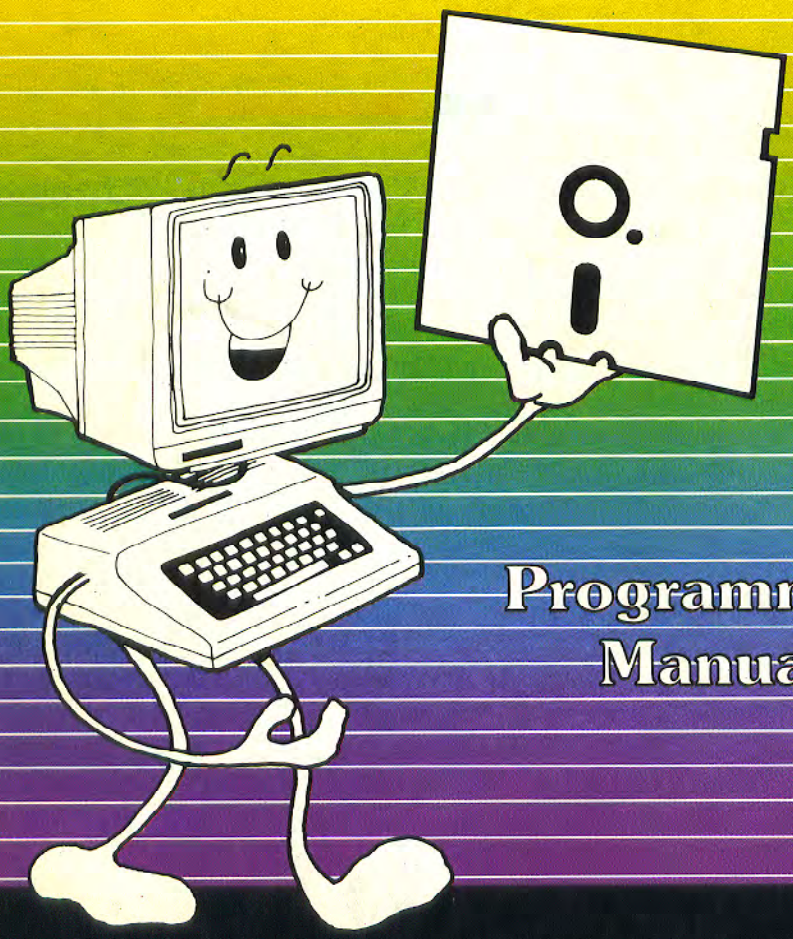
TANDY®

# Color Computer Disk System



## Programming Manual

For your own protection, we urge you to record the serial number of this unit in the space provided. You will find the serial number on the back panel of the unit.

Serial Number _____ _____

TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

# USA LIMITED WARRANTY

## I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

## II. LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. **This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations.** The warranty is void if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

## III. LIMITATION OF LIABILITY

A. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE."**
**NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.**

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

## IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on a multiuser or network system only if either, the Software is expressly labeled to be for use on a multiuser or network system, or one copy of this software is purchased for each node or terminal on which Software is to be used simultaneously.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

## V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack.

## VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

4/87

## SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

6 86

## The FCC Wants You to Know

This equipment generates and uses radio frequency energy. If not installed and used properly, that is in strict accordance with the manufacturer's instructions, it may cause interference to radio and television reception.

It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

## Warning

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/ouput devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

2/86

10987654321

# COLOR COMPUTER
# DISK SYSTEM

# A DISK IS FASTER

A disk is one means of storing information with your computer. It is far superior to cassette tape, the other alternative.

> **Note:** In particular, your Color Computer uses 5-1/4-inch *floppy diskettes*. However, we refer to these simply as *disks* throughout this manual.

A disk is especially designed to file your information so that the computer can immediately get the information you want. For you, this means that storing and retrieving information, which takes a long time on tape, now can be done quickly and efficiently.

# ABOUT THIS BOOK

This book describes how to read and write on a disk, using any of the Color Computers: the original Color Computer, the Color Computer 2, or the Color Computer 3. When we wrote it, we had three different groups of people in mind.

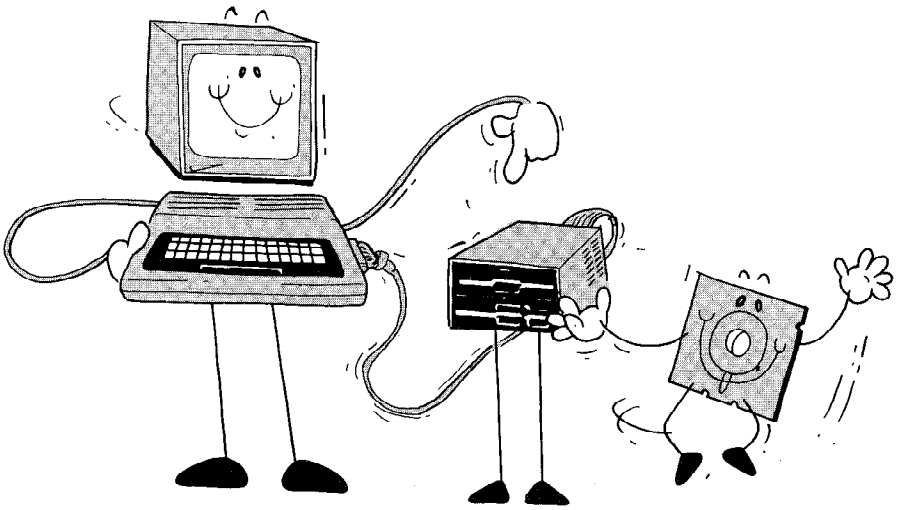
The first group includes all you accomplished Radio Shack programmers. We are referring, of course, to those of you who learned to program by reading the BASIC manual that came with your computer. You'll find Parts 1 and 2 of this book another enjoyable experience. If you're especially ambitious, you'll also enjoy Part 3.

How about those of you who have never programmed and intend to use application programs written by Radio Shack or someone else? You're the second group. Read Chapter 1, "To Get Started." Then, if you're interested in and want to take full advantage of your disk system, go on to Part 1, "The Disk." You don't need to know anything about programming to understand that part.

If you don't belong to either of these two groups, you probably already know how to program disk systems. Read Chapter 1 first to find out how to connect your system. Then, go straight to Appendix H, "Disk BASIC Summary." Everything is summarized there with chapter references, for the things you want to read more about.

# Contents

# 4 Appendices



# 1

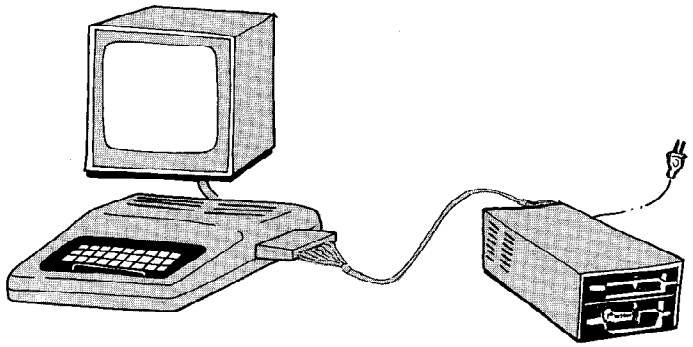
# To Get Started

Before you install your disk system, you need to connect your Color Computer to the TV. If you haven't done that yet, refer to the introduction manual for your computer.

## A. Connect Your Disk System

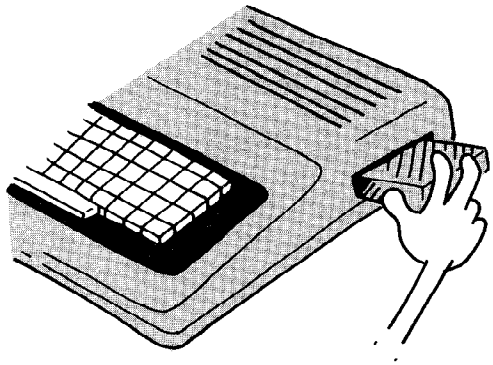
Your disk system is easy to connect. Do it **before** you turn on your computer, by plugging in three parts:

- The disk interface cartridge
- The disk drive cable
- The disk drive power cord

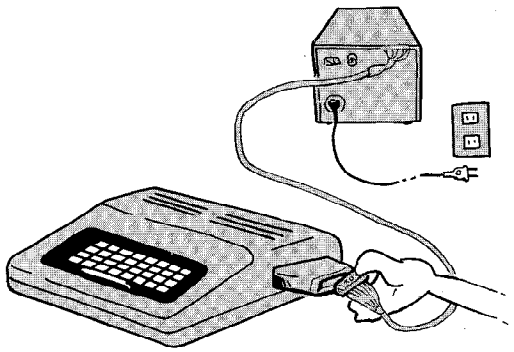


1. Insert the disk interface into the cartridge port on the right side of your computer. Use firm pressure, and be sure the interface cartridge is fully in place.

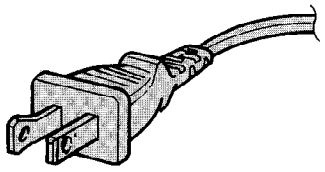
> **Caution:** Your computer must be off when you connect the disk interface. Otherwise, you could damage the system.



2. Connect the disk cable to the disk interface. Be sure that the TOP side of connector is facing up.



3. Plug the power cord, located at the rear of your disk drive housing, into a standard household electrical outlet.



**Note:** The power cord has a polarized plug—one blade is larger than the other. If you do not have polarized outlets, have such an outlet installed before using the unit. DO NOT ATTEMPT TO DEFEAT THIS SAFETY FEATURE.

Your disk drive system can consist of one or two drives. The lower drive is referred to as Drive 0, and the upper drive is referred to as Drive 1. If you have only one drive in your system, your Radio Shack Computer Center service technician can install a second drive.

# B. Power It Up

Because your disk system has several parts, you need to turn on several buttons to power up the entire system:

1. Turn on your television set.
2. Select Channel 3 or 4 on your computer's channel select switch.
3. Set the antenna switch on the TV to COMPUTER.
4. Turn on the computer. The power button is on the back of the computer, near the left side.
5. Turn on the disk drives. The power button is on the back of the drive.

If all the buttons are properly set, the following message appears on your screen. (The message varies depending on the computer you have.)

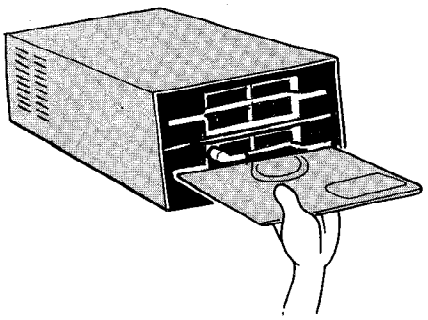
```
DISK EXTENDED COLOR BASIC v.r.
COPR. 1982,  1986 BY TANDY
UNDER LICENSE FROM MICROSOFT
AND MICROWARE SYSTEMS CORP.
```

(*v.r.* consists of two numbers that specify which version and release of Disk Extended Color BASIC you have.)

If the message does not appear, turn off the computer, and check the connections. Then, power up the computer again.

# C. Insert a Disk

After powering up the system, you can insert a disk. If you plan to go through Part 1, use the blank, unformatted disk that comes with your disk system. Otherwise, you can insert your *application program* disk. Insert the disk in Drive 0 as illustrated.



1. Open the drive door.
2. Position the disk with the label up, as illustrated.
3. Gently insert the disk until it stops.
4. Turn the drive latch clockwise until it stops in a vertical position.

> **Note:** You cannot use a blank disk until you *format* it. The next chapter shows how.

When you are ready to remove the disk, turn the drive latch counterclockwise until it stops in a horizontal position. Then, remove the disk by pulling it toward you.

Now that your system is connected and powered up, you're ready to begin. Begin what? Well, if you want to know how to take full advantage of your disk system, we'd like you to read Part 1. You'll find a lot of helpful information there.

If you're in a hurry to run your application program, that's O.K., too. But please read these guidelines first. We want your disks to last a long time.

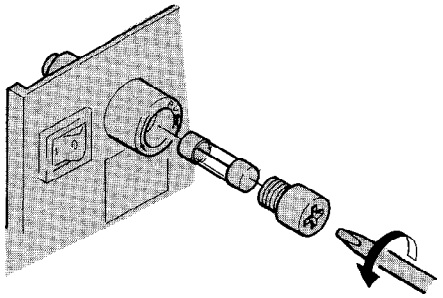
- When storing your disks, keep them in their protective envelopes. Store them upright in a file.
- Never turn the system on or off with a disk in a drive.
- Keep disks away from magnetic fields (transformers, AC motors, magnets, TVs, radios, and so on).

- Handle disks by the jacket only. Don't touch any exposed surfaces, even to dust them.
- Keep disks out of direct sunlight and away from heat.
- Avoid contamination of disks with cigarette ashes, dust, or other particles.
- Use a felt-tipped pen only to write on the disk label.

**Note:** Your disk drives should be no closer than six inches from your television set.

# D. Replacing the Power Fuse

The power fuse, located on the back panel, protects the unit from voltage surges or other abnormal operating conditions. If the disk drive does not operate and you do not hear the fan motor rotating, check the fuse. If it is blown, replace it with another fuse of the same size and amperage. Use a screwdriver to loose the fuse holder.



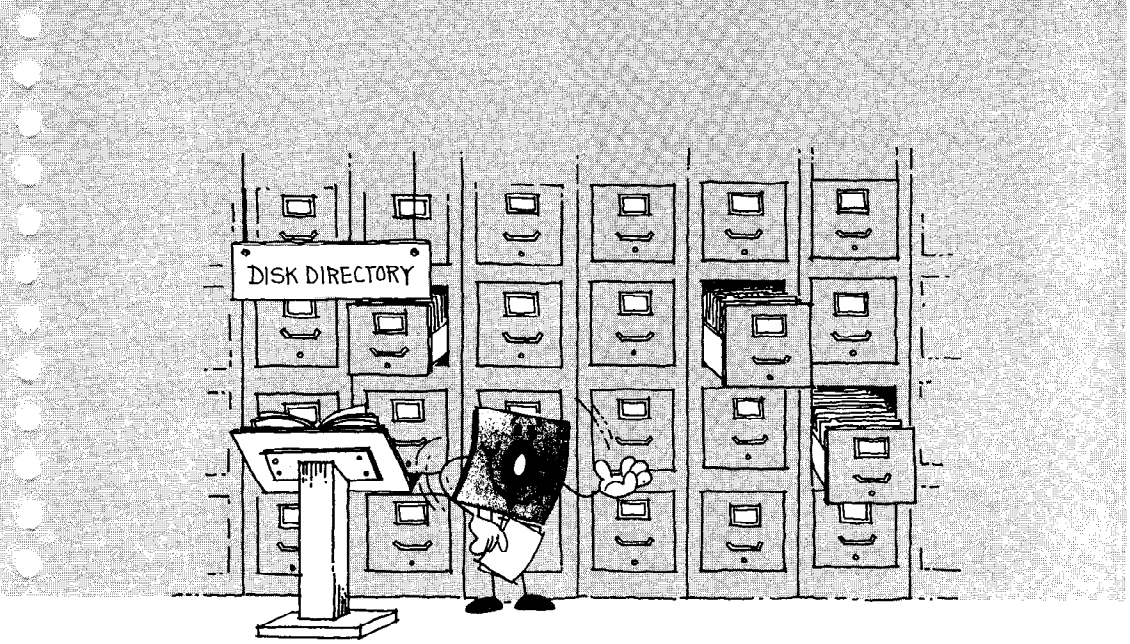
# Part 1

# THE DISK

A disk is like a filing system. Everything on it is organized.

This makes disks easy to work with. In this part, we describe how your computer organizes everything on your disk, and tell how you can take advantage of this organization.

We invite all of you to read this section. You don't need to know anything about computers to understand it.




# 2

# Meet Your Disk

## A Look Inside the Disk

Although your disk looks like a phonograph record, it is really more like a multitude of tiny magnets. One disk can hold more than a million magnetic charges. 1,290,240 of them are for your information. That's what we mean when we say a disk can hold 1,290,240 *bits* (or 161,280 *bytes*) of information. (There are eight bits in a byte.)

Some of these bits are magnetically charged, and some aren't. The pattern formed by these magnetic charges is important. It forms a code that the computer can read.

With more than a million of these bits on a disk, you can appreciate how your computer must organize them in order to find anything. It does this by building a massive disk filing system. First, it creates the file "cabinets" by dividing your disk into *tracks*. Then it puts "drawers" in the cabinets by dividing each track

into *sectors*. Then—the computer isn't finished yet—it divides each sector into bytes and each byte into bits.

> **Note:** To be precise, there are 35 tracks on a disk, 18 sectors in each track, 256 bytes in each sector, and eight bits in each byte.

After creating the filing system, the computer puts a master directory on the disk. In this directory, it keeps an index of the location of all the information on the disk. Whenever it wants to find something—a program, a mailing list, your letters—the computer uses the directory to find the tracks and sectors on which the information is stored. It can then go directly to that spot.

This filing system is, of course, what makes the disk system so powerful. You can quickly find anything stored on your disk.

Putting the filing system on your disk is called *formatting* the disk. The last thing you did in Chapter 1 was insert an *unformatted* disk. Before you use the disk, you must format it into tracks and sectors.

# Formatting a Disk

How do you format a disk? Well...why not just tell your computer to do it?

You have already powered up your system and inserted an unformatted disk. Be sure the drive latch is down.

Now, type any letters and press the (ENTER) key so that:

```
OK
```

is the last line on your screen. (**OK** means "OK, I'm ready to do something.") Now, tell the computer what you want it to do. Type:

```
DSKINIØ
```

and press the (ENTER) key.

Your computer might display **?SN ERROR**. If so, don't let this bother you. This error message means you typed the command incorrectly. Type it again.

Whenever anything goes wrong, the computer lets you know immediately with an error message. This way, you can correct the error right away. If you get any error message other than **SN**, look up the message in Appendix G. Appendix G lists all the error messages and what to do about them.

After typing **DSKINIØ** (ENTER) , you hear some noises from your disk drive, and the disk's red light comes on. Sounds promising...

After about 40 seconds, your computer prints **OK** on the screen. It is finished formatting the disk. You can now store your information.

Remember that you cannot store anything on an unformatted disk. Whenever you get a new, unformatted disk, you need to format it before you use it.

Later on, you might forget whether a disk is formatted. A quick way to find out is to check the directory. (See "Checking the Master Directory," at the end of this chapter.) If you get an error message, the disk is not formatted.

> **Note:** You can also reformat a disk. In fact, reformatting is a common way to erase everything on a disk.

If you have two disk drives, you can format a disk in Drive 1 by substituting 1 for 0 in your instruction to the computer. For example, type **DSKINI1** (ENTER) to format a disk in Drive 1.

# Putting a File on Your Disk

A disk file can contain any kind of information: a program, a mailing list, an essay, some checks, and so on. We'll make your first file contain a BASIC program, since it's the simplest thing to store.

If you don't know how to program in BASIC, type this program anyway. Type each line exactly as shown below. Press the (ENTER) key after typing each line.

```
10 PRINT "STORE ME IN A DISK FILE" (ENTER)
20 PRINT "AND YOU'LL NEVER LOSE ME" (ENTER)
```

Finished? Now that the program is in your computer's *memory*, you can put it on a disk. To do this, consider the program to be a file, and name the file SIMPLE/PRO. (All files have a name.) To store the file, type:

```
SAVE "SIMPLE/PRO" (ENTER)
```

Once you press the (ENTER) key, your disk drive's red light comes on, and the drive whirs some. Your computer is:

1. Finding a place on the disk to store SIMPLE/PRO.
2. Telling the directory where SIMPLE/PRO is stored.
3. Storing SIMPLE/PRO on your disk.

**Note:** The computer stores SIMPLE/PRO the same way it stores everything — else as a code of magnetic charges.

At this point, we must warn you about something. Do not remove your disk while you see the red light on. This confuses the computer. It might distort the contents, not only of the file you are presently storing, but of other things stored on the disk.

When your computer finishes storing SIMPLE/PRO, it displays the **OK** message on your screen.

**Note:** Upgrading your tape system? Note the difference. SAVE stores a program on disk; CSAVE stores it on tape.

# Memory vs. Disk Storage

To those of you new to computers, we would like to expound a little on computer memory. If you already know what memory is, skip to the next section, "Loading a File from Disk."

Whenever you type a BASIC program line and press (ENTER) , the computer automatically puts the line in its memory. Once the line is in memory, you can do things with it. For example, type:

```
RUN (ENTER)
```

Your computer displays:

```
STORE ME IN A DISK FILE
AND YOU'LL NEVER LOSE ME
```

To list the program as you have it above, type:

```
LIST (ENTER)
```

Memory is where the computer keeps track of everything you tell it. Once you put your information in its memory, the computer can print the information, rearrange it, combine it, or do many other things with the information.

Later, you can put other things, such as a mailing list, in memory. To do this, you need to write or purchase a program written especially for that application. This *application program* causes the computer to put the information you type into memory.

The most important thing to remember about memory is that turning off your computer erases it. Once memory is erased, there's no way to recover it. The only way to keep a permanent copy of what you type is to store it on a disk (or tape).

# Loading a File from Disk

Type **NEW** (ENTER) to erase everything in your computer's memory. To be sure that everything is erased, you can type one or both of these commands:

```
RUN (ENTER)
LIST (ENTER)
```

Although NEW erased the program from memory, the file SIMPLE/PRO is still safely stored on your disk. You can put SIMPLE/PRO back into memory whenever you wish by *loading* it from disk. To do this, type:

```
LOAD "SIMPLE/PRO" (ENTER)
```

Again, you hear the disk drive working. The computer is:

1. Reading the directory to find where SIMPLE/PRO is stored.
2. Going to that location on the disk, and reading the contents of SIMPLE/PRO.
3. Putting SIMPLE/PRO into memory.

