different patterns using *PRINT* statements. In this mode, you can change the shape and color that fill each of the 1,000 *PRINT* positions on the monitor.

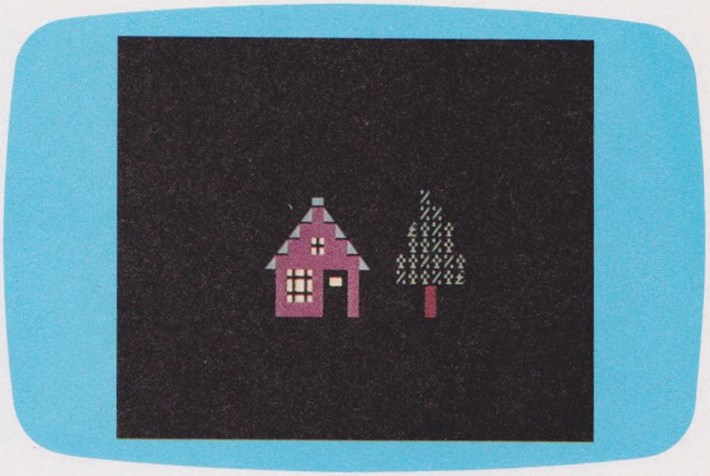
Another, more complicated way is to build your images point by point, in what is called the bit-mapping mode. In this mode, you use BASIC programs to define the color of any of the 64,000 pixels on the screen. The key to bit-mapped graphics is the *POKE* statement, which lets you put numeric values into specific memory locations that control the screen. Bit-mapped graphics are be-

With the subroutines written, the main body of the program is simple and short. Line 210 sets the screen and border colors with *POKE* statements; line 220 clears the screen. Line 230 sets the coordinates for the starting point of the house, and calls the cursor-positioning subroutine. Line 240 assigns a color-control character to **COL$** and calls the house subroutine. Line 250 sets the coordinates for the tree, and line 260 calls the tree subroutine. Line 900 freezes the screen by causing the program to loop until a key is pressed; line 999 ends the program.



```
200 REM MAIN PROGRAM
210 POKE 53281,0: POKE 53280,12
220 PRINT"♥":
230 X=13: Y=10: GOSUB 3000
240 COL$="▩": GOSUB 1000
250 X=23: Y=9: GOSUB 3000
260 GOSUB 2000
900 GET T$: IF T$="" THEN 900
999 END
```

This screen is the product of the completed program. To add more houses or trees, you could add more positioning statements and subroutine calls to the main body of the program. You could also add other elements to the screen by writing new subroutines for shapes of your own design.



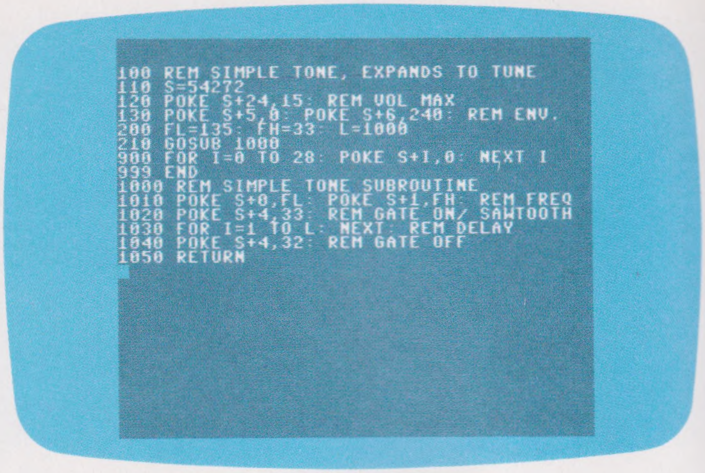
yond the scope of this introduction to BASIC, but the popularity of the C-64 as a graphics tool has given rise to a number of books and courses on the subject, which you can use to pursue the subject further.

Using character graphics need not limit creativity: You can put any of more than 100 shapes into any *PRINT* position on the screen, in any of 16 colors. You can create graphs and game boards using the many horizontal, vertical and diagonal lines, with corners and intersections, offered by the graphics character set *(chart, page 32).* There are also balls, blocks and card symbols to spice up your graphics. And there are other forms that are not shown on the keyboard — lower-case letters and reverse images of the graphic and alphanumeric characters.