You can now type one or both of these commands to verify that SIMPLE/PRO is in memory:

```
LIST (ENTER)
RUN (ENTER)
```

## Starting OS-9

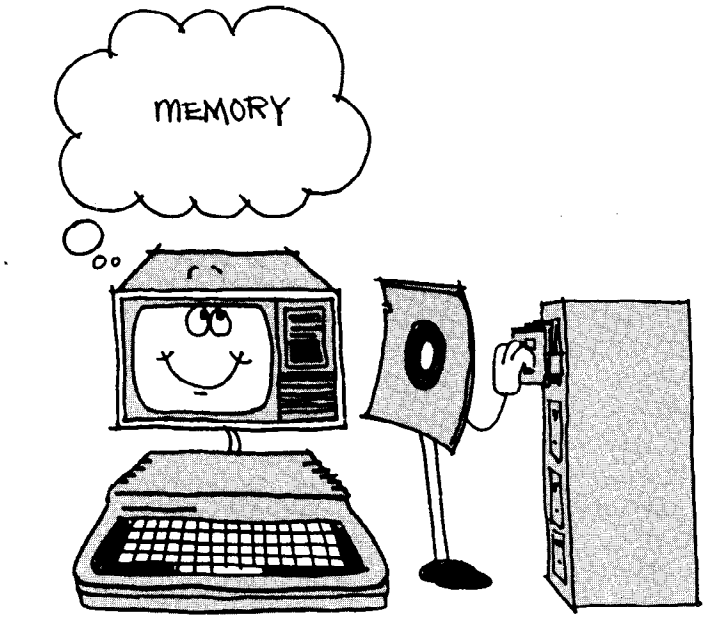
Although the Color Computer disk system is complete with its own built-in disk operating system, it is also capable of operating under other systems.

OS-9 is a flexible and sophisticated system that you can use with the your computer. With OS-9 Level Two, your Color Computer 3 has the capability of using 40 tracks and both sides of the disk. This capability allows over twice as much storage space on the same diskette. You execute OS-9 with a special command, DOS. If you have the OS-9 system diskette, load and execute the system by placing the diskette in Drive 0 and typing:

```
DOS (ENTER)
```

After a moment, the words **OS-9 BOOT** appear in the center of the screen. Soon the OS-9 copyright screen appears, and you are in the OS-9 operating system. For further information on starting the system, refer to your OS-9 manuals.

Some OS-9 application programs can be loaded and executed in the same manner as the OS-9 system disk.




# More about Memory vs. Disk Storage

If you're still a little fuzzy about what's in memory and what's on your disk, try this exercise. You just loaded a program called SIMPLE/PRO into memory, right? Change it by typing:

```
20 PRINT "WITH THIS CHANGE" (ENTER)
```

List the program again to see that the computer has registered the changed Line 20 in its memory:

```
10 PRINT "STORE ME IN A DISK FILE"
20 PRINT "WITH THIS CHANGE"
```

Store the program in a different file by typing:

```
SAVE "CHANGE" (ENTER)
```

You have two disk files now: SIMPLE/PRO and CHANGE. What do you think each of them contains? Try loading and then listing both.

> **Note:** You don't need to type **NEW** (ENTER) before **loading** a program into memory from disk—only before **typing in** a program. When you load a program from disk, the computer automatically erases everything presently in memory.

The CHANGE file contains the changed program:

```
10 PRINT "STORE ME IN A DISK FILE"
20 PRINT "WITH THIS CHANGE"
```

However, SIMPLE/PRO still contains the old program:

```
10 PRINT "STORE ME IN A DISK FILE"
20 PRINT "AND YOU'LL NEVER LOSE ME"
```

The only way to change a disk file is by... Well, you answer it. How can you make the file SIMPLE/PRO contain:

```
10 PRINT "CHANGED FILE"
```

*Answer:*

Type:

```
NEW (ENTER)
10 PRINT "CHANGED FILE" (ENTER)
SAVE "SIMPLE/PRO" (ENTER)
```

# Filenames

You have already used one filename: SIMPLE/PRO. If you did our memory vs. disk storage exercise, you also used a second filename: CHANGE.

We gave the filename SIMPLE an *extension*, PRO. You must give everything you store a name. The extension is up to you. It's optional.

Do you wonder what names you can give your files? Anything you want, as long as you follow these rules:

- The name can have no more than eight characters. (No blank spaces are allowed.)
- If you give the name an extension, the extension can have no more than three characters.
- There must be a slash (/) or a period (.) between the name and the extension.

Fair enough? Good.

**Note:** You can use any characters in the filename except a colon (:). You can use a slash (/) or a period (.) only to separate the name from the extension.

# Filenames When You Have Two Drives

If you have two disk drives, you can add the drive number to your filename. Remember, the drive numbers are 0 (the lower drive) and 1 (the upper drive).

```
LOAD "SIMPLE/PRO:1" ENTER
```

loads SIMPLE/PRO from the disk in Drive 1. Or

```
SAVE "CHANGE:1" ENTER
```

stores CHANGE on the disk in Drive 1. If you don't include a drive number, the computer assumes you want it to use Drive 0.

# Checking the Master Directory

As stated earlier, a disk has a master directory that the computer can use to find out what's on the disk. If the computer can use the directory, you can use it, too. Type **DIR** ENTER

The computer displays information about all the files stored on your disk. If the only files you stored so far are SIMPLE/PRO and CHANGE, the computer displays this:

```
SIMPLE    PRO  Ø  B  1
CHANGE    BAS  Ø  B  1
```

The first and second columns list the filename. The first contains the name and the second contains the extension. Notice that even though you did not assign CHANGE an extension when you stored it, the computer assigned it the extension BAS.

The computer prefers that all filenames have an extension. If you do not give a file an extension when you store it, the computer automatically assigns the file one of these extensions:

BAS if the file is a BASIC program
DAT if the file contains data (such as names and numbers)
BIN if the file is a machine-language program

> **Note:** *A machine-language* program is a highly technical program that talks directly to the computer.

The last three columns contain information that is primarily for the use of technical programmers. Interested? Then, read on.

The third column lists the type of file:

0 = a BASIC program
1 = data created by a BASIC program
2 = data created by a machine-language program
3 = a source program created by an editor/assembler

> **Note:** An *editor/assembler* is a program you can buy to help you create a machine-language program.

The fourth column lists the format the file is stored in:

A = ASCII format
B = binary format

We explain the meaning of these formats in Chapter 10.

The fifth column shows how many *granules* each file consumes. SIMPLE/PRO and CHANGE/BAS consume one granule each. (The computer uses granules to allocate file space on a disk. A disk contains 68 granules with 2,304 bytes per granule.)

If you have a disk inserted and formatted in Drive 1, you can check **its** directory also. For instance, typing **DIR1** (ENTER) displays the directory of the disk in Drive 1.

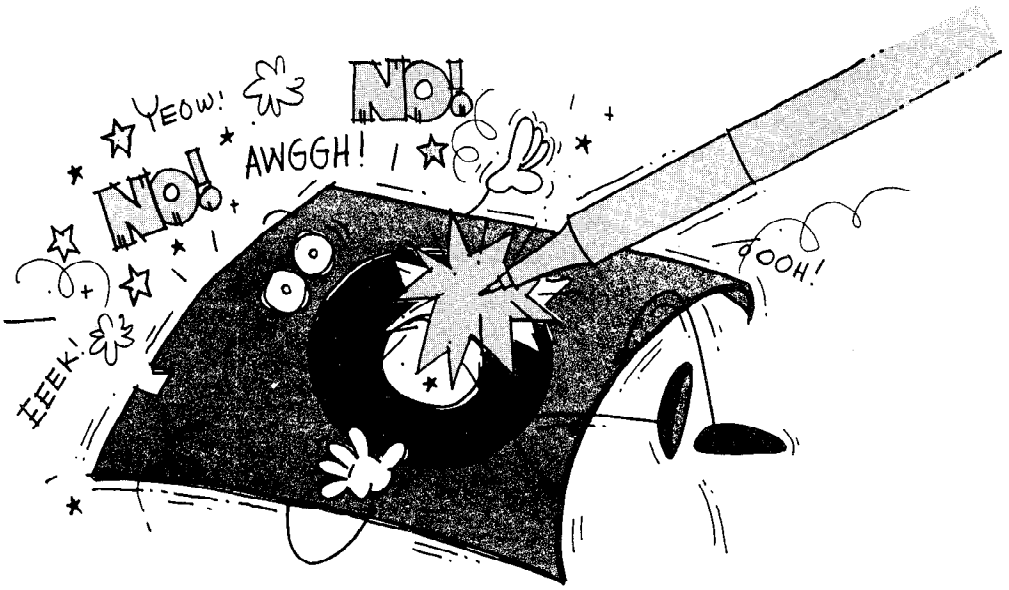
Impressed? Wait until you see how fast you can save and load long programs. But before you get too involved, please read the next chapter. It helps ensure that your experience with your disk system is smooth and enjoyable.

**Note:** To stop the directory from *scrolling* up the screen, press the (SHIFT) and @ keys simultaneously. Then press (BREAK).

### ☐ CHAPTER CHECKPOINT

1. *Why can't you store things on an unformatted disk?*
2. *What is the disk's directory?*
3. *What is a disk file?*
4. *What is the difference between what's in memory and what's on the disk?*
5. *How do you change the contents of a disk file?*

*Do you like quizzes? The answers are in Appendix B.*




3

# A Garbled Disk

With more than a million magnetic charges on a disk, you can see why it is so delicate. Any small particle such as a piece of dust or a cigarette ash could distort its contents. A scratch could ruin it. That's why we suggest that you keep the disk in its envelope when you're not using it, preferably upright in a dust-free container, and that you use only a felt-tipped pen when labeling it.

To help protect the disk, we encase most of it in a black plastic cover. However, as you can see, we aren't able to cover the entire disk. The middle section and two other small areas are exposed so that the computer can read and write to the disk. Be careful not to touch the exposed areas, not even to dust them. They scratch easily.

Since the disk is made of magnetic charges, putting it next to another magnetic device, such as your television set, could rearrange its magnetic code.

Your information would be lost. Heat and sunlight could have the same effect. The same goes for turning your computer on or off while the disk is in its drive.

One more thing: if you're in the middle of running a disk program, and need to switch disks, we recommend that you type the following command before switching disks:

**UNLOAD** (ENTER)

This command tells the computer to put the closing information on the disk in Drive 0. If you don't type this command, the computer might put this information on the wrong disk, and garble the contents of both disks.

**Note for BASIC programmers:** All open files must be closed before you can switch disks. UNLOAD closes all open files.

# Back It Up

To help protect the information on your disks, we've included a command called BACKUP. BACKUP enables you to make a duplicate, or *backup*, of any of your disks by copying the contents of one disk to another.

We suggest you regularly make a backup copy of any disk that contains important programs or data. By doing this, you don't have to worry about losing the information.

Also, since a disk can actually get worn out from too much use, it's a good idea to make a backup copy of an old disk on a new, unused disk. Then, if the computer begins having problems reading and writing to the disk, you can use your backup copy.

Want to make a backup copy? Get your two disks ready:

- Your *source* disk is the disk you want to duplicate. Use any disk that has files stored on it. If you're just getting started, use the disk that you worked with in Chapter 2.

- Your *destination* disk is the disk that you want to be your duplicate. Use a blank disk or, if you've been using your disk system for a while, use any disk that contains files you don't need anymore.

> **Note:** Everything previously on your destination disk is erased. The computer replaces the information with all the data on your source disk.

If your destination disk is blank, you must first format it. Remember how? Insert it in your disk drive, turn the latch, and type **DSKINIØ** (ENTER) .

Now, make the backup. The procedure you follow depends on whether you have one disk drive or two.

## Backup With One Disk Drive

If you have only one disk drive, it takes about five minutes to make a backup copy.

Insert your source disk in your disk drive, and turn the drive latch down. Type **DIR** (ENTER) to see which files you will be copying.

Now, start the backup procedure. Type:

```
BACKUP Ø (ENTER)
```

After reading a portion of your source disk, the computer displays:

```
INSERT DESTINATION DISKETTE AND PRESS
ENTER
```

Remove the source disk, and insert the destination disk. Turn the drive latch. Then press (ENTER) . The computer writes on the destination disk. Then, the screen displays:

```
INSERT SOURCE DISKETTE AND PRESS ENTER
```

The computer requests that you continue switching disks until it finishes copying everything from your source disk. During this process, be sure you insert the correct disk and insert it properly. When finished, the computer displays the **OK** message on your screen.

To be sure BACKUP worked, insert your destination disk and type **DIR** (ENTER)

## Backup With Two Disk Drives

If you have two disk drives, you can back up a disk in about two minutes.

Insert your source disk in Drive 0 (the lower drive) and your destination disk in Drive 1 (the upper drive). Then type:

**BACKUP Ø TO 1** (ENTER)

The computer backs up the contents of the disk in Drive 0 to the disk in Drive 1. After finishing, it displays the **OK** message. You can check to be sure BACKUP worked. Type **DIR1** (ENTER) .

You can use the two drives in reverse, if you want. For instance:

**BACKUP 1 TO Ø** (ENTER)

backs up the contents of the disk in Drive 1 to the one in Drive 0.

## If You have Problems During Backup

If you get an error message while backing up a disk, it's probably because you inserted the disk incorrectly or because there is something wrong with the disk. At the end of this chapter, we discuss error messages to help you determine the problem. If you have a bad disk, try BACKUP with another disk.

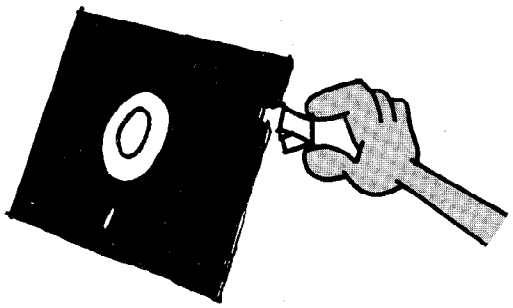
After determining the problem, press the reset button to get out of BACKUP. Then, start the BACKUP procedure again.

> **Note:** The reset button is on the back of the computer near the right side (as you face the computer).

# "Write Protect" It

*Write protecting* is one more way to protect your disk files. Let's assume you have a disk that contains some valuable information — such as a good program you don't plan to change. You plan to read its contents daily, by loading the program into memory. Yet, you never plan to write (store information) on it.

By folding a little gummed label over the disk's *write-protect notch*, you enable the computer to read the disk but not write on it. Any gummed label works. One comes with your new, unformatted disk.



# Salvage It

We mentioned earlier that a disk doesn't last forever. Before you throw away an old disk, though, see if you can salvage it. You might be able to do this by formatting the disk again as if it were a blank disk.

Although reformatting might salvage the disk, it does not salvage the **contents** of the disk. Reformatting the disk erases everything on the disk. However, it **also** saves you the expense of purchasing a new disk.

If you get an IO error while trying to reformat a disk, the disk has probably reached its limit (See "When Things Go Wrong," at the end of this chapter.) If you have a *bulk-eraser*, try *bulk-erasing* the disk and reformatting it. Otherwise, throw away the disk and use another.

> **Note:** If you have two disk drives, you might be able to copy some of the files on a bad disk to a good disk. We discuss COPY in the next chapter.

# Verify It

The computer writes data on your disk at a very fast speed. In almost all cases, it does this flawlessly.

There might be times when you want to be certain that there are no flaws in what the computer is writing. If so, you can turn on the computer's VERIFY command. To do this, type:

**VERIFY ON** (ENTER)

Now, the computer notifies you, whenever it is writing on a disk, if there are any flaws in what it is writing. The only catch is that it takes twice as long for the computer to write.

For example, if you use the VERIFY command when you make a backup of your disk, the computer takes twice as long, but notifies you if there is a flaw in the backup.

This VERIFY command remains on until you turn it off. To turn it off, type:

**VERIFY OFF** (ENTER)

# When Things Go Wrong

Whenever you make a mistake, your computer tries to notify you immediately and tell you what kind of error you made.

You have probably already made an SN ERROR. If you haven't, type **DIIR** (ENTER) , deliberately misspelling DIR.

SN means *Syntax* error. It's the computer's way of telling you that **DIIR** doesn't make sense. The word is not in its vocabulary. An SN error usually means you made a typographical error.

Here are some other error messages you're likely to get with your disk system:

AE — You are trying to rename a file to a filename that already exists. (RENAME is discussed in the next chapter.)

DF — The disk you are trying to store your file on is full. Use another disk.

DN — You are using a drive number higher than 1. You also get this error if you do not specify a drive number when using DSKINI or BACKUP. If you have only one drive, specify Drive 0 with these two commands (**DSKINIØ** or **BACKUP Ø**).

FN — You used an unacceptable format to name your file. Chapter 2 explains which filenames the computer accepts.

FS — There is something wrong with your disk file. See the IO error description for instructions on what to do.

/0 — Technically, this means you asked the computer to divide a number by 0, which is impossible. However, you might also get this error if you don't enclose a filename in quotation marks.

IO — The computer is having trouble inputting information from or outputting information to the disk.

1. Be sure that there is a disk inserted properly in the indicated drive and that the drive door is closed.

2. If you still get this error, there might be something wrong with your disk. Try reinserting the disk first. Then, try reformatting it or using a different disk. (Remember that reformatting a disk erases its contents.)
3. If you still get this error, you probably have a problem with the computer system itself. Call your nearest Radio Shack Computer Center.

**NE** — The computer can't find the disk file you want. Check the disk's directory to see if the file is there. If you have two disk drives, you might not have included the appropriate drive number in the filename. If you are using COPY, KILL, or RENAME (discussed in the next chapter), you might have left off the extension.

**TM** — Technically, this error is caused by a program that mixes *strings* with *numbers*. However, you might get this error if you don't enclose a filename in quotation marks.

**VF** — You get this error only when you have the VERIFY command on and are writing to a disk. The computer is informing you that there is a flaw in what it wrote. See the IO error description for instructions on what to do.

**WP** — You are trying to store information on a disk that is write protected. Either remove the label from the write-protect notch, or use a different disk. If your disk is not write protected, then there is an input/output problem. See the IO error description for instructions on what to do about this.

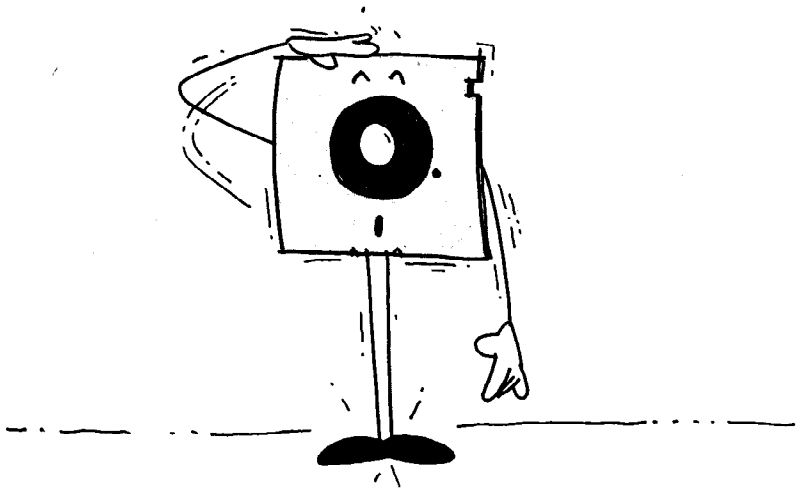
All other errors you might get are errors in the program you are using. If you did not write the program, and you get one of these errors, you need to contact the people who wrote it. If you did write it, check Appendix G, where you find an explanation of all the error messages.

Caring for your disk might seem a little awkward at first. It should. You've spent a lot of time using paper to store information, and now you're dealing with a different medium.

After a while, though, protecting your disk from dust and magnetic devices will seem as natural to you as protecting your papers from a strong gust of wind. And once you get used to caring for your disks, you aren't likely to go back to pencils and paper again.

### ☐ CHAPTER CHECKPOINT

1. *Why shouldn't you turn the computer on or off while the disk is in its drive?*
2. *What type of pen can you use to write on the disk's label?*
3. *What are error messages?*
4. *What does write protect mean? How do you do it?*
5. *How do you back up a disk?*



4

# You're the Boss

Thanks to your disk filing system, you are able to command the computer to do a lot of very helpful things. For example, you can copy a file to the same disk or to another disk. So that you have a disk on which to try out the COPY command, use DSKINI to format another disk as instructed in Chapter 2.

> **Note:** If you can't remember whether your disk is formatted, check the directory by typing **DIR** (ENTER) if you have one drive, or **DIR1** (ENTER) if you have two drives.

Now, replace the original diskette in Drive 0, and type:

```
10 PRINT "THIS IS A FILE" (ENTER)
SAVE "ORIGINAL/NAM" (ENTER)
```

You have just created and saved a file named ORIGINAL/NAM. To see this, use **DIR** to examine the disk's directory.

# Making One-Drive Copies

You can copy ORIGINAL/NAM to either the same disk or a new disk, using only one drive. To make a copy on the same disk, type:

```
COPY "ORIGINAL/NAM" TO "NEW/NAM" (ENTER)
```

Your disk drive turns on and runs while the computer copies ORIGINAL/NAM to a new file named NEW/NAM. Again, use the DIR command to examine the directory to see that it now includes both files.

You can also copy files from one disk to another in the same drive. You must have a second formatted disk before you can do this. To copy ORIGINAL/NAM from one disk to another, using the same name for both files, type:

```
COPY "ORIGINAL/NAM" (ENTER)
```

After the disk drive runs for a few moments, the screen displays:

```
INSERT DESTINATION DISKETTE AND
PRESS .ENTER.
```

Remove your original disk, and replace it with the disk on which you wish to copy the file. Then, press (ENTER). Use **DIR** to confirm that the computer has placed the file on the new diskette.

If a file is large, you might be requested to reinsert the original disk and press (ENTER) one or more times. After each time, you replace it with the destination disk, as instructed by the screen display. When finished, reinsert the original disk in your drive.

# Making Two-Drive Copies

Copying files between two disks is much easier if you have two drives. Be sure that the disk containing ORIGINAL/NAM is in Drive 0 and that your newly formatted disk in Drive 1. Type:

```
COPY "ORIGINAL/NAM:Ø" TO "ANY/NAM:1" ENTER
```

After the disk drives stop and the cursor returns to the screen, you can check your copies by using DIR.

You can also copy a program to another disk, using the same filename. In addition, you can copy from Drive 1 to Drive 0. To try this, type:

```
COPY "ANY/NAM:1" TO "ANY/NAM:Ø" ENTER
```

Your disk filing system also allows you to rename disk files. To change the name of ORIGINAL/NAM to FIRST/NAM, type.

```
RENAME "ORIGINAL/NAM" TO "FIRST/NAM" ENTER
```

Check your directory again. If you like, load and list FIRST/NAM. The program file has simply been renamed. Everything else is the same.

RENAME is easy to use, but there is one thing you need to remember. Save a file without an extension; then, try to rename it. Type:

```
1Ø PRINT "FILE NUMBER TWO" ENTER
SAVE "AFILE" ENTER
RENAME "AFILE" TO "BFILE" ENTER
```

The computer displays an **NE** error. This means the computer can't find the file.

Whenever you rename a file, you must type the **complete** name of the file so that the computer can find it. This includes the name and the extension. As discussed in Chapter 2, whenever you save a file the computer ensures that it has an extension. If you don't assign one, the computer does.

You can check the directory to find out the extension of AFILE. Then, rename the file. Type:

```
RENAME "AFILE/BAS" TO "BFILE/BAS" (ENTER)
```

When you rename a program file, be sure that your new filename has an extension. In other words, don't type **RENAME "AFILE/BAS" TO "BFILE"** (ENTER) . The computer would rename the file, however BFILE doesn't have an extension. This causes a problem when you try to load BFILE, because all files you load must have an extension.

This might seem to conflict with what we said above. You can save AFILE without assigning it an extension because the computer automatically assigns it one when it saves it. RENAME works differently. The computer doesn't automatically assign an extension to a program you rename.

> **Note:** There is one way to load BFILE without having to specify an extension. This method involves indicating that there is no extension, by typing **LOAD "BFILE/"** (ENTER) . You might find this method to be awkward. That's why we suggest that you always assign an extension when renaming a file.

# Multi-Disk Drives

You can rename a file on a drive other than Drive 0 by typing the appropriate drive number. Insert a formatted disk in Drive 1 (if it isn't already inserted). Store a file on it:

```
10 PRINT "ACCOUNTING" (ENTER)
SAVE "OLDACC/DAT:1" (ENTER)
```

Then, rename the file:

```
RENAME "OLDACC/DAT:1" TO "NEWACC/DAT:1"
(ENTER)
```

**Note:** If you want your renamed file on a different drive, you can't use RENAME. Use COPY.

# Almost out of Disk Space?

Sooner or later, you'll want to know how much space you have left on your disk. Type:

```
PRINT FREE(Ø) (ENTER)
```

The computer displays the number of available granules remaining on your disk.

There are 68 granules in all. If the computer tells you that you have only one granule free, do one of the following: start using another disk, or remove some of your disk files (using the KILL command).

"Killing" a disk file does exactly what the name implies. For example, if you put CHANGE on your disk in Chapter 2, type:

```
KILL "CHANGE/BAS" (ENTER)
```

Check your directory and the amount of free space on your disk. CHANGE/BAS is no longer on your disk. The space it occupied is now free for new files.

Notice, we had to include CHANGE's extension, BAS, in order to remove the file. The computer insists you type the complete filename as one extra precaution. We don't want you to delete a file you don't want destroyed.

> **Note:** Want to get very technical? The data still exists on the disk after you kill a file. However, the computer doesn't know that the data is there because KILL deletes the reference to it in the disk's directory. Therefore, you no longer are able to access the data and the computer is able to write over it with a new file.

## Multi-Disk Drives

You can also use FREE and KILL on Drive 1, as you can RENAME, by typing the drive number. Examples:

```
PRINT FREE(1) ENTER
```

tells you how much free space is on the disk in Drive 1.

```
KILL "NEWACC/DAT:1" ENTER
```

deletes NEWACC/DAT from the disk in Drive 1.

There is also a command that changes the drive number the computer goes to if you do not specify one. Until now, this was Drive 0. For example, if you type **SAVE "ANYTHING/EX"** ENTER , the computer assumes you want to use Drive 0. It then saves the program on the disk in Drive 0.

To change this assumption, you can type:

```
DRIVE 1 ENTER
```

This makes the computer assume you want it to use Drive 1, unless you tell it otherwise.

After you change the drive assumption, or *default*, the computer responds differently to the same command. Now, if you type **SAVE "ANYTHING/EX"** (ENTER), the computer stores ANYTHING/EX on the disk in Drive 1. You now need to type **SAVE "ANYTHING/EX:Ø"** (ENTER) if you want to save the program on the disk in Drive 0.

### ☐ *CHAPTER CHECKPOINT*

1. *How do you rename a file? Why do you have to specify the file's extension?*
2. *What can you do when you think you're running out of disk space?*
3. *If you have two disk drives and do not specify the drive number, which drive does the computer use? How can you change this?*

Congratulations. You are now a bonafide disk system operator. You should now have a good understanding of how your disk system works and how to take full advantage of it.

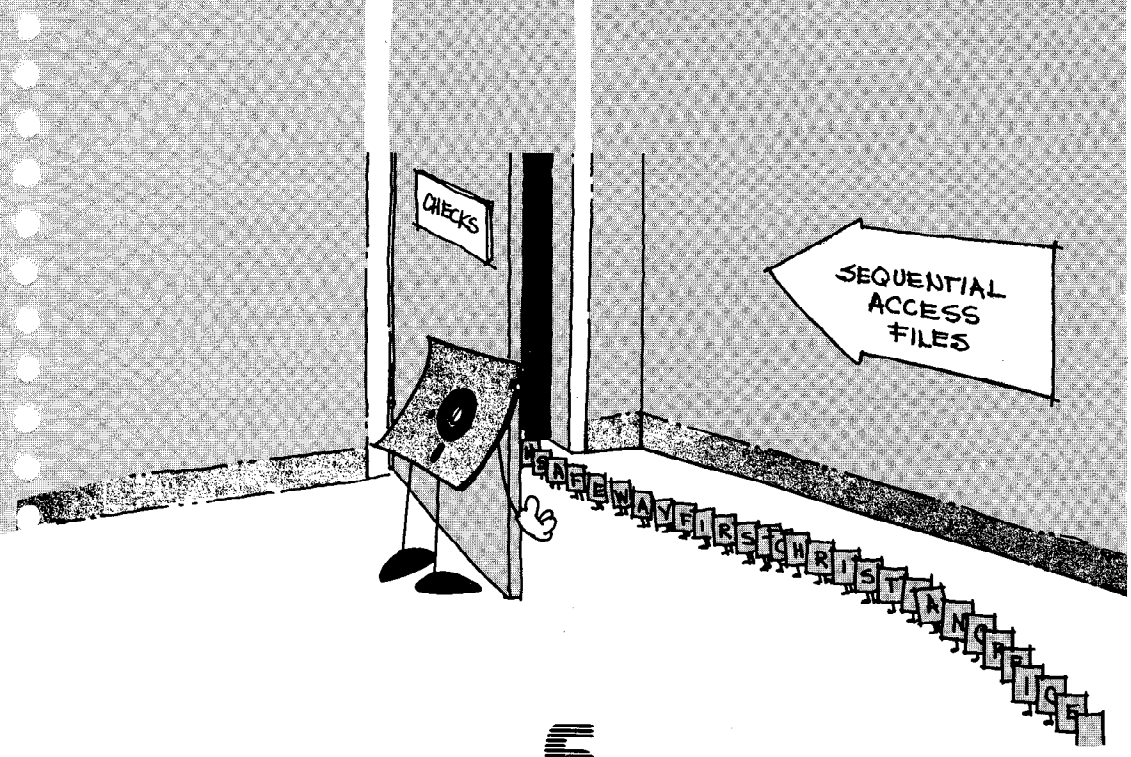
# Part 2

# THE DISK PROGRAM

Storing a BASIC program is easy. You only need to use the SAVE command. Storing data takes a little more effort. You need a program.

Some of you might prefer to buy a ready-made program. However, if you want more control and are willing to invest a little time, you can write your own.

In this part, we show you how to write a BASIC program that stores data on disk. We assume you already know some BASIC. If you don't, study the BASIC manual that came with your computer. It gives you all the background you need.




5

# One Thing at a Time

## (Sequential Access to a File)

A tape is simple. There's only one way to put data on or read data from a tape. A disk is more complex and has several ways to file your data.

In this chapter and the next, we show how to write a program that stores data in a *sequential access* disk file. This is the simplest type of file to create, and it is actually very similar to a tape file. In Chapter 7, we introduce *direct access*, an alternate type of disk file.

In showing how to store things on disk, we frequently use the words *disk file* and *disk directory*. We discussed these concepts in Chapter 2, but we'll summarize them now.

Everything you store on disk must go in a disk file and be assigned a filename. Your computer indexes the location of the disk file in the disk's directory. For example, if you want to store the names of your friends, you can put them in a disk file named FRIENDS. Your disk's directory then indexes where, on the disk, FRIENDS is stored.

There is, of course, a good reason for all of this. Using the disk filing system, the computer is able to immediately find any file on the disk.

# Writing a Disk File

Let's assume you want to write your checks on the disk:

CHECKS
DR. HORN
SAFEWAY
FIRST CHRISTIAN
OFFICE SUPPLY

We start with a short, simple program that writes the first check, DR. HORN, on the disk. Insert a formatted disk in your disk drive. (If you have two disk drives, use Drive 0.)

> **Note:** Chapter 2 shows how to format a disk. (Type **DIR** (ENTER) if you can't remember whether a disk is formatted. An unformatted disk causes an IO error.) Chapter 1 explains the drive numbers.

Then type:

```
10 OPEN "O", #1, "CHECKS/DAT"
20 WRITE #1, "DR. HORN"
30 CLOSE #1
```

Run the program. You hear the motor of the disk drive and see the red light. The computer is doing several tasks.

First, it opens communication to the disk so you can send your checks to it. Then, it finds an empty location in which to store the checks, and notes the beginning location of that disk file in the directory.

All of this happens in Line 10. Notice the meaning of O, #1, and CHECKS/DAT:

- The #1 refers to a special *buffer* area in memory called Buffer #1. This buffer communicates with the disk drive. Line 10 opens the buffer. (If you've been using tape, you might remember that Buffer # −1 communicates with the tape recorder.)

- The O is the letter O, not a zero. It stands for *output*. It tells the computer that Buffer #1 will be sending data **to** the disk.

- CHECKS/DAT is the name of the disk file. The disk's directory uses this name to index the file's beginning and ending locations.

In Line 20, the computer sends the words DR. HORN to Buffer #1, which writes the words on the disk.

Then, in Line 30, the computer closes communication with Buffer #1. In doing this, it:

1. Sends all the data remaining in Buffer #1 *to* the disk file.
2. Notes in the disk's directory where CHECKS/DAT ends.

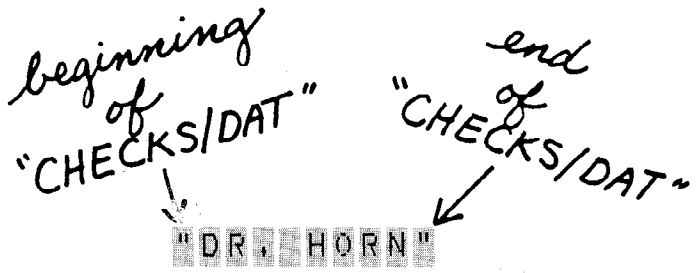
> **Note:** A buffer temporarily stores data so the computer can input data from and output data to the disk in blocks of 249 characters (bytes). Since Buffer #1 contains only eight characters (DR. HORN), the characters are not sent to the disk until you close the file.

It is important that you close communication with Buffer #1. Why? Well, let's leave Buffer #1 open. Delete Line 30 and run the program several times.

The program appears to work the same every time you run it. This is because every time you run (or load) a program, the computer automatically closes communication with any buffers left open.

Now, let's assume you switch disks and run or load a program. The computer automatically closes communication with Buffer #1. In doing this, it sends out its closing information to the new disk, thinking it's the old one. This very possibly garbles the contents of both disks.

Now that we've warned you of the importance of Line 30, re-insert this line in your program and run it again. This is what the program writes on your disk:




> **Note:** Like our drawing of the disk? The entire CHECKS/DAT file consists of the words DR. HORN. The disk's directory notes the beginning and ending locations of this file.

To verify that the computer does this, you can check the disk's directory. Type **DIR** (ENTER) .

Because this program sends your data **out** to the disk file, we call it an *output* program.

# Reading the Disk File

To get the computer to read this data from the disk back into memory, you need an *input* program. Erase the output program you now have in memory by typing **NEW** (ENTER). Then, type and run this input program:

```
100 OPEN "I", #1, "CHECKS/DAT"
110 INPUT #1, A$
120 PRINT A$
130 CLOSE #1
```

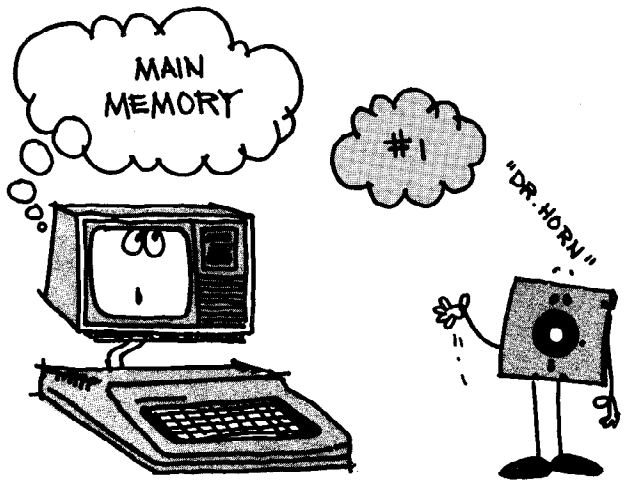
This program is actually just the reverse of the output program.

Line 100 again opens communication to Buffer #1. This time, we use the letter I (for input) in Line 100. The computer goes to the disk's directory to find where to start inputting the file named CHECKS/DAT.

In Line 110, the computer inputs the first data item from the disk file named CHECKS/DAT, and labels it A$. Line 120 prints (displays) A$.

Finally, Line 130 closes communication to Buffer #1. In doing this, the computer inputs any data remaining in the buffer.

**Note:** You can compare an input program to the LOAD command. An input program inputs a data file; LOAD inputs a program file.




# One Check at a Time

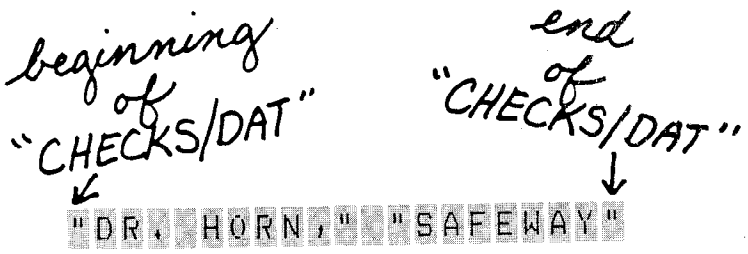
At this point, you have used an output program and an input program. Now, combine them into one program. Type:

```
10 OPEN "O", #1, "CHECKS/DAT"
20 WRITE #1, "DR. HORN"
30 CLOSE #1
100 OPEN "I", #1, "CHECKS/DAT"
110 INPUT #1, A$
120 PRINT A$
130 CLOSE #1
```

Add these lines, and run the program:

```
25 WRITE #1, "SAFEWAY"
115 INPUT #1, B$
120 PRINT A$, B$
```

Lines 10-30 output two checks into your disk file:




Lines 100-130 input them. Try to input more than two checks. Change Lines 115 and 120:

```
115 PRINT A$
120 GOTO 110
```

and run the program. The computer prints:

```
?IE ERROR IN 110
```

The computer is notifying you that you are asking it to input more checks than are in the file. Technically, the **IE** error means you've attempted to input past the end of the file.

The IE error makes things difficult when you want to input all the data, but you don't know how much is in the file. We showed you this error so you would appreciate our new word, EOF. Type:

```
105 IF EOF(1) = -1 THEN 130
120 GOTO 105
```

and run the program. EOF checks to see if you've reached the end of the buffer. If you have reached it, EOF(1) equals −1. If you haven't, EOF equals 0.

When you add Line 105 to the program, the computer checks to see if you've reached the end before inputting the next check. If you have, Line 130 closes communication to the file.

# Details...

So far, CHECKS/DAT has been easy to handle but not very useful. You would probably like to add details, such as:

| CHECKS | | |
|---|---|---|
| PAYABLE TO | AMOUNT | EXPENSE |
| DR.HORN | 45.78 | MEDICAL |
| SAFETY | 22.50 | FOOD |
| FIRST CHRISTIAN | 20.00 | CONTRIB. |
| OFFICE SUPPLY | 13.67 | BUSINESS |

Change Lines 25 and 115, and add some lines by typing:

```
25 WRITE #1, 45.78
27 WRITE #1, "MEDICAL"
110 INPUT #1, A$, B, C$
115 PRINT A$, B, C$
```

List the program. This is the way it should look now:

```
10 OPEN "O", #1, "CHECKS/DAT"
20 WRITE #1, "DR. HORN"
25 WRITE #1, 45.78
27 WRITE #1, "MEDICAL"
30 CLOSE #1
100 OPEN "I", #1, "CHECKS/DAT"
105 IF EOF(1) = -1 THEN 130
110 INPUT #1, A$, B, C$
115 PRINT A$, B, C$
120 GOTO 105
130 CLOSE #1
```

Now run the program.

# A Good, Tight Program

What if you need to store a long list of checks? You can continue to plod along with this program, but it soon becomes unbearable.

However, the following is a tight program that asks you to input all your data when you run the program. The program prompts you for the data, stores the data on disk, and reads it back into memory.

Type **NEW** (ENTER) to erase memory. Then type:

```
5 CLS
10 OPEN "O", #1, "CHECKS/DAT"
20 INPUT "CHECK PAYABLE TO :"; A$
30 IF A$ = "" THEN 80
40 INPUT "AMOUNT : $"; B
50 INPUT "EXPENSE :"; C$
60 WRITE #1, A$, B, C$
70 GOTO 20
80 CLOSE #1
90 CLS
100 PRINT "YOUR CHECKS ARE STORED ON DISK"
110 INPUT "PRESS <ENTER> TO READ THEM"; A$
120 OPEN "I", #1, "CHECKS/DAT"
130 IF EOF(1) = -1 THEN 170
140 INPUT #1, A$, B, C$
150 PRINT A$; B; C$
160 GOTO 130
170 CLOSE #1
```

Run the program. Input any checks. When you want to quit, simply press (ENTER) in answer to the **CHECK PAYABLE TO:** prompt. The following example shows the prompts and some possible responses to them.

```
CHECK PAYABLE TO :? GOODY BANK (ENTER)
AMOUNT :$? 23Ø.97 (ENTER)
EXPENSE :? CAR (ENTER)
CHECK PAYABLE TO :? (ENTER)

YOUR CHECKS ARE STORED ON DISK
PRESS <ENTER> TO READ THEM? (ENTER)
GOODY BANK 23Ø.97
CAR
```

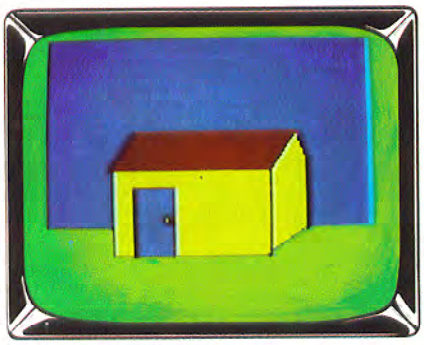
***PROGRAMMING EXERCISE 5-1***

***Write a program that prints only those checks that are for car expenses.***

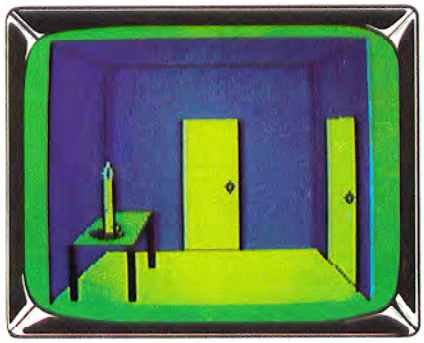
*The answers to the programming exercises are in Appendix A.*

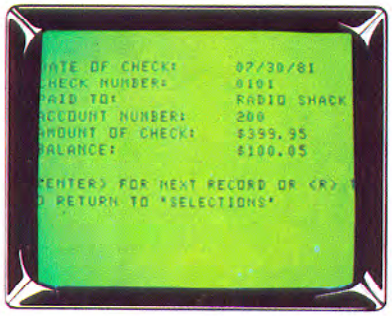
☐ ***CHAPTER CHECKPOINT***

1. ***What is Buffer #1?***
2. ***Why must you open a disk file?***
3. ***Why must you close it?***
4. ***What is the difference between a file that is open for input and one that is open for output?***



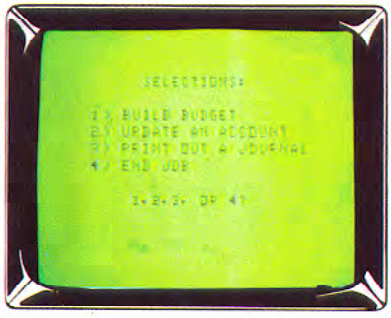
*Try saving many different graphics programs on disk and calling them from one main program. Sample Program # 7 in Appendix C shows how.*

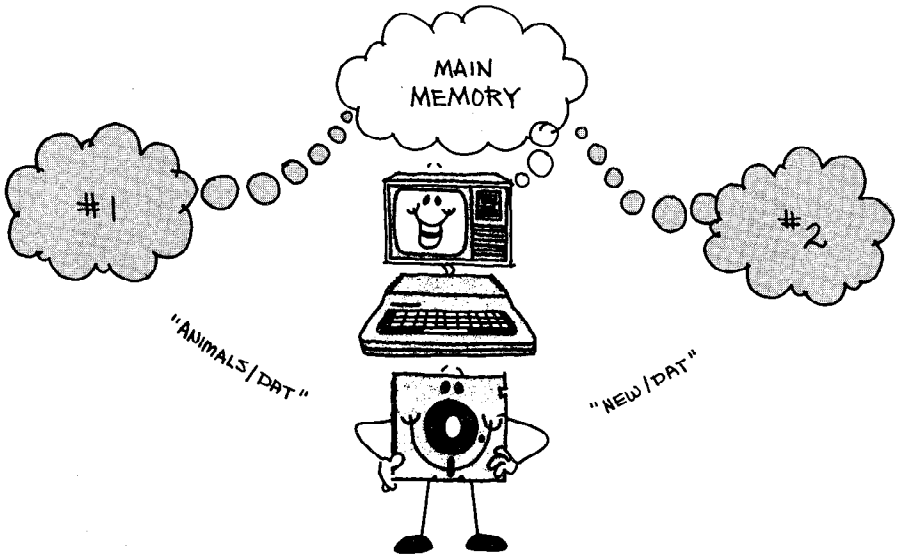






*You can quickly store, organize, and update all your financial information with a disk system. See Sample Programs 1, 2, and 8 in Appendix C for program listings.*




6

# Changing It All Around

## (Updating a Sequential Access File)

Everything you put on or take off a disk goes through a spot in memory called a buffer. When we told you how to put data on tape in your Extended Color BASIC manual, we didn't talk about these buffers. We didn't need to. Only one buffer Buffer # −1, communicates with the tape recorder.