Putting all these elements together lets you make images like the ones in the sample program above. The trick is in getting exactly the right sequences of characters into the *PRINT* statements, since the source code of the program is little help in showing what will take place when you run the program. As you begin to develop graphics programs, you may find it easiest to first make the shapes on the screen in the immediate mode — without line numbers — making a note of each keystroke you use. Then you can write the program, putting the same sequences of keystrokes into the *PRINT* statements.

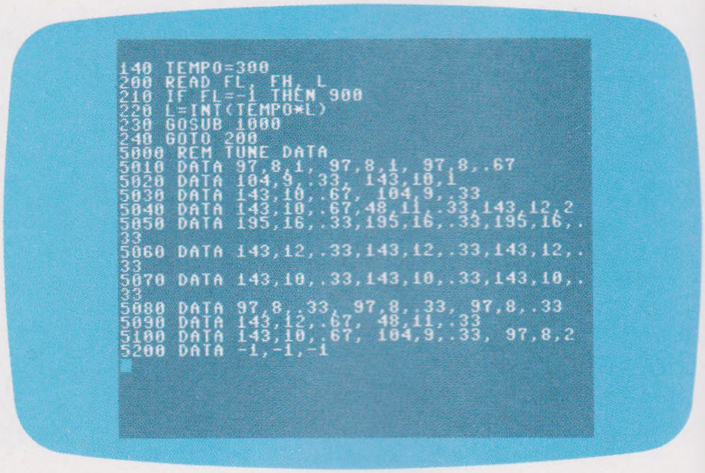
Sound in BASIC: a Treasury of Effects

This program sets the values for one note in one voice. The address of the first register is assigned to the variable **s** in line 110; all of the *POKE* statements use this as a base value. Line 120 sets the volume register; line 130 sets the attack/decay and sustain/release registers. Line 200 assigns values to variables for low frequency, high frequency and length of the note; line 210 calls the subroutine that sets the frequency (line 1010), starts the note (**gate on**, line 1020), holds it (line 1030) and ends it (**gate off**, line 1040). Line 900 resets all registers to zero.



```
100 REM SIMPLE TONE, EXPANDS TO TUNE
110 S=54272
120 POKE S+24,15: REM VOL MAX
130 POKE S+5,0: POKE S+6,240: REM ENV.
200 FL=135: FH=33: L=1000
210 GOSUB 1000
900 FOR I=0 TO 28: POKE S+I,0: NEXT I
999 END
1000 REM SIMPLE TONE SUBROUTINE
1010 POKE S+0,FL: POKE S+1,FH: REM FREQ
1020 POKE S+4,33: REM GATE ON/ SAWTOOTH
1030 FOR I=1 TO L: NEXT: REM DELAY
1040 POKE S+4,32: REM GATE OFF
1050 RETURN
```

Adding these lines to the program shown above lets you use the subroutine to play a tune. The new line 200 is a *READ* statement that assigns values from the *DATA* statements to the frequency and duration variables. The tempo in line 140 is multiplied by the duration value, to produce an end value for the *FOR . . . NEXT* loop on line 1030, which holds each note.



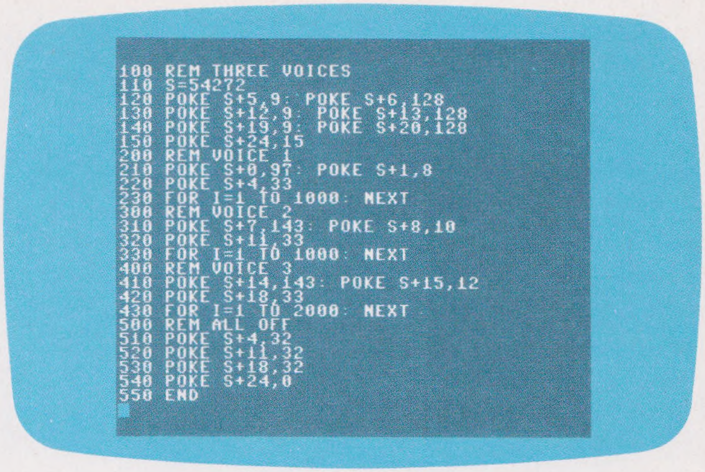
```
140 TEMPO=300
200 READ FL, FH, L
210 IF FL=-1 THEN 900
220 L=INT(TEMPO*L)
230 GOSUB 1000
240 GOTO 200
5000 REM TUNE DATA
5010 DATA 97,8,1, 97,8,1, 97,8,.67
5020 DATA 104,9,.33, 143,10,1
5030 DATA 143,10,.67, 104,9,.33
5040 DATA 143,10,.67,48,11,.33,143,12,2
5050 DATA 195,16,.33,195,16,.33,195,16,.
33
5060 DATA 143,12,.33,143,12,.33,143,12,.
33
5070 DATA 143,10,.33,143,10,.33,143,10,.
33
5080 DATA 97,8,.33, 97,8,.33, 97,8,.33
5090 DATA 143,12,.67, 48,11,.33
5100 DATA 143,10,.67, 104,9,.33, 97,8,2
5200 DATA -1,-1,-1
```

To use BASIC to harness the considerable sound capabilities of your C-64, you will have to master some of the intricacies of the Sound Interface Device, or SID chip. This chip has a large number of storage and work areas called registers; you will use *POKE* statements to put into the registers the numbers that will determine the sounds generated by the chip.

You start a sound by putting a number into a location called the control register. But before you do that, you must decide exactly how you want the note to sound — its pitch, how its volume changes during the time the note is played, and whether it should sound like a particular instrument, or just plain noise.

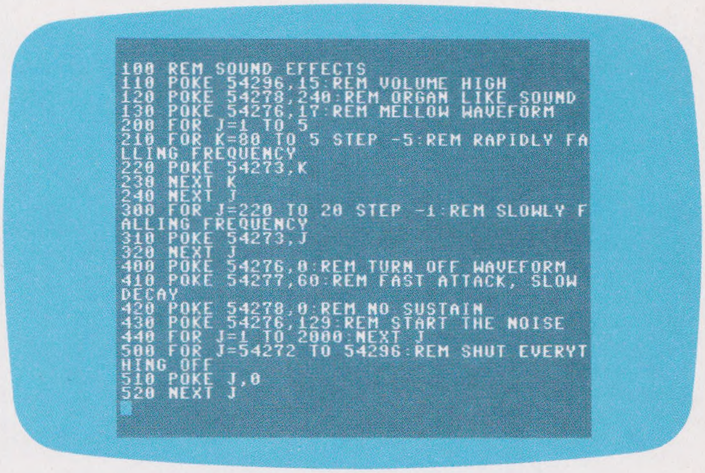
Each attribute of a note is set by putting numbers into a register. Two numbers determine the pitch of the note: One number in what is called the low-frequency register, and one in the high-frequency register. Another number sets the attack/decay register, which governs the initial levels of the sound — that is, the rate at which it rises to its maximum volume, and the rate of fall to the volume it will sustain until it is turned off. Yet another number sets the sustain/release register, which governs the volume at which the sound is held, and the rate at which it returns to silence when it is turned off.

The number you put into the control register to turn the sound on also governs the waveform. Four wave-



```
100 REM THREE VOICES
110 S=54272
120 POKE S+5,9: POKE S+6,128
130 POKE S+12,9: POKE S+13,128
140 POKE S+19,9: POKE S+20,128
150 POKE S+24,15
200 REM VOICE 1
210 POKE S+0,97: POKE S+1,8
220 POKE S+4,33
230 FOR I=1 TO 1000: NEXT
300 REM VOICE 2
310 POKE S+7,143: POKE S+8,10
320 POKE S+11,33
330 FOR I=1 TO 1000: NEXT
400 REM VOICE 3
410 POKE S+14,143: POKE S+15,12
420 POKE S+18,33
430 FOR I=1 TO 2000: NEXT
500 REM ALL OFF
510 POKE S+4,32
520 POKE S+11,32
530 POKE S+18,32
540 POKE S+24,0
550 END
```