With your disk system, you can use up to 15 buffers. This means you can have up to 15 spots in memory to communicate with 15 different disk files at the same time.

There is a particular reason for bringing up the subject of multiple buffers now. We want to demonstrate how to change some of the data in your file. To change data, it is very helpful to use two buffers.

> **Note:** In Chapter 10, we demonstrate how to take advantage of more buffers.

Type this program:

```
10 OPEN "O", #1, "ANIMALS/DAT"
20 WRITE #1, "HORSE"
30 WRITE #1, "COW"
40 CLOSE #1
```

Run it. Now, let's assume you want to change COW to GIRAFFE. First, you need to read the data items into memory with an input program. Erase memory by typing **NEW**(ENTER). Then, type:

```
5 CLS
10 OPEN "I", #1, "ANIMALS/DAT"
20 IF EOF(1) = -1 THEN 110
30 INPUT #1, A$
40 PRINT "DATA ITEM :" A$
100 GOTO 20
110 CLOSE #1
```

Then you need to add lines that let you change one of these data items and store the change in the disk file. Type:

```
50 PRINT @ 451, "PRESS <ENTER> IF NO
CHANGE";
60 PRINT @ 263, "CHANGE :";
70 INPUT X$
80 IF X$ = "" THEN X$ = A$
90 WRITE #1, X$
```

Run the program. As soon as the computer gets to Line 90, it prints:

```
?FM ERROR IN 90
```

List the program. Line 10 opens Buffer #1 to **input** data. Line 90, however, is attempting to **output** data to Buffer #1. The computer won't output data to a buffer opened for input.

This is where the additional buffer becomes handy. To output your changed data to the disk, you can open another buffer, this time for output. Add these lines:

```
15 OPEN "O", #2, "NEW/DAT"
90 WRITE #2, X$
95 CLS
120 CLOSE #2
```

Run the program. Change COW to GIRAFFE. This is the way the entire program looks:

```
5 CLS
10 OPEN "I", #1, "ANIMALS/DAT"
15 OPEN "O", #2, "NEW/DAT"
20 IF EOF(1) = -1 THEN 110
30 INPUT #1, A$
40 PRINT "DATA ITEM :" A$
50 PRINT @ 451, "PRESS <ENTER> IF NO
CHANGE";
60 PRINT @ 263, "CHANGE :";
70 INPUT X$
80 IF X$ = "" THEN X$ = A$
90 WRITE #2, X$
95 CLS
100 GOTO 20
110 CLOSE #1
120 CLOSE #2
```

Line 10 opens communication to Buffer #1 for input from a disk file named ANIMALS/DAT. Line 15 opens communication to Buffer #2 for output to a disk file named NEW/DAT.

Line 30 inputs A$ from Buffer #1. Line 70 allows you to input X$, which replaces A$. If you input X$, Line 90 outputs it. Line 90 outputs X$ to Buffer #2, which, in turn, writes X$ to NEW/DAT.

Line 110 closes communication to Buffer #1, and Line 120 closes communication to Buffer #2.

Now you have two files. ANIMALS/DAT contains the old data, and NEW/DAT contains the new. Add these lines to the program:

```
130 KILL "ANIMALS/DAT"
140 RENAME "NEW/DAT" TO "ANIMALS/DAT"
```

Run the program. The computer deletes ANIMALS/DAT from the disk. Then it renames NEW/DAT to ANIMALS/DAT—in effect, updating ANIMALS/DAT. (**Note:** If you think you might want to use the renaming program again later, save it on disk now.) Now, you can check the updated contents of ANIMALS/DAT. To do so, type **NEW** (ENTER); then, type and run the following program. (If you want to keep the last program, save it on disk.)

```
10 OPEN "I", #1, "ANIMALS/DAT"
20 IF EOF(1) = -1 THEN 60
30 INPUT #1, A$
40 PRINT A$
50 GOTO 20
60 CLOSE #1
```

To see if you understand about updating files, try these exercises:

### *PROGRAMMING EXERCISE 6-1*

*Write a program that allows you to add animals to ANIMALS/DAT.*

*Hint: You must add them to the end of the file.*

### *PROGRAMMING EXERCISE 6-2*

*Write a program that allows you to delete animals from ANIMALS/DAT.*

The next exercise is a program many of you might want, a mailing list program. Start with these lines which let you input the names, addresses, and phone numbers of your club members:

```
80 OPEN "O", #1, "MEMBERS/DAT"
90 GOSUB 430
100 IF N$="" THEN CLOSE#1:END
110 WRITE #1, N$, A$, P$
120 GOTO 90
430 CLS: PRINT "PRESS <ENTER> WHEN
    FINISHED" :PRINT
440 INPUT "NAME OF MEMBER:";N$
450 IF N$="" THEN 480
460 INPUT "ADDRESS :"; A$
470 INPUT "PHONE NUMBER :"; P$
480 RETURN
```

Now finish the program by solving this programming exercise. It is difficult, but we think you can do it. Remember, no one's watching. If you get bogged down, refer to the answer in Appendix A for help.

### *PROGRAMMING EXERCISE 6-3*

*Write a program in which you can:*

- *See the names, addresses, and phone numbers of your club's members.*
- *Change the addresses of members.*
- *Add members.*
- *Delete members.*

All of this works quite well on a small scale, but how does it work in a large file? What if you have 500 members in your MEMBERS/DAT file, and you want to change only the address of the 453rd member?

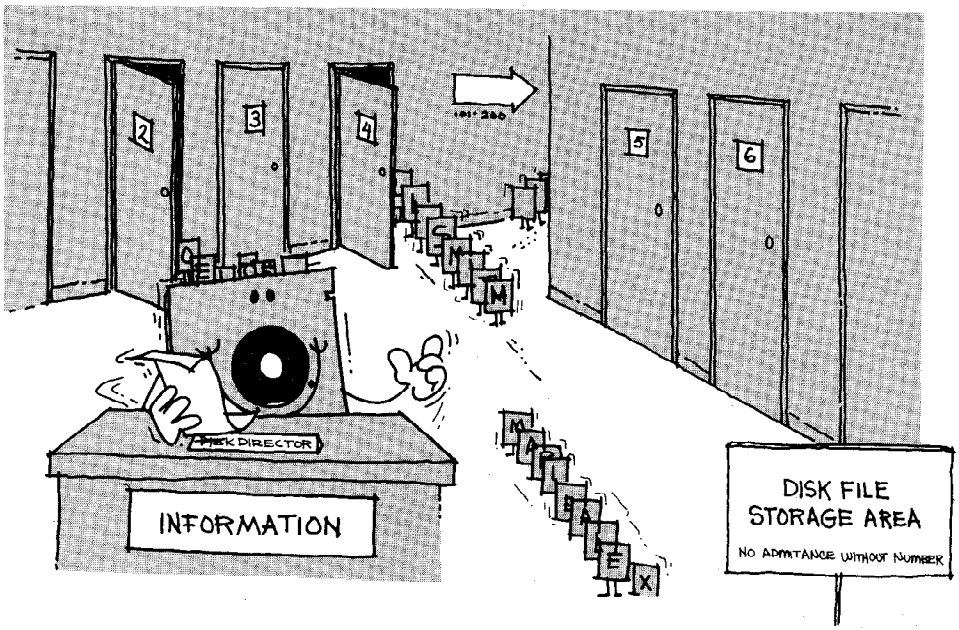
The process is still the same. You have to input each of the 500 members from one file, and then output them all to another file. All of this just to change one record. There must be an easier way!

The easier way is called the direct access method of programming. It makes your files easier and faster to update, but in many cases it takes up more space on your disk. The choice is yours. We discuss direct access in the next chapter.

**Note:** The demonstration programs we have used are short. There are many ways you could improve them. See the sample programs in Appendix C for ideas.

### ☐ *CHAPTER CHECKPOINT*

1. *Why can't you input data from and output data to the same buffer at the same time?*
2. *Can you input data from a file opened for output?*

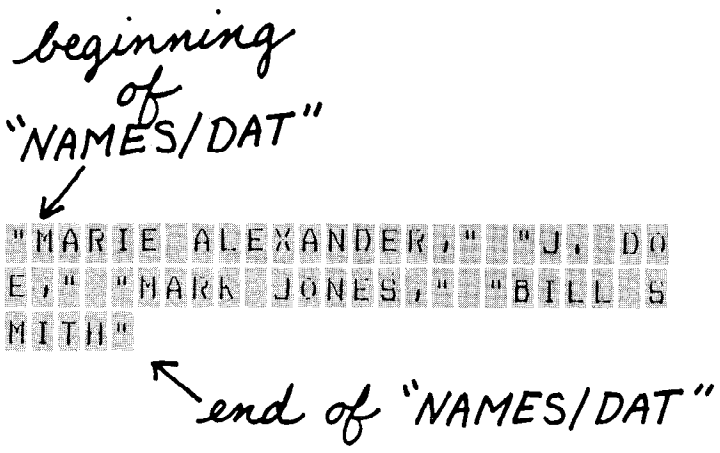



7

# A More Direct Approach

## (Direct Access to a File)

Until now, we haven't been concerned with the format in which the computer stores the data that you put on a disk. We have simply shown how to put the data there.



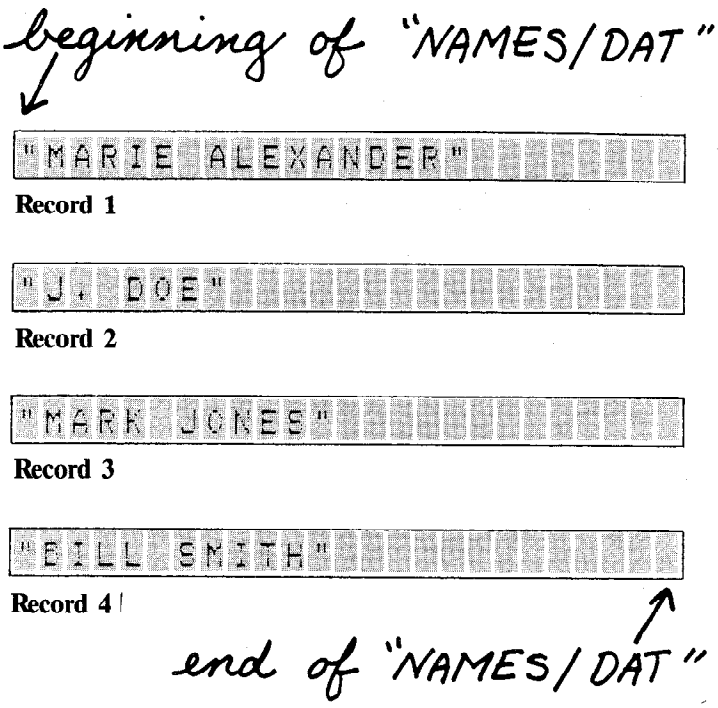

What if you want to change J. DOE to ELLIOTT HOBBS? You cannot ask the computer to go directly to J. DOE. The computer does not know where J. DOE is.

All the files created so far are sequential access. To find a particular item in a sequential access file, the computer must start at the beginning and search through each item. It can't go directly to the item. In short, a sequential access file does not take full advantage of your disk's filing system.

# Using the Disk Filing System

In Chapter 2, we talk about how formatting your disk creates this filing system. In our analogy, the file cabinets are the disk tracks and the file drawers are the disk sectors. You can use tracks and sectors to immediately find any item you want.

To do this, you can divide your file into parts called *records.* You can then write a program that stores each record in a sector and lets you put data in the records. Here is how such a disk file looks:




Because each record has the same length (the length of a sector), the computer can go directly to J. DOE. All it has to do is count to the second record.

We call this direct access. By direct access, we mean you can directly access any record you want in the file.

A direct access file has one shortcoming. Because each record is the size of one sector, it is 256 bytes long—large enough to hold 256 characters.

This means that the preceding drawing is a little misleading. If we illustrated all the empty space in each record, the records would be nearly ten times as long.

If you're a beginner, all this empty space probably won't bother you. An empty disk can hold up to 612 records. Later, when you become more comfortable with programming, you might want to pack more records into a disk file. You can then progress to Chapter 9, where we demonstrate how to make smaller records.

# Putting a Record on Disk

Enough theory! Let's put a record in a disk file. Since it is a direct access file, we don't have to start with the first record. We can start with the second. Erase memory and type:

```
10 OPEN "D", #1, "NAMES/DAT"
20 WRITE #1, "J. DOE"
30 PUT #1, 2
40 CLOSE #1
```

The program looks familiar, except for the word PUT in Line 30 and the D in Line 10. We'll talk about that in detail later.

Now, add some lines so that the computer reads this record back into its main memory. Type:

```
34 GET #1, 2
36 INPUT #1, A$
38 PRINT A$
```

Note that Line 34 uses another new word, GET. *Hmmm*...any ideas? Look at the entire program:

```
10 OPEN "D", #1, "NAMES/DAT"
20 WRITE #1, "J. DOE"
30 PUT #1, 2
34 GET #1, 2
36 INPUT #1, A$
38 PRINT A$
40 CLOSE #1
```

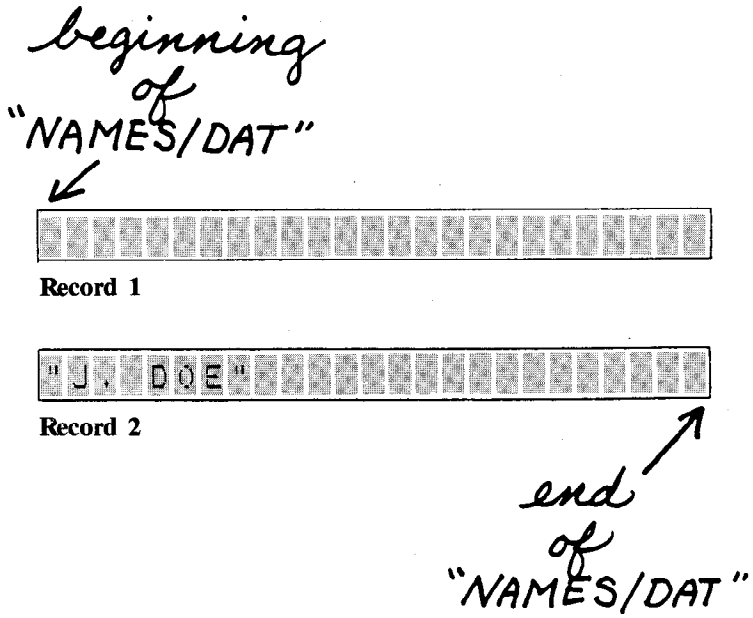
Run it. You hear the now-familiar sound from your disk drive. The computer is writing J. DOE in the disk file and then reading it back into memory. Here's how.

Line 10 opens Buffer #1, which communicates with a disk file named NAMES/DAT.

Communication is being opened for D. The D stands for direct access. With direct access, unlike with sequential access, you don't have to specify whether you're opening communication for output or input. The D suffices for both.

Line 20 writes J. DOE to Buffer #1. Since this program is open for direct access, J. DOE remains in Buffer #1 until the program sends it elsewhere.

Line 30 does just that. It puts the contents of Buffer #1 into the disk file as Record 2:



At this point, J. DOE is no longer in Buffer #1. It is in Record 2 of the disk file.

Line 34 gets Record 2, and reads it back into Buffer #1. Now J. DOE is in both the disk file and Buffer #1.

Line 36 inputs the record from Buffer #1 into main memory, and labels it A$. Now, J. DOE is in both the disk file and main memory. It is no longer in Buffer #1.

With J. DOE in main memory, Line 38 can print it.

> **Note:** In the sequential access programs in Chapters 5 and 6, you didn't need PUT and GET. The computer performed both operations automatically. The OPEN line specified whether the buffer should output (put) data into the disk file or input (get) data from the disk file.

Notice that the drawing shows only two records in the file. Get Record 4. Type:

```
34 GET #1, 4
```

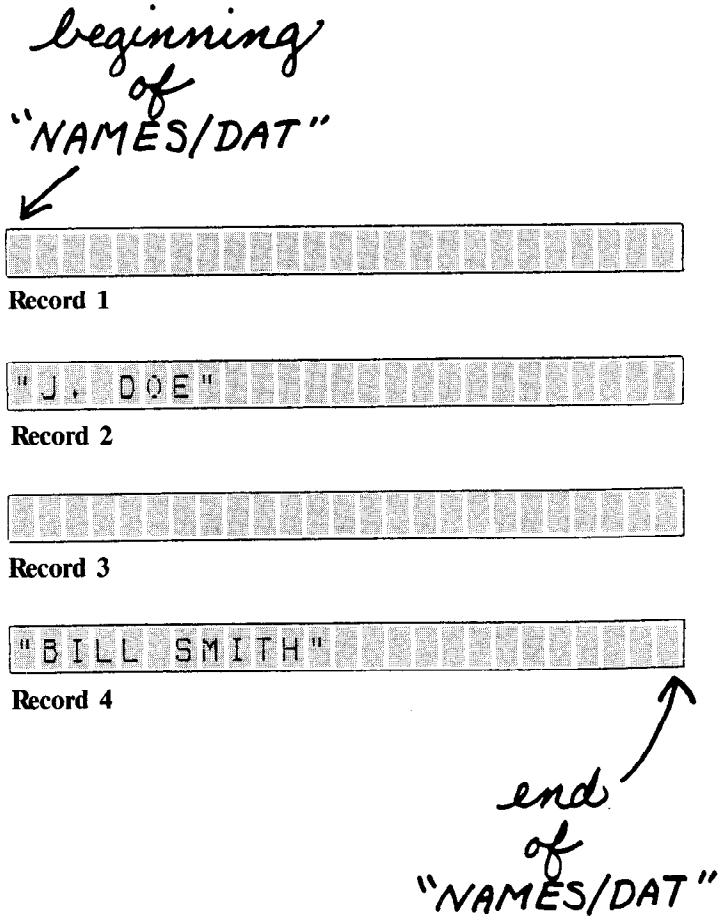
Run the program. The computer displays an **IE** (input past the end of the file) error. This is because the last record the program put in the file was Record 2. Hence, Record 2 became the end of the file.

> **Note:** Didn't get this error? You must already have a NAMES/DAT file on your disk with three or more records.

To put more records in the file, add these lines. Then, run the program:

```
31 WRITE #1, "BILL SMITH"
32 PUT #1, 4
```

Now, your NAMES/DAT file has these four records:




***PROGRAMMING EXERCISE 7-1***

*Change Lines 32 and 34 so that your computer uses Record 3 to put and get BILL SMITH.*

# Dealing With Garbage

You have not yet put anything in Record 1. Ask the computer to get Record 1, and see what happens. Type this; then, run the program again:

```
34 GET #1, 1
```

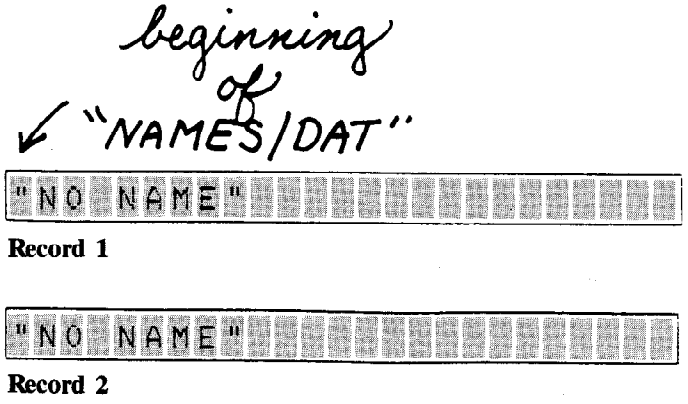
Because the computer didn't put anything in Record 1, Record 1 contains whatever *garbage* is already there.

When you ask the computer to get and input, it either gets the garbage or gives you an **OS** (out of string space) error. The **OS** error means that the garbage consumes more than 200 bytes (characters).

Because your empty records contain garbage until you fill them with something, it's a good idea to put some kind of data in all of them in advance. Erase memory, and type this program:

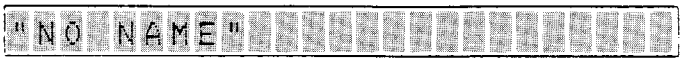
```
10 OPEN "D", #1, "NAMES/DAT"
20 FOR X = 1 TO 10
30 WRITE #1, "NO NAME"
40 PUT #1, X
50 NEXT X
60 CLOSE #1
```

Run it. This program sets up a disk file named NAMES/DAT, which has ten records. Each record contains NO NAME:




**Record 3**


**Record 4**



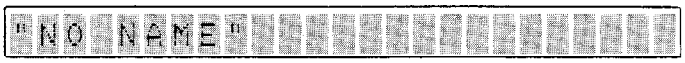

**Record 5**


**Record 6**


**Record 7**


**Record 8**


**Record 9**




**Record 10**

end of "NAMES/DAT"

Now, erase memory and type this:

```
10 OPEN "D", #1, "NAMES/DAT"
20 CLS : INPUT "RECORD NO. (1-10)"; R
30 IF R>10 THEN 20
40 IF R<1 THEN 130
50 GET #1, R
60 INPUT #1, A$
70 PRINT A$ : PRINT "-- IS THE NAME
   IN RECORD"R
80 PRINT : LINE INPUT "TYPE NEW NAME OR
   PRESS <ENTER> "; A$
90 IF A$ = "" THEN 20
100 WRITE #1, A$
110 PUT #1, R
120 GOTO 20
130 CLOSE #1
```

Run the program. See how all your records initially contain NO NAME. Then, you can change the data in any of the records at will, as many times as you want. To end the program, input **0** in response to the **RECORD NO.** prompt.

# Reading All Records

At this point, you might like the computer to print all the records in your NAMES/DAT file, with their appropriate record numbers. Save your program, if you want. Erase memory by typing **NEW**(ENTER). Then, type and run:

```
5 CLS
10 OPEN "D", #1, "NAMES/DAT"
20 R = 1
30 GET #1, R
40 INPUT #1, A$
50 PRINT A$ "-- IS IN RECORD" R
60 IF R = 10 THEN 90
70 R = R + 1
80 GOTO 30
90 CLOSE #1
```

Line 20 makes R equal to 1. In the next lines, the computer gets, inputs, and prints (displays) Record 1.

Line 70 then makes R equal to 2, and the whole process is repeated with Record 2. When R equals 10, the last record in the file, the program ends.

There are many occasions when you don't know the last record number in the file. Change Line 60 and run the program:

```
60 IF R = LOF(1) THEN 90
```

LOF looks at the file with which Buffer #1 (the number in parentheses) is communicating. It tells the computer the last record number in that file.

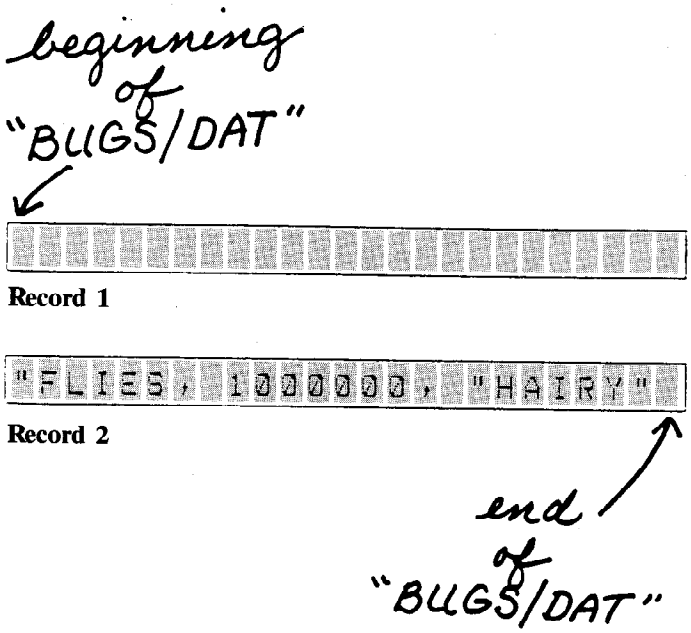
# More Power to a Record

So far, we put only one *field* of data in each record. We can make the file more organized by subdividing each record into several fields.

Erase memory. Then, type and run this program:

```
10 OPEN "D", #1, "BUGS/DAT"
20 WRITE #1, "FLIES", 1000000, "HAIRY"
30 PUT #1, 2
34 GET #1, 2
36 INPUT #1, D$, N, T$
38 PRINT D$, N, T$
40 CLOSE #1
```

Line 20 writes three fields of data into Buffer #1. Then, Line 30 puts the entire contents of Buffer #1 (all three fields) into Record 2 of the file BUGS/DAT.




Line 34 gets everything in Record 2, and reads it into Buffer #1. Then, Line 36 inputs all three fields of data from Buffer #1, and labels them as D$, N, and T$.

Try substituting this for Line 36:

```
36 INPUT #1, D$
```

Run the program. Since this line asks the computer to input only the first field of data in Buffer #1, the computer inputs only FLIES.

### PROGRAMMING EXERCISE 7-2

*What do you think the computer displays if you run the program, using this for Line 36? Why?*

```
36 INPUT #1, N
```

### PROGRAMMING EXERCISE 7-3

*Change the program that stores the NAMES/DAT file so that each record contains five fields of data:*

1. *name*
2. *address*
3. *city*
4. *state*
5. *zip*

### ☐ CHAPTER CHECKPOINT

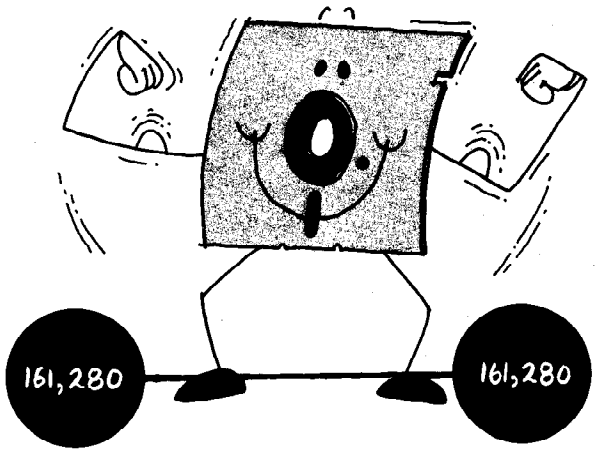
1. *What are records? Why must you use them to access data directly?*
2. *What are fields?*
3. *What is the difference between a sequential access file and a direct access file?*
4. *Why is it quicker to update a direct access file?*

# Part 3

# THE REFINED DISK PROGRAM

After writing disk programs for a while, you might want to make them more efficient. Perhaps you'll want to put more data on the disk. You might also want to economize on memory space or use some extra buffer space.

At that time, we invite you to read this section. The subject matter is more advanced and technical. Once you finish it, though, you'll have all the information you need to write the best possible disk programs.




8

# How Much Can One Disk Hold?

## (What the Computer Writes in a Disk File)

Your disk is divided into thousands of equal-sized units. Each unit is a byte. Each byte can hold one character. Thus, the word STRAW consumes five bytes of disk space.

An empty disk contains 161,280 bytes. The directory takes up 4,608 of these bytes, leaving you with 156,672 bytes for your disk files.

> **Note:** Your Color Computer formats each disk to have 35 tracks, and the directory takes up one track. Each track contains 18 (256-byte) sectors; so, there are 4,608 bytes per track (18 sectors per track × 256 bytes per sector). Therefore, each disk has 156,672 bytes for your disk files (4,608 bytes per track × 34 tracks per disk).

Does this mean you can use the entire 156,672 bytes for data? Possibly. There are two factors that determine the answer to this question.

The first factor has to do with the way the computer allocates space for a disk file. It stores a file in clusters. (We call them granules.) Each granule contains 2,304 bytes.

Because of this, all of your disk files contain a multiple of 2,304 bytes. If your file contains 2,305 bytes of data, for example, the computer allocates two granules for it, or 4,608 bytes (2,304 x 2).

The computer allocates file space in this manner because it's the most efficient way to create a file. It is very tricky to change this and is something that only very technical people would want to do. (See Chapter 11, "Technical Information," for additional information.)

The second factor that affects the amount of data that you can put in a disk file is your program. Some disk programs are very efficient. Others put a lot of overhead and empty space in the file.

The next two chapters compare eight different types of programs. Each program stores the same two records in a disk file named OFFICE/DAT. One record contains the data 5, PEN. The other contains the data 16, PAPER. The amount of overhead and empty space that each program puts in OFFICE/DAT varies greatly.

# Writing on the Disk

Program 1 uses WRITE to put the data on the disk. Type and run it:

**PROGRAM 1**
**21 bytes**

```
10 OPEN "O", #1, "OFFICE/DAT"
20 WRITE #1, 5, "PEN"
30 WRITE #1, -16, "PAPER"
40 CLOSE #1
```

There is an easy way to see the data that Lines 20 and 30 wrote on your disk. Type the two lines as they are above, but this time leave off the program line number and the #1. Leaving off the #1 prevents the computer from writing the data on your disk (via Buffer #1). The computer writes it on your screen instead. Type:

```
WRITE 5, "PEN" (ENTER)
WRITE -16, "PAPER" (ENTER)
```

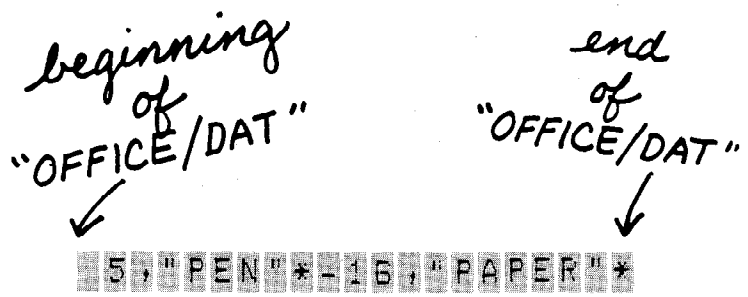
Look carefully at what the computer writes. Every blank space and punctuation mark counts.

Notice the way the computer writes the two strings (PEN and PAPER). It puts quotation marks around them. It writes the numbers (5 and −16) differently. If a number is negative, the computer puts a minus sign in front of it. If a number is positive, the computer puts a blank space in front of it.

There are two characters you typed that the computer didn't write on the screen. These are the two (ENTER) characters you typed at the end of the WRITE lines. The computer skipped down to the next line instead:

```
 5, "PEN"
OK
-16, "PAPER"
OK
```

When writing on the disk, the computer **does** write each (ENTER) character. The following illustration shows what Program 1 writes on your disk. (Asterisks represent the (ENTER) characters):




> **Note:** Want to be precise? What the computer actually writes on the disk are binary codes. Each character has an ASCII code that the computer converts to a binary number. (See Appendix D.)

Count the characters. (**Note:** Each blank space, comma, quotation mark, (ENTER) character, and minus sign counts as one character, as does each letter and numeral.) You should come up with 21 characters. Did you remember the blank space preceding the number 5? Program 1 puts 21 bytes in OFFICE/DAT.

Since the computer allocates file space in clusters, OFFICE/DAT actually consumes one granule of disk space, or 2,304 bytes. However, for the purpose of comparison, we'll look only at the 21 bytes that Program 1 puts in OFFICE/DAT.

# A Disk-Eye View

To input OFFICE/DAT, type and run this INPUT program. (Erase memory first.)

**INPUT PROGRAM**

```
10 CLS
20 OPEN "I", #1, "OFFICE/DAT"
30 IF EOF(1) = -1 THEN 80
40 INPUT #1, A, B$
50 PRINT: PRINT "DATA ITEM:"A
60 PRINT "DATA" ITEM:" B$
70 GOTO 30
80 CLOSE #1
```

The program inputs your data items. However, it does not input the quotation marks, commas, and blank spaces that are interspersed with your data.

To see what Program 1 actually wrote on your disk, you can use a LINE INPUT program. First, save the INPUT program you now have in memory. (You use it later.)

Now, change the INPUT program to a LINE INPUT program. Delete Line 50 and change Lines 40 and 60 by typing the following lines:

```
40 LINE INPUT #1, L$
50
60 PRINT "DATA LINE :" L$
```

Run the program. Line 40 inputs an entire line, rather than one single data item, from the disk file. This line includes everything up to the (ENTER) character — punctuation marks, spaces, and all.

In the OFFICE/DAT file, the first line contains 5, PEN. Line 40 labels this line as L$, and Line 60 prints the line on your screen.

The program then inputs and prints −16, PAPER — the second and final line in the file.

You can easily alter this program so that it counts the number of bytes in the file. Add these lines, and run the program.

```
25 PRINT "THIS FILE CONTAINS :"
27 PRINT: PRINT: PRINT: PRINT
57 M$ = L$+ "*"
60 PRINT M$;
65 L = LEN(M$) + L
90 PRINT @ 394, L "BYTES"
```

Line 57 adds an asterisk to each line. This asterisk represents the (ENTER) character. Line 65 then counts the number of characters in each line.

This is the entire LINE INPUT program:

**LINE INPUT PROGRAM**

```
10 CLS
20 OPEN "I", #1, "OFFICE/DAT"
25 PRINT "THIS FILE CONTAINS :"
27 PRINT: PRINT: PRINT: PRINT
30 IF EOF(1) = -1 THEN 80
40 LINE INPUT #1, L$
57 M$ = L$ + "*"
60 PRINT M$;
65 L = LEN(M$) + L
70 GOTO 30
80 CLOSE #1
90 PRINT @ 394, L "BYTES"
```

Save the program. It will be useful in comparing the data that Programs 2, 3, and 4 put in your disk file.

# Print — For a Change

So far, we use only WRITE to put data in a disk file. If you've used other forms of BASIC, you might be accustomed to using PRINT rather than WRITE.

The Color Computer disk system allows you to use either PRINT or WRITE. However, in some cases, PRINT can be more tricky to use than WRITE. If you're not used to it, you might want to skip to Program 4.

Are you still with us? Delete your old OFFICE/DAT file by typing:

```
KILL "OFFICE/DAT" (ENTER)
```

Now, erase memory, and type and run Program 2. Then run the INPUT or the LINE INPUT program, if you like.

Here's Program 2:

**PROGRAM 2**
**42 bytes**

```
10 OPEN "O", #1, "OFFICE/DAT"
20 PRINT #1, 5, "PEN"
30 PRINT #1, -16, "PAPER"
40 CLOSE #1
```

Lines 20 and 30 print your data to Buffer #1, which, as you know, is one of the 15 buffers that will send your data to the disk file. To see what Program 2 prints, type:

```
PRINT 5, "PEN" (ENTER)
PRINT -16, "PAPER" (ENTER)
```

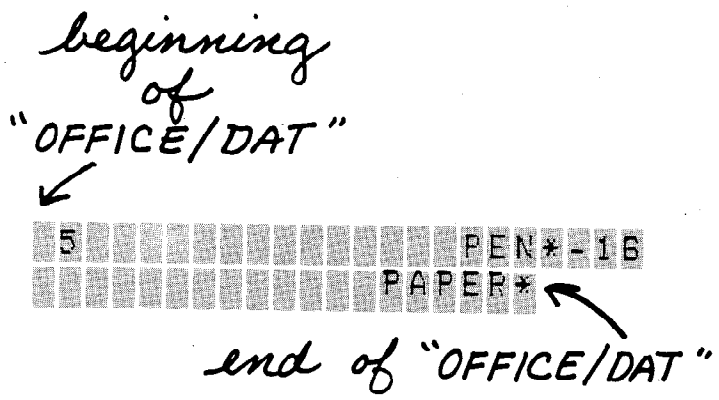
Notice that the computer does not enclose the strings, PEN and PAPER, in quotation marks, as WRITE does. This is important to know and is discussed later.

Now, look at the blank spaces. Start with the first one, the one before the 5. This space means the same thing it did with WRITE. It indicates that 5 is a positive number.

Now for the other blank spaces... Whenever the computer prints a number, it prints one *trailing* blank space after it. This explains the first blank space after the 5 and the −16.

How about all the additional spaces? They are caused by the commas. Using a comma in a PRINT line causes the computer to print your data in columns, inserting spaces between the columns.

The computer prints every one of these blank spaces in your disk file:




Count the characters. Program 2 puts 42 bytes into OFFICE/DAT.

**Note:** Are you still unclear about the effect of commas in a PRINT line? Type some more PRINT lines with commas between data items:

```
PRINT 1, 2, 3, 4, 5, 6, 7, 8 (ENTER)
PRINT "HORSE", "COW", "RABBIT", "DOG"
(ENTER)
```

# Printing Less

You might feel that all the blank spaces that PRINT inserts in your disk file are a waste of space. They are. The way to get around this waste is to use semicolons. Semicolons in a PRINT line compress your data. Type:

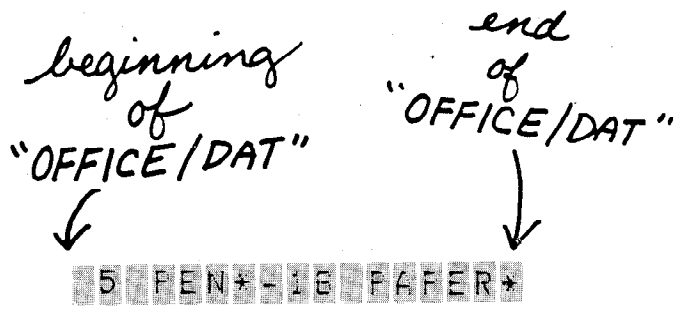
```
PRINT 5; "PEN" (ENTER)
PRINT -16; "PAPER" (ENTER)
```

You can compress your data on the disk in the same manner. Erase memory and delete your old OFFICE/DAT file. Then, type and run this program:

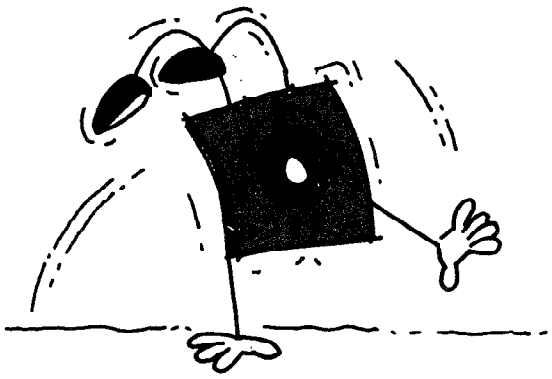
**PROGRAM 3**
**17 bytes**

```
10 OPEN "O", #1, "OFFICE/DAT"
20 PRINT #1, 5; "PEN"
30 PRINT #1, -16; "PAPER"
40 CLOSE #1
```

This is what Program 3 prints on your disk. (Use the LINE INPUT program to test this, if you'd like):




Very efficient. The data takes only 17 bytes. There are only three blank spaces in this disk file. There is a space before the 5 (to indicate that it is positive) and spaces after 5 and −16 (to indicate that they are numbers). There are no blank spaces around the strings.

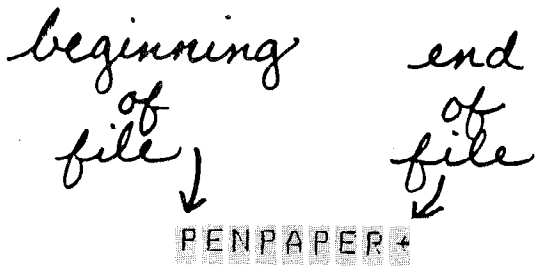


# The Tricky Part

Certain types of PRINT lines can be tricky. (We did caution you, didn't we?) Type:

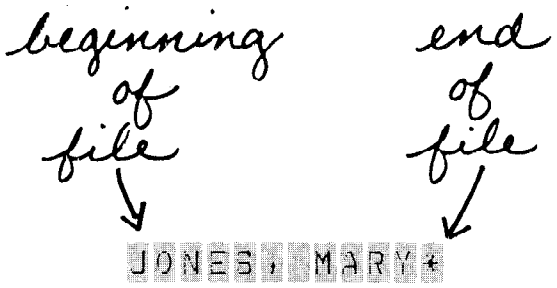
```
PRINT "PEN"; "PAPER" (ENTER)
PRINT "JONES, MARY" (ENTER)
PRINT "PEN", 5 (ENTER)
```

The line **PRINT #1, "PEN"; "PAPER"** (in your disk program) would print this in your disk file:



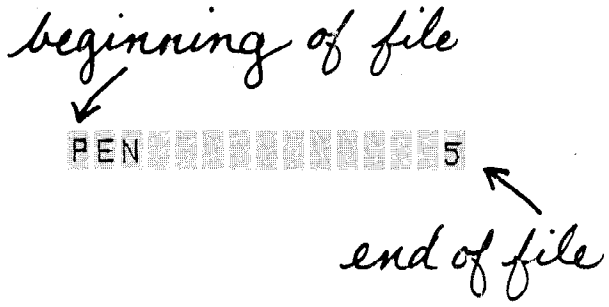
The computer reads PENPAPER back into memory as one item. (Reason: there is not a *delimiter* — a comma, quotation mark, or space — to separate PEN from PAPER.)

The line **PRINT #1, "JONES, MARY"** prints this in your disk file:




The computer reads JONES, MARY back as two items: JONES and MARY. (Reason: The computer interprets the comma as a delimiter.)

The line **PRINT #1, "PEN", 5**, prints this in your disk file:




The computer reads PEN 5
(with all the blank spaces) back into memory as one item. (Reason: although the computer normally interprets a blank space as a delimiter, it does not interpret the space this way if the space follows a string and precedes a number.

# An Attractive Disk File

PRINT USING is another command you can substitute for WRITE. Type:

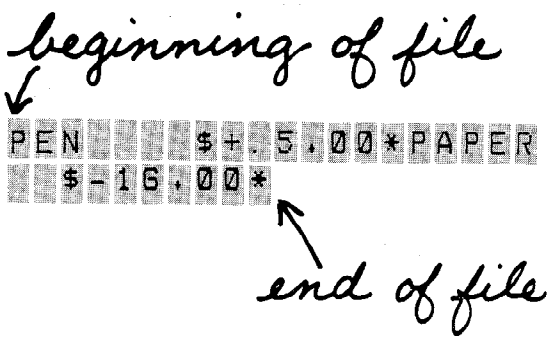
```
PRINT USING "%    %$+##,##"; "PEN", 5
  ENTER
PRINT USING "%    %$+##,##"; "PAPER", -16
  ENTER
```

You can get the computer to print these same images on your disk with the following program. Delete OFFICE/DAT, and erase memory. Then, type and run:

**PROGRAM 4**
**32 bytes**

```
10 OPEN "O", #1, "OFFICE/DAT"
20 PRINT #1, USING "%    %$+##,##";
   "PEN", 5
30 PRINT #1, USING "%    %$+##,##";
   "PAPER", -16
40 CLOSE #1
```

The program prints the following in your disk file:




**Note:** There are five blank spaces between the percent signs (%) in Lines 20 and 30. Counting the two percent signs, this string field (for printing PEN and PAPER) contains seven bytes.