This program uses all three of the voices in a C major chord. The registers for Voice 1 start at 54272, or S+0. The registers for Voice 2 start at S+7, and for Voice 3, at S+14. Frequencies are programmed in lines 110 through 140, then each voice is switched on in turn. The statements beginning at line 500 switch the voices off in turn; line 540 turns the volume level to zero.



```
100 REM SOUND EFFECTS
110 POKE 54296,15:REM VOLUME HIGH
120 POKE 54278,240:REM ORGAN LIKE SOUND
130 POKE 54276,17:REM MELLOW WAVEFORM
200 FOR J=1 TO 5
210 FOR K=80 TO 5 STEP -5:REM RAPIDLY FA
LLING FREQUENCY
220 POKE 54273,K
230 NEXT K
240 NEXT J
300 FOR J=220 TO 20 STEP -1:REM SLOWLY F
ALLING FREQUENCY
310 POKE 54273,J
320 NEXT J
400 POKE 54276,0:REM TURN OFF WAVEFORM
410 POKE 54277,60:REM FAST ATTACK, SLOW
DECAY
420 POKE 54278,0:REM NO SUSTAIN
430 POKE 54276,129:REM START THE NOISE
440 FOR J=1 TO 2000:NEXT J
500 FOR J=54272 TO 54296:REM SHUT EVERYT
HING OFF
510 POKE J,0
520 NEXT J
```

This program generates three different sounds in rapid sequence, to create a "space war" effect. Lines 110 through 130 set all but the frequency registers of the first two sounds. Lines 200 through 240 create a burst of five quickly falling tones; lines 300 through 320 create a longer falling tone. Lines 400 through 440 reset the registers and establish a different waveform — random noise — to create the final explosion-like sound. Lines 500 through 520 shut off the sound by filling each register with a zero.

forms are available. A triangular wave produces a nearly pure sound, without overtones. A sawtooth wave gives a sound like a stringed instrument, and a pulsed wave is similar to the sound of a clarinet. The final waveform produces random noise. For each waveform number you use to turn on the sound, there is a corresponding number you poke into the control register when you want to turn the sound off.

The C-64 has three separate voices, each governed by its own set of registers, so that you can program three notes to play simultaneously. A single volume register sets the overall volume for all three voices; the attack/decay and sustain/release registers set proportions of this master volume for each voice.

The registers for each voice are numbered sequentially, which makes programming easier; you can assign the address of the first register to a variable, then refer to each of the other registers by adding a number to the variable.

This brief introduction to programming sound in BASIC only scratches the surface of the subject. The examples on the screens above illustrate the way the system works. The addresses of the registers of the SID chip and the numbers that set each register are listed in the *Commodore 64 User's Guide;* you can use them, and the examples shown here, as a guide for further exploration.

Appendix: Useful BASIC Commands

COMMAND	COMMAND FORMAT	ABBREVIATION OF COMMAND	PAGES
CLOSE	CLOSE 4	CL┌ 4	44-45
COMMAND	CMD 4	C╲ 4	44-45
COPY	OPEN 15,8,15 PRINT#15,"C0:NEWFILE=0:OLDFILE" CLOSE 15	NO ABBREVIATION	42
DIMENSION AN ARRAY	110 DIM M$(12) ⋮ 120 M$(1)="JANUARY":M$(2)="FEBRUARY"	110 D╮ M$(12) ⋮ 120 M$(1)="JANUARY":M$(2)="FEBRUARY"	80-81
END	130 END	130 E/	74-75
FOR . . . NEXT	120 FOR I=1 TO 5 ⋮ 140 NEXT I	120 F┌ I=1 TO 5 ⋮ N‾ I	88
GET	120 GET A$	120 G‾ A$	78-79
GOSUB . . . RETURN	120 GOSUB 1000 ⋮ 1070 RETURN	120 GO♥ 1000 ⋮ 1070 RE|	90-91
GOTO	140 GOTO 100	140 G┌ 110	82-83
IF . . . THEN	IF N=7 THEN 150	IF N=7 T| 150	82-83
INITIALIZE	OPEN 15,8,15 PRINT#15,"I0" CLOSE 15	NO ABBREVIATION	42
INPUT	110 INPUT "YOUR NAME, PLEASE"; NAME $	NO ABBREVIATION	78-79
LET	100 LET N%=76	100 L‾ N%=76	80-81
LIST	LIST	L╮	40-41
LOAD	LOAD "SAMPLE NAME PROGRAM",8	L┌ "SAMPLE NAME PROGRAM",8	40-42
NEW	NEW	NO ABBREVIATION	74-75