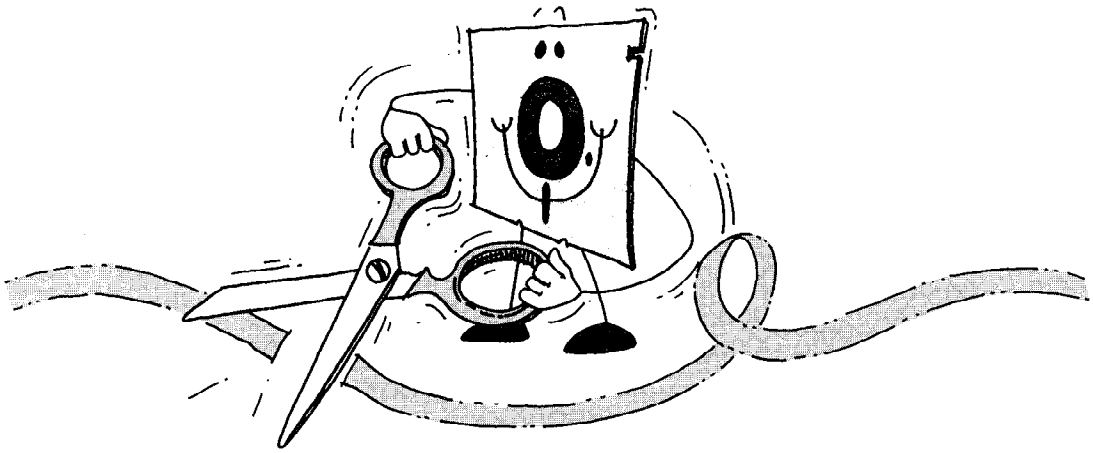
Now, the data is in an attractive print format. You can input and print it using a simple LINE INPUT program. Erase memory. Then, type and run:

```
10 OPEN "I", #1, "OFFICE/DAT"
20 IF EOF(1) = -1 THEN 60
30 LINE INPUT #1, A$
40 PRINT A$
50 GOTO 20
60 CLOSE #1
```

All the files created in this chapter are sequential access. The next chapter compares four more programs, which put the same data in direct access files.

### ☐ CHAPTER CHECKPOINT

1. What is the minimum size of a disk file? Why can't it be smaller?
2. How does the computer write numbers in a disk file?
3. How does it write strings when you use WRITE?
4. What is the difference between INPUT and LINE INPUT?
5. What does a comma in a PRINT line cause the computer to do?
6. What does a semicolon in a PRINT line cause it to do?
7. How does the computer print strings when you use PRINT?



# 9

# Trimming the Fat Out of Direct Access

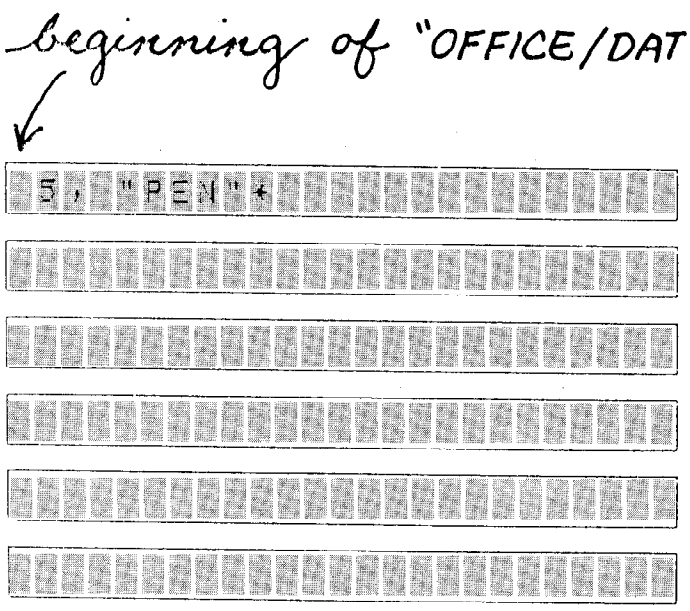
## (Formatting a Direct Access File)

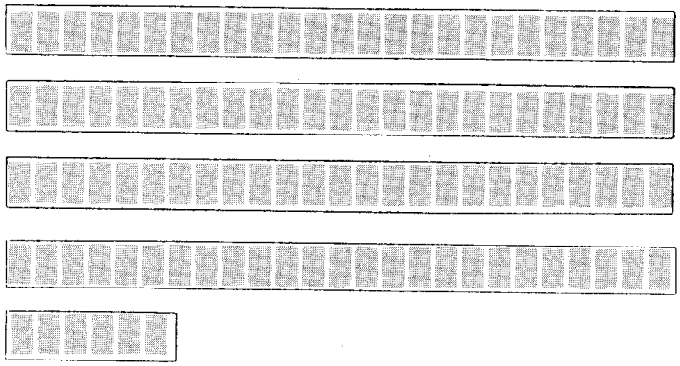
Direct access files often contain a lot of empty space. For example, the first program in this chapter is very similar to Program 1 from the last chapter. The WRITE lines are identical. However, because Program 5 uses direct access, it puts 512 bytes in OFFICE/DAT.

**PROGRAM 5**
**512 bytes**

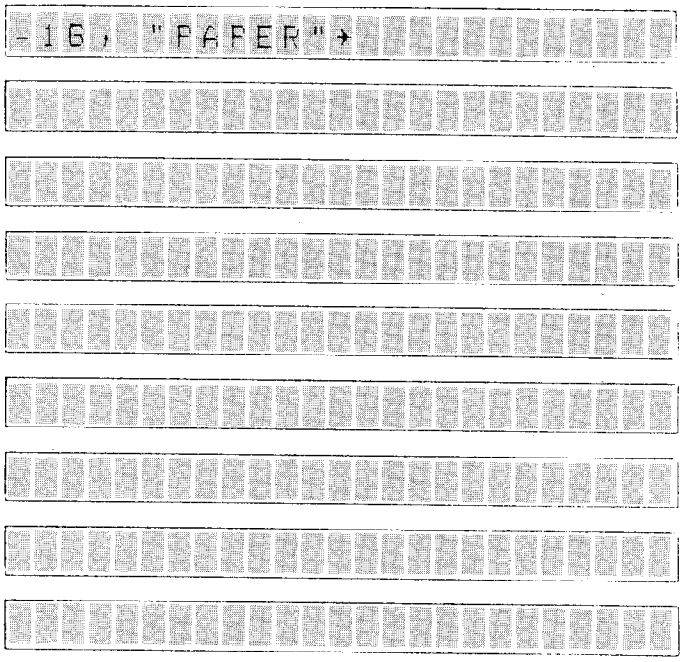
```
10 OPEN "D", #1, "OFFICE/DAT"
20 WRITE #1, 5, "PEN"
30 PUT #1, 1
40 WRITE #1, -16, "PAPER"
50 PUT #1, 2
60 CLOSE #1
```

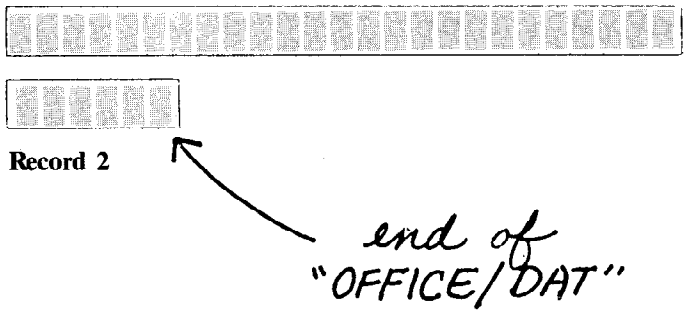
A direct access program puts your data inside records. Each record is 256 bytes long. Program 5 puts two records in the OFFICE/DAT file. Therefore, the file consumes 2×256, or 512 bytes:





**Record 1**






This obviously wastes a massive amount of space. Notice what the computer actually writes in each record:

```
 5, "PEN"*
-16, "PAPER"*
```

This is the same as what Program 1 wrote. Count the bytes. There are nine bytes in the first record and 12 in the second. You need to know this for the next program.

> **Note:** You can use PRINT or PRINT USING rather than WRITE. If you do, the computer prints your data inside each record using the PRINT or PRINT USING format.

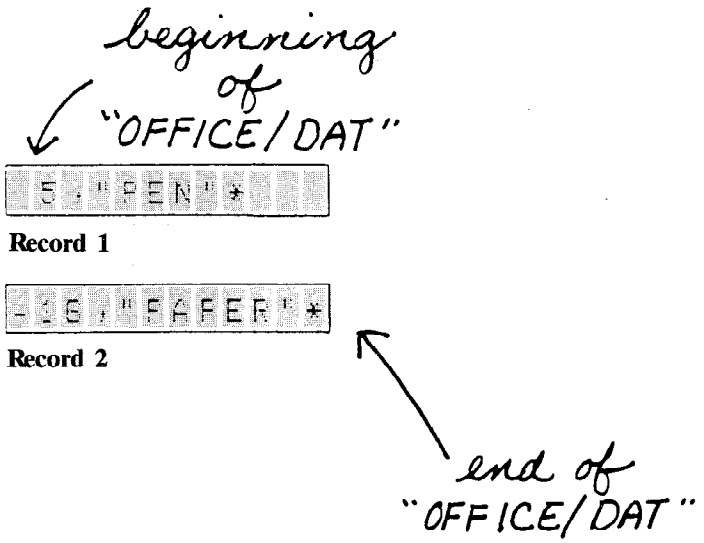
# Trimming the Fat

Program 6 is the same as Program 5, except that we inserted the number 12 at the end of Line 10. This number tells the computer to make each record 12 bytes long:

**PROGRAM 6**
**24 bytes**

```
10 OPEN "D", #1, "OFFICE/DAT", 12
20 WRITE #1, 5, "PEN"
30 PUT #1, 1
40 WRITE #1, -16, "PAPER"
50 PUT #1, 2
60 CLOSE #1
```

and cuts the file down immensely:



In a direct access file, all records must be the same length. (We explained why in Chapter 7.) If you don't tell the computer how long to make the records, it makes them 256 bytes long.

In Program 6, we make each record 12 bytes long, the size of the largest record. Type and run Program 6, if you like. (Be sure to erase memory and delete your old OFFICE/DAT file first.) After running Program 6 you can use the following program to input the file:

**DIRECT INPUT PROGRAM**

```
10 OPEN "D", #1, "OFFICE/DAT", 12
20 R = R + 1
30 GET #1, R
40 INPUT #1, A, B$
50 PRINT "RECORD" R ":" A; B$
60 IF LOF(1) <> R THEN 20
70 CLOSE #1
```

**Note:** You can't use the LINE INPUT program to determine how many bytes this file consumes. LINE INPUT does not input the spaces that follow the (ENTER) characters.

# Efficiency, Efficiency...

We can get even more efficient. Our next direct access program consumes only 16 bytes. Erase memory, delete the old OFFICE/DAT file, and type and run Program 7.

**PROGRAM 7**
**16 bytes**

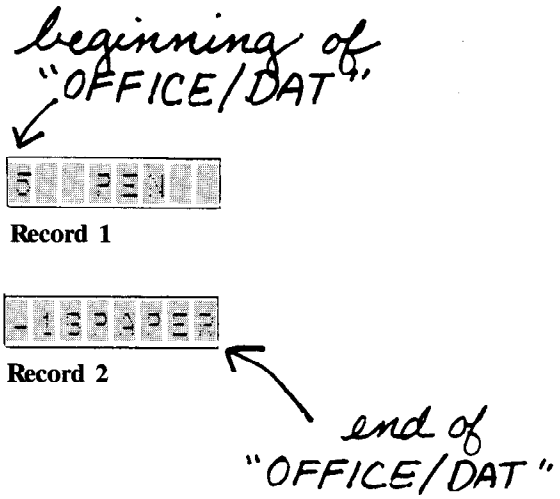
```
10 OPEN "D", #1, "OFFICE/DAT", 8
20 FIELD #1, 3 AS A$, 5 AS B$
30 LSET A$ = "5"
40 LSET B$ = "PEN"
50 PUT #1, 1
60 LSET A$ = "-16"
70 LSET B$ = "PAPER"
80 PUT #1, 2
90 CLOSE #1
```

This program contains two words (FIELD and LSET) that we talk about later. Run the program and save it. Then, erase memory and use the following program to input the file.

**FIELDED INPUT PROGRAM**

```
10 OPEN "D", #1, "OFFICE/DAT", 8
20 FIELD #1, 3 AS A$, 5 AS B$
30 R = R + 1
40 GET #1, R
50 PRINT "RECORD" R ":" A$; B$
60 IF LOF(1) <> R THEN 30
70 CLOSE #1
```

By using FIELD and LSET, Program 7 works the same as any direct access program. The difference is in what FIELD and LSET put in each record:




Here's how Program 7 works:

Line 20 tells the computer to divide each record into two fields. The first field is A$ and the second is B$. These two fields are the same size in every record. A$ is always three bytes long, and B$ is always five bytes long.

Once the program establishes the size of the fields, it can put data in each field. The first data item is 5. The word LSET sets the character 5 as far left as possible in the A$ field. Because the character 5 consumes only one byte, and because there are three bytes in the A$ field, there are two empty spaces following the 5.

In looking at Line 30, notice that you had to convert the number 5 to a string (by putting quotation marks around it). You cannot use LSET on numeric data. You must convert the data beforehand.

Line 40 left-justifies the word PEN in the B$ field. This leaves two empty spaces at the end of B$, since PEN uses only three bytes.

Line 50 puts all this data in Record 1. Then, the program repeats the entire process for Record 2.

Now, look at the Fielded INPUT program. Notice the FIELD line. Run the program without Line 20, and see what happens. Type:

```
20 ENTER
RUN ENTER
```

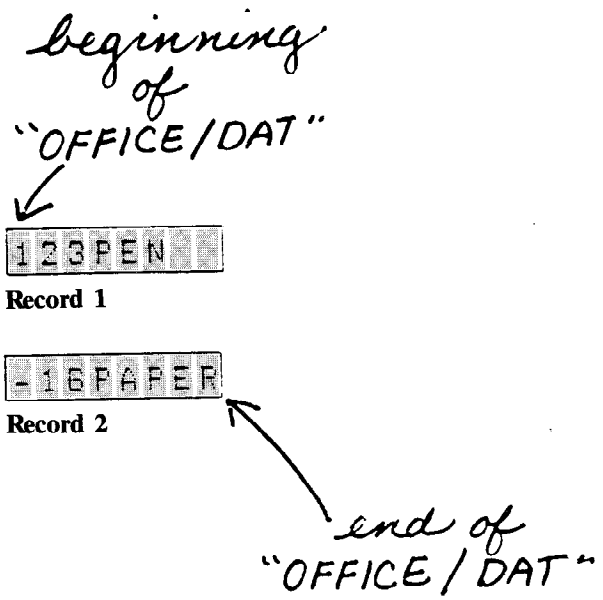
Without a FIELD line, the computer does not know where the two fields are. Whenever you input FIELDed records, use a FIELD line in your input program.

Re-insert Line 20, and save the Fielded INPUT program. Can you guess what the computer does if you try to left-justify a long string, such as 123456789, into one of the fields? Load Program 7, change Line 30 as follows, and run the program:

```
30 LSET A$ = "123456789"
```

Now, load and run the Fielded INPUT program.

A$ is only three bytes long. Therefore, the computer left-justifies only the first three bytes of 123456789. It chops off the remaining characters:



More on this later... Before going on to the next program, try writing your own fielded program:

### *PROGRAMMING EXERCISE 9-1*

*Write a direct access program to put a mailing list in a disk file. Make each record 57 bytes with these six fields:*

*1. last name — 15 bytes*
*2. first name — 10 bytes*
*3. address — 15 bytes*
*4. city — 10 bytes*
*5. state — 2 bytes*
*6. zip code — 5 bytes*

### *PROGRAMMING EXERCISE 9-2*

*Write a program to input the file you created in Exercise 9-1.*

# A Number is a Number,...

Let's assume you put several numbers in your disk file. Every number might be a different length:

−5.237632  31  673285

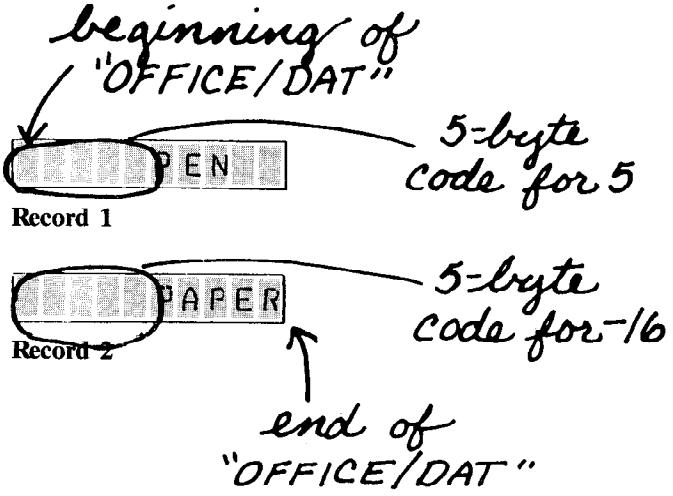
However, it is very important that the computer doesn't chop off any of the digits. Doing so might change the number's value entirely.

The word MKN$ solves this problem:

**PROGRAM 8**
**20 bytes**

```
10 OPEN "D", #1, "OFFICE/DAT", 10
20 FILED #1, 5 AS A$, 5 AS B$
30 LSET A$ = MKN$(5)
40 LSET B$ = "PEN"
50 PUT #1, 1
60 LSET A$ = MKN$(-16)
70 LSET B$ = "PAPER"
80 PUT #1, 2
90 CLOSE #1
```

The only differences between this program and Program 7 are in Lines 10, 20, 30, and 60. This is what the program stores in your disk file:

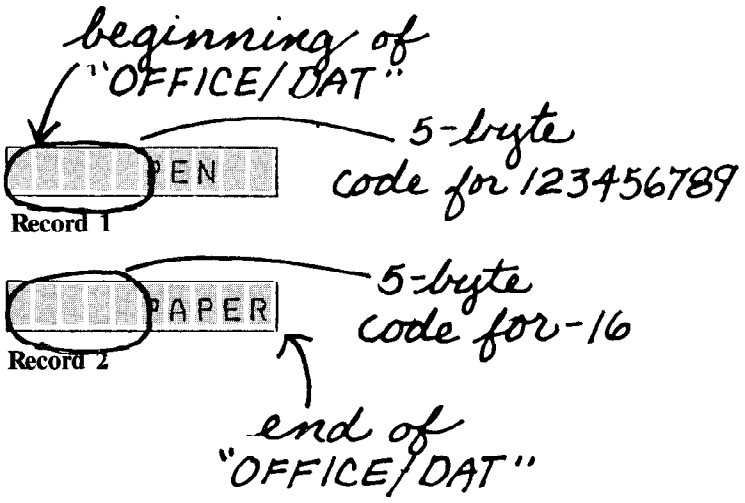



MKN$ converts a number to a coded string. Regardless of how long the number is, MKN$ always converts it to a string that is five bytes long.

For example, change Line 30 to LSET a number with more than five digits:

```
30 LSET A$ = MKN$(123456789)
```

Erase memory, and delete OFFICE/DAT. Type and run the program. This is what it stores in your disk file:




To read this program, you need to decode the string. Load the Fielded INPUT program, and make these changes to it:

```
10 OPEN "D", #1, "OFFICE/DAT", 10
20 FIELD #1, 5 AS A$, 5 AS B$
50 PRINT "RECORD" R ":"; CVN(A$); B$
```

Run it. CVN (in Line 50) decodes A$ to the number it represents.

> **Note:** The computer sees only the first nine digits of a number. It rounds off the rest.

### PROGRAMMING EXERCISE 9-3

*Write a fielded direct access program to store the populations of all countries. Make each record contain 15 bytes with these two fields:*

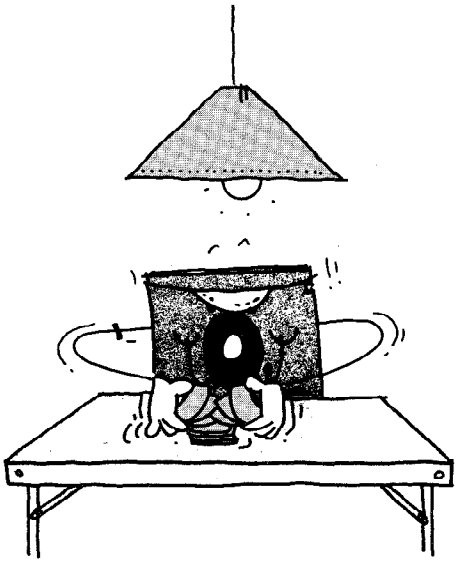
1. *country — 10 bytes*
2. *population — 5 bytes*

### PROGRAMMING EXERCISE 9-4

*Write a program to input the file you created in Exercise 9-3.*

### ☐ CHAPTER CHECKPOINT

1. *If you do not specify the record length, how many bytes does each record contain?*
2. *Why must you include a FIELD line when you use LSET on your data?*
3. *Into how many bytes does MKN$ convert a number?*



# 10

# Shuffling Disk Files

## (Merging Programs, Using File Buffers)

Because storing and retrieving disk files is so easy, you probably want to use files as much as you can. In this chapter, we talk about some special ways you can use them.

## Merging Program Files

With the first method, you can build a program out of related program *modules* saved on disk. You can then merge any of these modules with a program you have in memory.