This chart lists the most common BASIC commands, with examples of how they are used. Abbreviated forms are also listed; these usually consist of the first letter of the command plus the *shifted* second (or third) letter, which appears as a graphic symbol. The examples preceded by line numbers indicate commands that are usually used in the program mode; commands used in the immediate mode are shown without line numbers. Page numbers refer to the pages where these commands are explained.

COMMAND	COMMAND FORMAT	ABBREVIATION OF COMMAND	PAGES
NEW (disk)	OPEN 15,8,15 PRINT#15, "N0:PRACTICE,P1" CLOSE 15	NO ABBREVIATION	40-42
ON . . . GOTO	120 ON N GOTO 150, 170, 190	120 ON N G┌ 150, 170, 190	82-83
OPEN	OPEN 4,4	O┐ 4,4	44-45
POKE	110 POKE 53281,1: REM WHITE SCREEN	110 P┌ 53281,1: REM WHITE SCREEN	34, 84-85, 96
PRINT	110 PRINT "ABC"	110 ?"ABC"	74-75
PRINT#	PRINT#4, "THIS APPEARS ON THE PRINTER"	P▁4,"THIS APPEARS ON THE PRINTER"	44-45
READ . . . DATA	130 READ X 5̇0̇0̇ DATA 10, 9, 8, 7, 6, 5, 4, 3, 2, 1	130 R─ X 5̇0̇0̇ D♠ 10, 9, 8, 7, 6, 5, 4, 3, 2, 1	89
REMARK	100 REM ABC PROGRAM	NO ABBREVIATION	74-75
RENAME	OPEN 15,8,15 PRINT#15,"R0:NEWNAME=0:OLDNAME" CLOSE 15	NO ABBREVIATION	42
RESTORE	120 IF I=11 THEN RESTORE	120 IF I=11 THEN RE♥	88-89
RUN	RUN	R╭	74-75
SAVE	SAVE "SAMPLE NAME PROG",8	S♠ "SAMPLE NAME PROG",8	40-42
SCRATCH	OPEN 15,8,15 PRINT#15,"S0:FILENAME" CLOSE 15	NO ABBREVIATION	42
STEP	120 FOR I=10 TO 1 STEP -1 1̇4̇0̇ NEXT I	120 F┌ I=10 TO 1 ST─ -1 1̇4̇0̇ N─ I	88-89
VALIDATE	OPEN 15,8,15 PRINT#15,"V0" CLOSE 15	NO ABBREVIATION	42

Bibliography

Books

Boom, Michael:
Commodore 64: A User's Guide to Expansion, Applications, and Maintenance. Alfred Publishing, 1983.
How to Use the Commodore 64. Alfred Publishing, 1983.

Carlson, Edward H., *Kids and the Commodore 64.* Datamost, 1983.

Commodore 64 Programmer's Reference Guide. Commodore Business Machines and Howard W. Sams & Co., 1982.

Commodore 64 User's Guide. Commodore Business Machines and Howard W. Sams & Co., 1982.

Compute!'s First Book of Commodore 64 Sound and Graphics. Compute! Publications, 1983.

Darcy, Laura, and Louise Boston (compilers), *Webster's New World Dictionary of Computer Terms.* Simon & Schuster, 1983.

Editors of *Consumer Guide:*
The Best VIC/Commodore Software. Publications International, 1984.
The User's Guide to the Commodore 64 & VIC 20: Computers, Software, & Peripherals. Publications International, 1983.

Heilborn, John, and Ran Talbott, *Your Commodore 64: A Guide to the Commodore 64 Computer.* Osborne/McGraw-Hill, 1983.

Held, Gilbert, *Commodore 64 BASIC: Quick Reference Guide.* Wiley, 1983.

Hergert, Douglas, *The Commodore 64/VIC-20 BASIC Handbook.* Sybex, 1983.

Kascmer, Joseph, *The Easy Guide to Your Commodore 64.* Sybex, 1983.

Onosko, Tim, *Commodore 64: Getting the Most from It.* Brady, 1983.

Osborne, Adam, Jim Strasma and Ellen Strasma, *PET Personal Computer Guide.* Osborne/McGraw-Hill, 1982.

Sanders, William B., *The Elementary Commodore 64.* Datamost, 1983.

West, Raeto, *Programming the PET/CBM.* Compute! Books, 1982.

Willis, Jerry, Merl Miller and Deborrah Willis, *Things to Do with Your Commodore 64 Computer.* New American Library, 1983.

Willis, Jerry, and Deborrah Willis, *How to Use the Commodore 64 Computer.* dilithium Press, 1984.

Magazine Articles

Peele, Gregg, "Understanding Sound on the Commodore 64." *Compute!'s Gazette,* October 1983.

"Sprites Made Easy for the Commodore 64." *Compute!'s Gazette,* December 1983.

Magazines

Available on newsstands:
Commodore: The Microcomputer Magazine
Compute!
Compute!'s Gazette
Available by subscription:
The Midnite Paper Software Gazette. 635 Maple, Mount Zion, Ill. 62549
The Torpet. Horning's Mills, Ont., Canada L0N 1J0

Picture Credits

Sources for all pictures in this book are shown below. Credits for the illustrations from top to bottom are separated by dashes.

The following photographs are by Larry Sherer: pages 3-8, 12-24, 33 bottom, 36, 46-56, 59 bottom, 61, 64-67, 72, 85 bottom, 86, 95 bottom.

All drawings are by Matt McMullen except the following: page 15: Frederic F. Bigio from B-C Graphics. 28 top: William J. Hennessy. 32: Frederic F. Bigio from B-C Graphics. 77: Walter Hilmers.

Computer screen images have been generated courtesy the following sources: pages 46, 48: Loren Wright. 49 top: Monopole, Toronto Pet Users Group. 50: Lemonade, Commodore Educational Software — Nuclear Power Plant, Commodore Educational Software. 51, 52: Loren Wright. 53: Stephen Murri. 54, 55: Loren Wright. 56: Vizastar 64, Viza Software Limited, Gillingham, England. 58, 59: Loren Wright. 61: Easy Calc, Commodore Business Machines. 63: The Mail Disk, Midnite Software Inc. 70, 71: CompuServe Information Service. 86: Loren Wright. 95 bottom: Loren Wright.

All remaining illustrations and screen images have been created by Time-Life Books with the assistance of the authors and consultants.

Acknowledgments

The editors are particularly indebted to the following consultants for their help in the preparation of this book: Robert P. Bartoszak, Fairfax, Va.; William H. Harr, Suitland, Md.; H. Dean Murray, Alexandria, Va.; Michael F. Spengel and Ben Waldman, Arlington, Va.

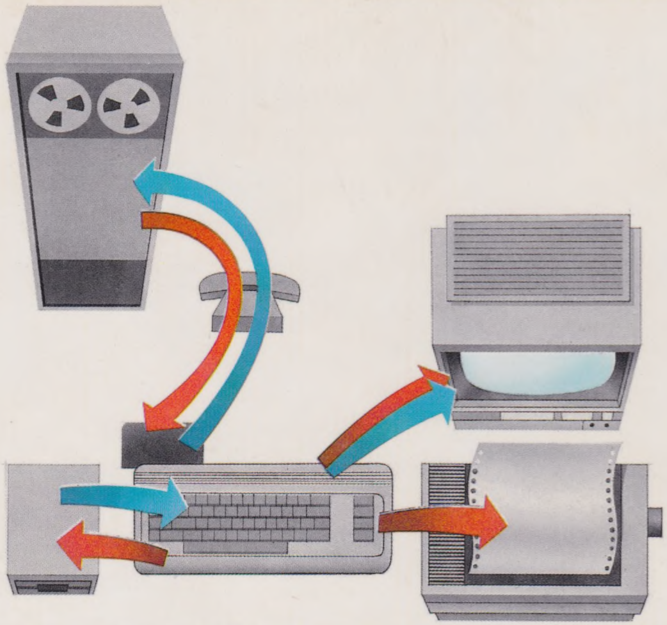
The editors also wish to thank: John Krout, Arlington, Va.; Julie Shanks, Washington, D.C.; Ron Solomon, Falls Church, Va.; Susan West, Public Relations, Commodore Business Machines, West Chester, Pa.; Stu Mitchell, Shelley Barker, Virginia Micro Systems, Woodbridge, Va.