Type and save these two related programs:

```
10 REM AGE CONVERSION TO MONTHS
20 N = N * 12
30 A$ = STR$(N) + " MONTHS"
SAVE "MONTHS/AGE", A (ENTER)

10 REM AGE CONVERSION TO WEEKS
20 N = N * 52
30 A$ = STR$(N) + " WEEKS"
SAVE "WEEKS/AGE", A (ENTER)
```

Be sure to type the **A** when you save these programs. (We explain why later.) Erase memory. Now, put the following program in memory:

```
5 INPUT "TYPE YOUR AGE"; N
40 PRINT "YOU HAVE LIVED" A$
```

Combine it with the MONTHS/AGE program you saved. Type:

```
MERGE "MONTHS/AGE" (ENTER)
```

List the program that is now in memory by typing **LIST** (ENTER). It now looks like this:

```
5 INPUT "TYPE YOUR AGE"; N
10 REM AGE CONVERSION TO MONTHS
20 N = N * 12
30 A$ = STR$(N) + " MONTHS"
40 PRINT "YOU HAVE LIVED" A$
```

Notice that the line numbers are the same as they were in the individual programs.

Now, merge WEEKS/AGE with the program in memory. To do this, type: **MERGE "WEEKS/AGE"** (ENTER). Then, list the merged program.

Notice that Lines 10, 20, and 30 of the program you had in memory are replaced by Lines 10, 20, and 30 of the WEEKS/AGE program.

The line numbers tell the computer how to merge the two programs. When there is a conflict of line numbers (for example, when there are two Line 10's), the line from the disk file prevails.

Now, we get technical (for those of you who are interested). What the computer normally writes in your disk file is the ASCII code for each character of data. For example, it writes the word AT with two codes: the ASCII code for A (65) and the ASCII code for T (84). (The ASCII codes are all listed in Appendix D.)

However, when it saves a program, the computer writes the BASIC words differently. To save space, it compresses each BASIC word into a 1- or 2-byte *binary* code, often referred as a *token*.

You can't merge a file that contains binary code. This is why we had you type the **A** when you saved the two previous programs. The **A** tells the computer to write the ASCII code for each BASIC word, rather than the binary code.

By checking the directory, you can see if the data in your files is in ASCII or binary code. If there is an **A** in the fourth column, all the data is in ASCII. A B indicates that some words are in binary.

> **Note:** Try typing **MERGE "MONTHS/AGE" , R** (ENTER). The **R** tells the computer to run the program after merging it.

# Using More Buffer Space

When you start up your disk system, it sets aside two buffers in memory for disk communication. You can use either or both for reading from or writing to a disk file.

Until now, those were all we used—Buffers #1 and #2. But, as we said earlier, you can use as many as 15 disk buffer areas.

To use more than two buffers, you must first reserve space in memory for the buffers. To do this, use the word FILES. For example, typing **FILES 3** (ENTER) reserves three buffers.

Making use of all these buffers greatly simplifies your programs. For example, assume you own a computer school. To organize it, you first put all your students in a file named COMPUTER/SCH.

Erase memory. Then, type and run:

```
10 OPEN "O", #1, "COMPUTER/SCH"
20 FOR X = 1 TO 6
30 READ A$
40 PRINT #1, A$
50 NEXT X
60 CLOSE #1
70 DATA JON, SCOTT, CAROLYN
80 DATA DONNA, BILL, BOB
```

Now, you can write a program to assign the students to either a BASIC or an assembly-language class. Erase memory, and type the following program:

**CLASS ASSIGNMENT PROGRAM**

```
10 FILES 3
20 OPEN "O", #1, "BASIC/CLS"
30 OPEN "O", #2, "ASSEMBLY/CLS"
40 OPEN "I", #3, "COMPUTER/SCH"
50 IF EOF(3) = -1 THEN 150
60 INPUT #3, ST$
70 CLS
80 PRINT "CLASS FOR "ST$ "?": PRINT
90 PRINT "(1) BASIC"
100 PRINT "(2) ASSEMBLY LANGUAGE"
110 PRINT: INPUT
    " ENTER 1 OR 2...";R
120 IF R>2 THEN 90
130 WRITE #R, ST$
140 GOTO 50
150 CLOSE #1
160 CLOSE #2
170 CLOSE #3
```

Run the program. After assigning all students to a class, you can use the following program to print a class roster. Erase memory. Then, type and run.

**CLASS ROSTER PROGRAM**

```
10 CLS
20 PRINT "BASIC/CLS" : PRINT
30 OPEN "I", #1, "BASIC/CLS"
40 IF EOF(1) = -1 THEN 80
50 INPUT #1, A$
60 PRINT A$
70 GOTO 40
80 CLOSE #1
```

**Note:** To print the roster of the assembly language class, substitute ASSEMBLY/CLS for BASIC/CLS in Lines 20 and 30.

The Class Assignment program has three buffers open at the same time. Because of this, you are able to communicate with three disk files at the same time.

Line 10 reserves memory for these three buffers. Lines 20-40 open the three buffers. Then, Line 60 inputs a student from COMPUTER/SCH into Buffer #3.

Line 100 writes the name of the student to either Buffer #1 (BASIC/CLS) or Buffer #2 (ASSEMBLY/CLS).

When all the students from Buffer #3 (COMPUTER/SCH) are input, Line 50 sends the computer to Lines 150-170, which close the three buffers.

# Crowding the Buffer

There is yet another use for FILES. Erase memory. Then, type and run this program:

```
10 CLEAR 400
20 FILES 1,400
30 A$ = "NORMALLY, YOU WILL NOT BE ABLE TO
   PUT ALL OF THESE SENTENCES IN A DISK
   FILE AT THE SAME TIME."
40 B$ = "THIS IS BECAUSE, WITHOUT USING
   FILES, YOU WILL ONLY HAVE A TOTAL OF
   256 BYTES OF BUFFER SPACE."
50 C$ = "IN THIS PROGRAM, WE'VE RESERVED
   400 BYTES OF BUFFER SPACE."
60 D$ = "THIS WAY YOU CAN SEND ALL OF THESE
   SENTENCES TO THE BUFFER AT THE SAME
   TIME."
70 E$ = "WHICH WILL OUTPUT THEM ALL TO THE
   DISK FILE AT ONCE."
80 OPEN "O", #1, "WORD/DAT"
90 WRITE #1, A$, B$, C$, D$, E$
100 CLOSE #1
```

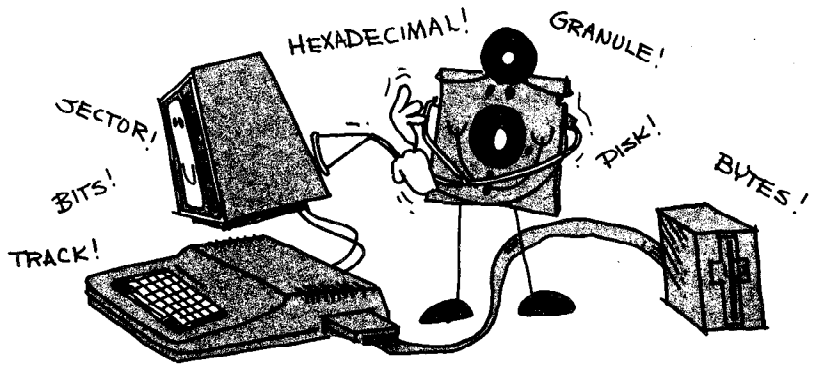
Want to input the above paragraph? Add the following lines, and run the program agian:

```
200 OPEN "I", #1, "WORD/DAT"
210 INPUT #1, A$, B$, C$, D$, E$
220 CLS
230 PRINT A$; B$; C$; D$; E$
240 CLOSE #1
```

**Note:** You can make the buffer as large as you want.

### ☐ CHAPTER CHECKPOINT

1. *How must you save a program that you want to merge?*
2. *When the two programs you're merging have a line with the same number, which line prevails?*
3. *How many buffers does the computer reserve when it starts up?*
4. *How much buffer space does it reserve?*
5. *What does* `FILES 3, 3000` *mean?*




# 11

# Technical Information

## (Disk Structure and Machine-Language)

In this chapter, we discuss the technical operations that go on behind the scenes. You don't need this information when you are programming in BASIC. In fact, you won't even be aware of the operations as they occur.

However, if you plan to write machine-language disk programs, or if you are simply interested in knowing all you can, you definitely want to read this chapter. It begins by describing how the computer organizes all the bytes on the disk. Then, it shows how to access them through machine-language programming and other advanced techniques.

# What a Disk Contains

When you power up the computer, it organizes the bytes on the disk into tracks and sectors. Some of the bytes control the system. Most are for data.

## Tracks

The computer organizes the disk into 35 tracks, numbered 0-34. Each track contains approximately 6,250 bytes. Of these, 6,084 are divided into sectors; the remaining bytes are for system controls.

| Bytes | Contents |
|---|---|
| 0-31 | System controls |
| 32-6115 | Sectors |
| 6116-6249 | System controls |

**Note:** The number of system control bytes at the end of each track might vary slightly because of slight speed variations.

The system control bytes all contain the hexadecimal value 4E.

**Note:** One byte contains eight bits. Each bit contains either the value 1 or the value 0. Normally, we express the contents of the bits as a hexadecimal (base 16) number. For example, if we say a byte contains the value of hexadecimal 4E, it contains this bit pattern: 01001110. You can find more information on hexadecimal and binary number systems in a math textbook.

# Sectors

Each track contains 18 sectors, numbered 1-18. Each sector contains 338 bytes. Of these, 256 are for data. The remaining bytes are for system controls.

| Bytes | Contents |
|---|---|
| 0-55 | System controls |
| 56-311 | Data |
| 312-337 | System controls |

The hexadecimal contents of the system control bytes are:

| Bytes | Hexadecimal Contents |
|---|---|
| 0-7 | 00 |
| 8-10 | F5 |
| 11 | FE |
| 12 | Track number |
| 13 | 00 |
| 14 | Sector number |
| 15 | 01 |
| 16-17 | Cyclic redundancy check (CRC) |
| 18-39 | 4E |
| 40-51 | 00 |
| 52-54 | F5 |
| 55 | FB |
| 312-313 | Cyclic redundancy check (CRC) |
| 314-337 | 4E |

# How the Data is Organized

Each track contains 4,608 bytes that the computer can use for data:

```
   18  sectors per track
x 256  data bytes per sector
-----------------------------
4,608  data bytes per track
```

The data bytes in the 17th track contain the disk's directory. The data bytes in the remaining 34 tracks are for disk files:

| Track | Contents of Track's Data Bytes |
|---|---|
| 0-16 | Disk files |
| 17 | Disk directory |
| 18-34 | Disk files |

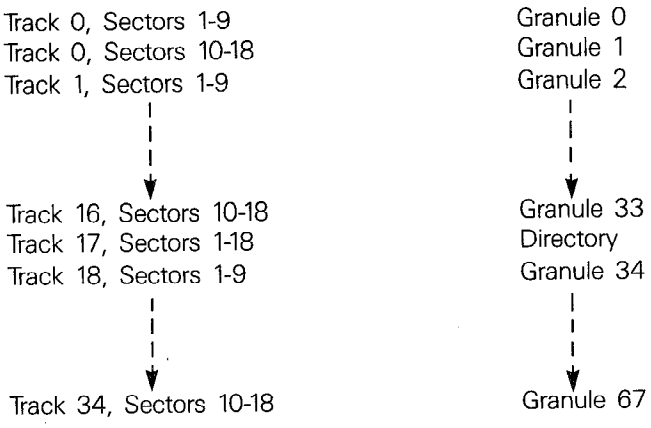
## Disk Files

The computer divides the 34 tracks for disk files into 68 granules. Since each track contains two granules, one granule is 2,304 bytes long:

```
   9  sectors in 1/2 track
x 256  data bytes per sector
-----------------------------
2,304  bytes in a granule
```

The computer uses granules to allocate space for disk files in 2,304-byte clusters. Thus, if a file contains 4,700 bytes, the computer allocates three granules (6,912 bytes) of disk space for it.

The location of the 68 granules, numbered 0-67, is as follows:




**Note:** The minimum size of a disk file is one granule, or 2,304 bytes. A disk holds a maximum of 68 disk files.

# Disk Directory

The directory track (Track 17) contains a file allocation table and directory entries. The sectors on this track that contain this information are:

| Sector | Contents |
|---|---|
| 2 | File allocation table |
| 3-11 | Directory entries |

The remaining sectors in the directory track are for future use.

## Directory Entries

The nine sectors of the directory that contain directory entries (Sectors 3-11) hold as many as 72 entries. Each entry is 32 bytes long and contains:

| Bytes | Contents |
|---|---|
| 0-7 | Filename, left-justified, blank-filled.<br>If Byte 0 = 0, the file no longer exists, and the entry is available.<br>If Byte 0 = FF (hexadecimal), the entry (and all following entries) are not being used. |
| 8-10 | Filename extension, left-justified, blank-filled. |
| 11 | File type<br>0 = BASIC program<br>1 = BASIC data file<br>2 = Machine-language program<br>3 = Text editor source file |
| 12 | ASCII flag<br>0 = the file is in binary format<br>FF (hexadecimal) = the file is in ASCII format |
| 13 | The number of the first granule in the file (0-67). |
| 14-15 | The number of bytes in use in the last sector of the file. |
| 16-31 | Reserved for future use. |

## File Allocation Table

Sector 2 of the directory contains a file allocation table for each of the 68 granules on the disk. This information is located in the first 68 bytes of the sector. The remaining bytes contain zeroes:

| Bytes | Contents |
|---|---|
| 0-67 | Granule information |
| 68-255 | Zeroes |

Each of the first 68 bytes corresponds with a granule. For example, Byte 15 corresponds with Granule 15.

These bytes contain a hexadecimal value of FF, 0-43, or C0-C9:

| Value | Meaning |
|---|---|
| FF | The corresponding granule is free. It is not part of a disk file. |
| 00-43 | The corresponding granule is part of a disk file. The value, converted to decimal, points to the next granule in the file. For example, if the value in a byte is 0A, the next granule in the file is Granule 10. |
| C0-C9 | The corresponding granule is the last granule in the file. The value contained in Bits 0-5 of this byte tells how many of the sectors in the granule are part of the disk file. (Bits 6 and 7 both equal 1.) |

# Skip Factor

The computer transfers data to and from the disk one sector at a time. Between each sector read or write, it does some processing.

The disk does not stop while the computer does processing. It spins continuously.

For example, the computer might read Sector 1 first. By the time it finishes this, the disk might have spun to Sector 6.

To allow for this time differential, the computer sets a *skip factor* of four when it formats the disk. This notes on the disk that the computer skips four *physical* sectors between each *logical* sector:

| PHYSICAL SECTOR | LOGICAL SECTOR |
|---|---|
| 1 | 1 |
| 2 | 12 |
| 3 | 5 |
| 4 | 16 |
| 5 | 9 |
| 6 | 2 |
| 7 | 13 |
| 8 | 6 |
| 9 | 17 |
| 10 | 10 |
| 11 | 3 |
| 12 | 14 |
| 13 | 7 |
| 14 | 18 |
| 15 | 11 |
| 16 | 4 |
| 17 | 15 |
| 18 | 8 |

Thus, after reading Sector 1, the computer skips Physical Sectors 2, 3, 4, and 5. The second sector it reads is Physical Sector 6 (or logical Sector 2).

A skip factor of 4 is the optimum setting for BASIC loads and saves. However, if you're not using BASIC, you might be able to use a faster skip factor. For example:

```
DSKINIØ, 3 (ENTER)
```

tells the computer to skip three physical sectors between each logical sector.

> **Note:** Determining the optimum skip factor is difficult. We recommend you leave the factor at 4 unless you have a good understanding of how it works.

# Machine-Language Disk Programming

The disk system contains a machine-language routine called DSKCON, which you can call for all disk input/output operations. To call this routine, you write instructions to the Color Computer's 6809 microprocessor.

For the procedures to use in accessing a machine-language subroutine, see "Using Machine-Languages Subroutines" in the BASIC manual that came with your computer. See *TRS-80 Color Computer Assembly Language Programming,* by William Bardon Jr. for the specific 6809 instructions.

# Information on DSKCON

A pointer to DSKCON's entry address is stored in locations C004 and C005 (hexadecimal). You can call it with this assembly-language instruction:

```
JSR [$CØØ4]
```

DSKCON's parameters are located in six memory locations, organized as follows:

| | | |
|---|---|---|
| DCOPC | RMB | 1 |
| DCDRV | RMB | 1 |
| DCTRK | RMB | 1 |
| DSEC | RMB | 1 |
| DCBPT | RMB | 2 |
| DCSTA | RMB | 1 |

The pointer to the address of the first, DCOPC, is contained in locations C006 and C007 (hexadecimal). You can use the first five memory locations to pass parameters to DSKCON. DSKCON returns a status byte to the sixth location, DCSTA.

These are the parameters you can pass to the first five memory locations:

DCOPC — *operation code*
- 0 = Restore head to Track 0
- 1 = No operation
- 2 = Read sector
- 3 = Write sector

DCDRV — *drive number* (0-3)

DCTRK — *track number* (0-34)

DCSEC — *sector number* (1-18)

DCBPT — *buffer pointer* (the address of a 256-byte buffer)
For a "read sector" operation, the data is returned in the buffer. For a "write sector" operation, the data in the buffer is written on the disk.

This is the meaning of the status byte that the DSKCON routine returns to location DCSTA:

DCSTA — *status*

A value of 1 in Bit 7 = Drive not ready
A value of 1 in Bit 6 = Write protect
A value of 1 in Bit 5 = Write fault
A value of 1 in Bit 4 = Seek error or record not found
A value of 1 in Bit 3 = CRC error
A value of 1 in Bit 2 = Lost data

If all the bits contain 0, no error occurred. (See the disk service manual for further details on the error bits.)

After returning from DSKCON, you can turn off the drive motor by putting the value of 0 in the memory location FF40 (hex).

## Sample Programs Using DSKCON

The following program uses DSKCON to restore the head to Track 0:

```
LDX  $C006    SET X AS A POINTER TO THE
              PARAMETERS
CLR  ,X       DCOPC =0 FOR RESTORE
LDA  #1       DCDRV =1 TO SELECT DRIVE
              ONE
STA  1,X
JSR  [$C004] CALL DSKCON
LDA  #$00     TURN OFF THE DRIVE MOTOR
STA  $FF40
TST  6,X      CHECK FOR ERRORS
BNE  ERRORS   GO REPORT THE ERRORS
RTS
LDA  #$45     "E" FOR ERROR
STA  $41D     TOP RIGHT OF THE DISPLAY
RTS
```

This program uses DSKCON to read Track 3, Sector 17 of Drive 0 into memory Locations 3800-38FF:

```
LDX  $C006    SET X AS A POINTER TO THE
              PARAMETERS
LDA  #2       DCOPC =2 FOR READ A SECTOR
STA  ,X
CLR  1,X      SELECT DRIVE 0
LDA  #3       SELECT TRACK 3
STA  2,X
LDA  #17      SELECT SECTOR 17
STA  3,X
LDU  #$3800   DCBPT=3800(HEX) FOR
              STORING DATA
STU  4,X
JSR  [$C004]  CALL DSKCON
LDA  #$00     TURN OFF THE DRIVE MOTOR
STA  $FF40
TST  6,X      CHECK FOR ERRORS
BNE  ERRORS   GO REPORT THE ERRORS
RTS
LDA  #$45     "E" FOR ERROR
STA  $41D     TOP RIGHT OF THE DIPSLAY
RTS
```

**Note:** DSKCON preserves the contents of all registers except CC.

You can write a similar program to write to a sector by setting DCOPC to 3 instead of 2.

## Saving a Machine-Language Program

You can use the SAVEM command to store a machine-language program on disk. You need to specify where in memory the program resides (the starting and ending addresses of the program.) You also need to specify the address at which the program should be executed. Use the hexadecimal numbers for all these addresses.

For example, assume you have a machine-language program that resides in Addresses 5000-5FFF of memory. The address at which it should be executed is 500A. You store this program on disk by typing:

```
SAVEM "PROG/MAC", &H5000, &H5FFF, &H500A
ENTER
```

To load the program back into memory, use the LOADM command:

```
LOADM "PROG/MAC" ENTER
```

The preceding command loads PROG/MAC back into Locations 5000-5FFF. The computer begins executing the program at Location 500A.

If you want to load the program into a different memory location, specify an offset address to add to the program's loading address. For example, typing:

```
LOADM "PROG/MAC", &H1000
```

loads PROG/MAC into Locations 6000-6FFF. The computer begins executing the program at Address 600A.

# Special Input/Output Commands

BASIC offers two special input/output commands. These commands transfer data directly to and from a particular sector. To do this, they bypass the entire disk's filing system.

The first command, DSKI$, inputs the data from the sector you specify. This is its format:

DSKI$ *drive number, track, sector, string variable1, string variable2*

The first 128 bytes of the sector are input into *string variable1*. The second 128 bytes are input into *string variable2*. For example:

```
CLEAR 26Ø (ENTER)
DSKI$ Ø, 17, 1, A$, B$ (ENTER)
```

inputs the contents of Sector 1, Track 17 of the disk in Drive 0. It inputs the first 128 bytes into A$ and the second 128 bytes into B$. After typing this command, you can display the contents of this sector by typing:

```
PRINT A$; B$ (ENTER)
```

Since DSKI$ reads any sector on the disk, it is the only BASIC command that can read the directory sector. The following sample program uses DSKI$ to search the directory for filenames with the extension DAT:

```
 5  CLEAR 1ØØØ
1Ø  FOR X=3 TO 11
2Ø  DSKI$ Ø,17,X,A$,B$
3Ø  C$ = A$ + LEFT$(B$,127)
4Ø  NAM$(Ø) = LEFT$(C$,8)
5Ø  EXT$(Ø) = MID$(C$,9,3)
6Ø  FOR N=1 TO 7
7Ø  NAM$(N) = MID$(C$,N*32+1,8)
8Ø  EXT$(N) = MID$(C$,9+N*32,3)
9Ø  NEXT N
1ØØ FOR N=Ø TO 7
11Ø IF EXT$(N) = "DAT" AND
    LEFT$(NAM$(N),1)<>CHR$(Ø) THEN PRINT
    NAM$(N)
12Ø NEXT N
13Ø NEXT X
```

The second command, DSKO$, outputs data directly to the sector you specify. Since it bypasses the disk filing system, it outputs data without opening a file and listing its location in the directory. For this reason, you need to be careful:

- Do not write data on the directory sectors unless you no longer plan to use the directory.
- Do not write data on other sectors that contain data you wish to keep.