Index with Glossary

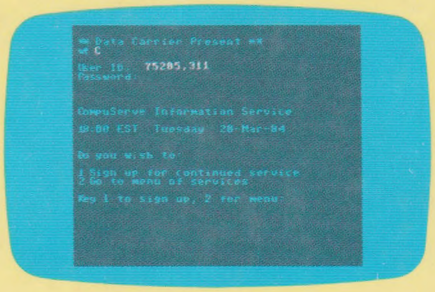
Numerals in italics indicate an illustration of the subject mentioned.

Only TIME LIFE shows it so clearly

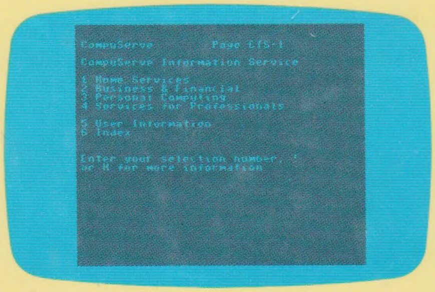
Making the link with Software



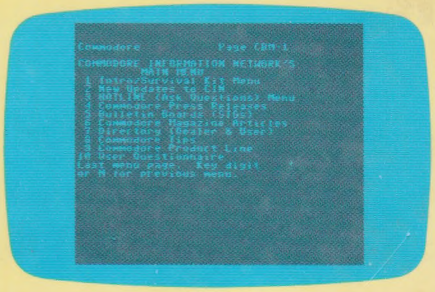
The red arrows show how communication begins between your 64 and a distant computer. First, you load a communications program into the 64 through your disk drive or Datasette. The program asks you to enter communication settings at the keyboard. The computer then uses these settings to establish a link: signals translated by the modem sent out via phone lines to the remote computer. The blue arrows indicate the reverse flow of information, which passes through the phone lines and modem into your computer; it is displayed on your monitor and can be saved to disk or printer.



```
** Data Carrier Present **
* C
User ID:  75205,311
Password:


CompuServe Information Service

10:00 EST  Tuesday  20-Mar-84

Do you wish to:

1 Sign up for continued service
2 Go to menu of services

Key 1 to sign up, 2 for menu:
```



```
CompuServe         Page CIS-1

CompuServe Information Service

1 Home Services
2 Business & Financial
3 Personal Computing
4 Services for Professionals

5 User Information
6 Index


Enter your selection number,
or H for more information
```



```
Commodore              Page CBM-1

COMMODORE INFORMATION NETWORK'S
         MAIN MENU
 1 Intro/Survival Kit Menu
 2 New Updates to CIN
 3 HOTLINE (Ask Questions) Menu
 4 Commodore Press Releases
 5 Bulletin Boards (SIGs)
 6 Commodore Magazine Articles
 7 Directory (Dealer & User)
 8 Commodore Tips
 9 Commodore Product Line
10 User Questionnaire
Last menu page.  Key digit
or M for previous menu.
```

The screen above shows a typical log-on procedure for Compuserve. First dial the CompuServe number provided for your area. Once connected, hold down the **CONTROL** key and press **C**. You will then be asked for your ID number and password. To ensure that your password remains private, CompuServe keeps the word from showing on the monitor screen when you type it.

Once you are connected to CompuServe, a main menu like that above lists a wide array of services. Each menu entry leads to another, specialized menu; entering **2**, for example, brings up a list of financial and business services. To go directly to Commodore's Information Network, type **go cbm**.

The menu for the Commodore network shows the variety of Commodore-related information available. The hotline (option 3) offers help on knotty computer problems; option 9 leads you to announcement about new products. And you can gain access to the Commodore's user bulletin board by choosing option 5.

$12.95 Cover design: Robert Aulicino Cover photograph: Larry Sherer 394-72515-8