The format of DSKO$ is:

DSKO$ *drive number, track, sector, string1, string2*

*String1* goes in the first 128 bytes of the sector. *String2* goes in the next 128 bytes. For example:

```
DSKO$ 0, 1, 3, "FIRST STRING", "SECOND
STRING" ENTER
```

outputs data to Sector 3, Track 1, on the disk in Drive 0. The string FIRST STRING goes in the first 128 bytes of the sector. The string SECOND STRING goes in the second 128 bytes.

When you use **DSKO$** to write data on a diskette, neither the directory nor the file allocation table is updated to reflect the entry. If you write to a sector that is not currently allocated, later file operations might overwrite the data. You can update the directory and file allocation table using DSKO$, but such a process is difficult and might make your diskette unusable. If you wish to attempt writing to the directory and/or the file allocation table, always do so using a backup diskette. This way, if you make an error, you do not destroy important data. See "File Allocation Table," earlier in this chapter, for information on the directory and file allocation table.

# Part 4

# APPENDICES

Appendix A

# Programming Exercise Answers

## Programming Exercise 5-1

```
5 CLS
10 PRINT: PRINT "CHECKS FOR CAR EXPENSES"
20 OPEN "I", #1, "CHECKS"
30 IF EOF(1) = -1 THEN 100
40 INPUT #1, A$, B, C$
50 IF C$ = "CAR" THEN 70
60 GOTO 90
70 PRINT: PRINT "CHECK PAYABLE TO:";A$
80 PRINT "AMOUNT:";B
90 GOTO 30
100 CLOSE #1
```

## Programming Exercise 6-1

```
10 OPEN "I", #1, "ANIMALS/DAT"
20 OPEN "O", #2, "NEW/DAT"
30 IF EOF(1) = -1 THEN 70
40 INPUT #1, A$
50 WRITE #2, A$
60 GOTO 30
70 CLOSE #1
80 CLS
90 PRINT "PRESS <ENTER> WHEN FINISHED"
100 PRINT: PRINT "NEW ANIMAL... ";
110 INPUT A$
120 IF A$ = "" THEN 150
130 WRITE #2, A$
140 GOTO 80
150 CLOSE #2
160 KILL "ANIMALS/DAT
170 RENAME "NEW/DAT" TO "ANIMALS/DAT"
```

## Programming Exercise 6-2

```
10 OPEN "I", #1, "ANIMALS/DAT"
20 OPEN "O", #2, "NEW/DAT"
30 IF EOF(1) = -1 THEN 100
35 CLS
40 INPUT #1, A$
50 PRINT: PRINT: PRINT "THIS ANIMAL IS A ";A$
60 INPUT "DELETE IT (YES/NO)..."; R$
70 IF R$= "YES" THEN 90
80 WRITE #2, A$
90 GOTO 30
100 CLOSE #1
110 CLOSE #2
120 KILL "ANIMALS/DAT"
130 RENAME "NEW/DAT" TO "ANIMALS/DAT"
```

## Programming Exercise 6-3

```
10 CLS: PRINT "     -- DO YOU WISH TO --":PRINT
20 PRINT "     (1) STORE A NEW FILE"
30 PRINT "     (2) SEE THE FILE"
40 PRINT "     (3) END":PRINT
50 INPUT "ENTER 1 2 OR 3... ": Q1
60 ON Q1 GOTO 80, 130, 420
70 GOTO 10
80 OPEN "O", #1, "MEMBERS/DAT"
90 GOSUB 430
100 IF N$="" THEN CLOSE #1: GOTO 10
110 WRITE #1, N$, A$, P$
120 GOTO 90
130 OPEN "I", #1, "MEMBERS/DAT"
140 OPEN "O", #2, "TEMP/DAT"
150 CLS
160 IF EOF(1) = -1 THEN 320
170 INPUT #1, N$, A$, P$
180 PRINT: PRINT "NAME :" N$
190 PRINT "ADDRESS :"A$
200 PRINT "TELEPHONE:"P$
```

```
205 PRINT: INPUT "CHANGE THIS FILE (YES/NO)...";
    Q2$
210 IF Q2$ = "NO" THEN 300
220 PRINT: PRINT "DO YOU WISH TO:"
230 PRINT "1) CHANGE THE ADDRESS?"
240 PRINT "2) DELETE THE MEMBER?"
250 PRINT "3) GO ON TO THE NEXT MEMBER?"
260 INPUT N
270 ON N GOTO 290, 160, 300
280 GOTO 230
290 INPUT "INPUT NEW ADDRESS"; A$
300 WRITE #2, N$, A$, P$
306 CLS
310 GOTO 160
320 PRINT: INPUT "ADD NEW MEMBER (YES/NO)...";
    Q3$
330 IF Q3$="NO" THEN 380
340 GOSUB 430
350 IF N$ = "" THEN 380
360 WRITE #2, N$, A$, PS$
370 GOTO 340
380 CLOSE #1, #2
390 KILL "MEMBERS/DAT"
400 RENAME "TEMP/DAT" TO "MEMBERS/DAT"
410 GOTO 10
420 END
430 CLS: PRINT "PRESS <ENTER> WHEN FINISHED" :
    PRINT
440 INPUT "NAME OF MEMBER:";N$
450 IF N$="" THEN 480
460 INPUT "ADDRESS : ";A$
470 INPUT "PHONE NUMBER :"; P$
480 RETURN
```

**Programming Exercise 7-1**

```
10 OPEN "D", #1, "NAMES/DAT"
20 WRITE #1, "J. DOE"
30 PUT #1, 2
```

```
31 WRITE #1, "BILL SMITH"
32 PUT #1, 3
34 GET #1, 3
36 INPUT #1, A$
38 PRINT A$
40 CLOSE #1
```

## Programming Exercise 7-2

Using the proposed line produces an FD Bad File Data error in Line 36. The first field in Record 2 is FLIES, a string. Line 36 inputs it into N, a numeric variable.

## Programming Exercise 7-3

```
10 OPEN "D", #1, "NAMES/DAT"
20 GOTO 70
30 FOR X = 1 TO 10
40 PRINT: PRINT "RECORD "X
50 GOSUB 220
60 NEXT X
70 INPUT "WHICH RECORD(1-10)"; X
80 IF X > 10 THEN 70
90 IF X < 1 THEN END
100 GET #1, X
110 INPUT #1, N$, A$, C$, S$, Z$
120 PRINT: PRINT "RECORD" X
130 PRINT N$
140 PRINT "     "; A$
150 PRINT "     "; C$
160 PRINT "     "; S$
170 PRINT "     "; Z$
180 INPUT "DO YOU WANT TO CHANGE THIS"; R$
190 IF R$ = "YES" THEN GOSUB 220
200 GOTO 70
210 CLOSE #1: END
220 INPUT "NAME :"; NS
230 INPUT "ADDRESS :"; A$
240 INPUT "CITY :"; C$
250 INPUT "STATE :"; S$
```

```
260 INPUT "ZIP :"; Z$
270 WRITE #1, N$, A$, C$, S$, Z$
280 PUT #1, X
290 RETURN
```

### Programming Exercise 9-1

```
10 OPEN "D", #1, "MAIL/DAT", 57
20 FIELD #1, 15 AS LAST$, 10 AS FIRST$, 15 AS
   ADDRESS$, 10 AS CITY$, 2 AS STATE$, 5 AS ZIP$
30 R = R + 1
40 CLS
50 INPUT "LAST NAME"; L$
60 INPUT "FIRST NAME"; F$
70 INPUT "ADDRESS"; A$
80 INPUT "CITY"; C$
90 INPUT "STATE"; S$
100 INPUT "ZIP CODE"; Z$
110 LSET LAST$ = L$
120 LSET FIRST$ = F$
130 LSET ADDRESS$ = A$
140 LSET CITY$ = C$
150 LSET STATE$ = S$
160 LSET ZIP$ = Z$
170 PUT #1, R
180 PRINT
190 INPUT "MORE DATA(Y/N)"; AN$
200 IF AN$ = "Y" THEN 30
210 CLOSE #1
```

### Programming Exercise 9-2

```
10 OPEN "D", #1, "MAIL/DAT", 57
20 FIELD #1, 15 AS LAST$, 10 AS FIRST$, 15 AS
   ADDRESS$, 10 AS CITY$, 2 AS STATE$, 5 AS ZIP$
30 R = R + 1
40 CLS
50 GET #1, R
60 PRINT LAST$ "," FIRST$
```

```
70 PRINT ADDRESS$
80 PRINT CITY$ "," STATE$
90 PRINT ZIP$
100 PRINT
110 IF LOF(1)=R THEN 140
120 INPUT "PRESS <ENTER> FOR NEXT NAME"; E$
130 GOTO 30
140 CLOSE #1
```

## Programming Exercise 9-3

```
10 OPEN "D", #1, "POP", 15
20 FIELD #1, 10 AS COUNTRY$, 5 AS POP$
30 R = R + 1
40 CLS
50 INPUT "COUNTRY"; C$
60 INPUT "POPULATION"; P
70 LSET COUNTRY$ = C$
80 LSET POP$ = MKN$(P)
85 PUT #1, R
90 PRINT
100 INPUT "MORE DATA(Y/N)"; AN$
110 IF AN$ = "Y" THEN 30
120 CLOSE #1
```

## Programming Exercise 9-4

```
10 OPEN "D", #1, "POP", 15
20 FIELD #1, 10 AS COUNTRY$, 5 AS POP$
30 R = R + 1
40 GET #1, R
50 PRINT COUNTRY$, CVN (POP$)
60 IF LOF(1)<>R THEN 30
70 CLOSE #1
```

## Appendix B

# Chapter Checkpoint Answers

### Chapter 2

1. Unless the disk is formatted, there is no way to locate any given area on the disk.

2. The disk directory is an index of the names, locations, and types of all the files on the disk.

3. A disk file is a defined block of information stored on the disk, under a unique filename.

4. Information stored in memory is only there temporarily. It is destroyed when the computer is turned off or if you execute a NEW, LOAD, DISKINI, BACKUP, or COPY command. (We discuss BACKUP and COPY in the next appendices.) Information stored on disk will be there permanently. It isn't destroyed when you turn off the computer or clear memory. Don't leave a disk in the drive when you turn the computer off. (We explain why in Chapter 3.)

5. The only way to change the contents of a disk file is to store different information under the same filename.

### Chapter 3

1. Turning the computer on or off while the disk is in its drive might damage information on the disk.

2. Use only felt tip pens to write on the disk's label. Hard point pens and pencils can damage the disk and garble the information on it.

3. Error messages tell you that something is wrong with either the program you are running or the last command that you used.

4. *Write protecting* is a way of protecting your disks from alteration. It is done by putting a gummed label over the write-protect notch. You can read from a write-protected disk, but you can't write to it.

5. On a one-drive system, insert the source disk into the drive and type **BACKUP0**(ENTER). The computer asks you to insert the destination disk and press(ENTER). This procedure is repeated until the computer displays **OK**. On a multi-drive system, type the BACKUP command, specifying the drive numbers of the source disk and the destination disk. For example, **BACKUP 0 TO 1** backs up the source disk (in Drive 0) to the destination disk (in Drive 1).

## Chapter 4

1. You can rename a file by using the RENAME command. For example, **RENAME OLDFILE/NAM TO NEWFILE/NAM** renames OLDFILE/NAM to NEWFILE/NAM. You must specify the extension with both filenames so that the computer can find the files.

2. You can find out how much space you have remaining on the disk by typing **PRINT FREE(0)**(ENTER). This tells you the number of granules left on the disk in Drive 0. If you are running out of granules, you might want to delete a few files or switch to another disk.

3. Unless you specify otherwise, the computer uses Drive 0. You can change the "assumed drive" by typing **DRIVE 1**(ENTER), which enables you to access Drive 1 without having to specify the number in your command. (Now, you can use either **DIR** or **DIR1** to display the directory of the disk in Drive 1.)

## Chapter 5

1. Buffer #1 is a temporary storage area for information going between the disk and memory.

2. A disk file must be opened before any information can go between the disk and memory.

3. A disk file must be closed so that the information still in the buffer ends up where it is supposed to. Also, if you don't close a file, you can't reopen it. All files must be closed before you switch disks.

4. A file opened for input allows information to go from the disk file into the memory of the computer. A file opened for output allows information to go from memory to the disk file.

## Chapter 6

1. You can only open a sequential access file for input or output—not both. You can't output to a file opened for input; nor can you input from a file opened for output.

2. No. You must close the file and then reopen it for input.

## Chapter 7

1. Records are equal-sized divisions of your disk file in which you can put your data. Since each record is the same size, the computer can use records to access your data directly.

2. Fields are divisions of records.

3. In a sequential access file, the only locations the computer knows are the beginning and ending of the file. In a direct access file, the computer can determine the location of each record.

4. Since each record of a direct access file has a known location, the computer can access a record without going through the preceding parts of the file, as it must do with sequential access files.

## Chapter 8

1. The minimum size of a disk file is 2,304 bytes (one granule). It can be no smaller because the computer allocates disk space in granules.

2. The computer first writes the number's sign (a minus sign if the number is negative or a blank space if it is positive). Then, the computer writes the number itself. Immediately following the number, it writes one trailing blank space.

3. When you use WRITE, the computer puts quotation marks around strings.

4. INPUT inputs only the data items listed. LINE INPUT inputs everything up to the (ENTER) character.

5. A comma causes the computer to space over to the next print column before printing another data item.

6. A semicolon causes the computer to print the data items immediately next to each other.

7. When you use PRINT, the computer prints strings without enclosing them in quotation marks.

## Chapter 9

1. The computer sets the record length to 256 bytes.

2. The data must be in a field of a specified length so that the computer can left-justify it in that field. This length is assigned in the FIELD line.

3. MKN$ converts a number to a string that is five bytes long.

## Chapter 10

1. Type an A at the end of your SAVE command if you plan to merge the program with a program in memory.

2. The line in the program saved on disk prevails.

3. The computer reserves two buffers when you power up.

4. The computer reserves 256 bytes of buffer space when you power up.

5. **FILES 3, 3000** gets the computer to reserve three buffers with a total of 3000 bytes of buffer space.

# Sample Programs

## Sample Program #1 / Balancing Your Checkbook

This program creates a master disk file that contains all your checks and deposits for the entire year. You can print out the transactions by the month or the year. If you want to use your printer, change the appropriate PRINT lines to PRINT #-2.

```
10 ' Checkbook Program
20 '
30 ' This program provides a record of your
40 ' checks, deposits, and balances. The checks
50 ' can be labeled with an account number to show
60 ' to what expense they paid, such as medical,
70 ' rent, food. The program uses direct
80 ' addressing; each file is 40 bytes long and
90 ' formatted as follows: 8 bytes for data,
100 ' 4 bytes for check or deposit slip number,
110 ' 20 bytes for payee, 3 bytes for account
120 ' number, and 5 bytes for amount. If your
130 ' computer has enough memory, you can change
140 ' the value of the statement in Line 150 to
150 ' allow for more than 50 checks.
160 CLEAR 1000
170 DIM CHK$(50)
180 CLS
190 PRINT @ 107, "SELECTION:"
200 PRINT @ 162,"1) ADD CHECKS TO YOUR FILE"
210 PRINT @ 194,"2) LIST YOUR CHECKS, DEPOSITS."
220 PRINT @ 229,"AND BALANCES
230 PRINT @ 258,"3) END JOB"
240 PRINT @ 392, "(PRESS 1 2 OR 3)"
250 AN$=INKEY$
260 IF AN$="" THEN 250
270 ON VAL(AN$) GOSUB 300,670,1040
280 GOTO 180
```

```
290 '
300 '     This subroutine inputs the data
310 '
320 OPEN "D", #1, "CHECKS/DAT", 40
330 FIELD #1, 8 AS DATE$, 4 AS CHNO$, 20 AS PDTO$,
    3 AS ACNO$, 5 AS AMT$
340 REC = LOF(1)
350 REC = REC + 1
360 CLS
370 PRINT @ 64, "CHECK OR DEPOSIT(C/D)"
380 AN$ = INKEY$
390 IF AN$ = "D" THEN 420
400 IF AN$ = "C" THEN 480
410 GOTO 380
420 INPUT "DEPOSIT DATE (MM/DD/YY)"; D$
430 INPUT "DEPOSIT SLIP NUMBER (NNNN)"; C$
440 P$ = ""
450 INPUT "ACCOUNT NUMBER (NNN)"; A$
460 INPUT "AMOUNT OF DEPOSIT"; AMT
470 GOTO 540
480 INPUT "CHECK DATE (MM/DD/YY)"; D$
490 INPUT "CHECK NUMBER (NNNN)"; C$
500 INPUT "PAID TO"; P$
510 INPUT "ACCOUNT NUMBER (NNN)"; A$
520 INPUT "AMOUNT OF CHECK"; AMT
530 AMT = -AMT
540 LSET DATE$ = D$
550 LSET CHNO$ = C$
560 LSET PDTO$ = P$
570 LSET ACNO$ = A$
580 LSET AMT$ = MKN$(AMT)
590 PUT #1, REC
600 PRINT @ 320, "MORE INPUT(Y/N)"
610 AN$ = INKEY$
620 IF AN$ = "N" THEN 650
630 IF AN$ = "Y" THEN 350
640 GOTO 610
650 CLOSE #1
660 RETURN
```

```
670 ' This subroutine balances the checkbook and
680 ' outputs the results.
690 OPEN "D", #1, "CHECKS/DAT", 40
700 FIELD #1, 8 AS DATE$, 4 AS CHNO$, 20 AS PDTO$,
    3 AS ACNO$, 5 AS AMT$
710 CLS
720 PRINT @ 160, "DO YOU WANT A LISTING FOR A"
730 PRINT @ 192, "MONTH OR FOR THE WHOLE YEAR?"
740 PRINT @ 224, "PRESS <Y> OR <M> ";
750 A$=INKEY$
760 IF A$="Y" OR A$="M" THEN 770 ELSE 750
770 IF A$ = "M" THEN PRINT @ 224, "WHAT
    MONTH(MM) ";:INPUT MN$
780 BAL = 0
790 FOR REC = 1 TO LOF(1)
800 GET #1, REC
810 BAL = BAL + CVN(AMT$)
820 IF A$ = "M" AND LEFT$(DATE$,2) <> MN$ THEN
    1010
830 CLS
840 IF PDTO$ = "THEN 920
850 PRINT @ 64, "DATE OF CHECK:":PRINT @ 84, DATE$
860 PRINT "CHECK NUMBER:":PRINT @ 116,CHNO$
870 PRINT "PAID TO:":PRINT @ 148,PDTO$
880 PRINT @ 160, "ACCOUNT NUMBER:":PRINT @ 180,
    ACNO$
890 PRINT "AMOUNT OF CHECK:":PRINT @ 211, USING
    "$$###.##";-CVN(AMT$)
900 PRINT "BALANCE:":PRINT @243, USING
    "$$###.##"; BAL
910 GOTO 960
920 PRINT: PRINT: PRINT "DATE OF DEPOSIT:": PRINT
    @ 85,DATE$
930 PRINT "DEPOSIT SLIP NUMBER:":PRINT @
    117,CHNO$
940 PRINT "ACCOUNT NUMBER:":PRINT @ 149, ACNO$
950 PRINT "AMOUNT OF DEPOSIT:": PRINT @ 180,
    USING "$$###.##": BAL
```

```
960 PRINT @ 288, "PRESS <ENTER> FOR NEXT RECORD OR
    <R> TO RETURN TO 'SELECTIONS'"
970 AN$ = INKEY$
980 IF AN$ = CHR$(13) THEN 1010
990 IF AN$ = "R" THEN 1020
1000 GOTO 970
1010 NEXT REC
1020 CLOSE #1
1030 RETURN
1040 ' This subroutine terminates the program
1050 CLOSE
1060 END
```

## Sample Program #2/Sorting Your Checks

This subroutine can be especially helpful at tax time. It takes the checks file you created in Sample Program #1, and sorts the checks by account number. Want to know exactly how much you spent on medical bills (or business expenses, contributions, and so on)? This program lets you know right away. Change or add to Sample Program #1:

```
230 PRINT @ 258, "3) SORT YOUR CHECKS BY"
232 PRINT @ 293, "ACCOUNT NUMBER"
234 PRINT @ 322, "4) END JOB"
240 PRINT @ 392, "(PRESS 1 2 3 OR 4)"
270 ON VAL(AN$) GOSUB 310, 690, 1080, 1560
1060 '
1070 '
1080 ' This subroutine sorts the checks, from
1090 ' smallest account number to largest
1100 ' account number, using a bubble sort.
1110 ' Each check is handled as one data string.
1120 '
1130 OPEN "D", #1, "CHECKS/DAT", 40
1140 FIELD #1, 40 AS INFO$
1150 FOR I = 1 TO LOF(1)
1160 GET #1, I
1170 CHK$(I) = INFO$
```

```
1180 NEXT I
1190 CNT = 0
1200 FOR I = 1 TO LOF(1) -1
1210 IF MID$(CHK$(I),33,3) <=
     MID$(CHK$(I+1),33,3) THEN 1260
1220 TEMP$ = CHK$(I)
1230 CHK$(I)=CHK$(I+1)
1240 CHK$(I+1) = TEMP$
1250 CNT = 1
1260 NEXT I
1270 IF CNT = 1 THEN 1190
1280 CLS
1290 PRINT @ 194, "WHAT ACCOUNT NUMBER(NNN/ALL)"
1300 INPUT AN$
1310 FOR I = 1 TO LOF(1)
1320 IF AN$ <> "ALL" AND AN$ <>
     MID$(CHK$(I),33,3) THEN 1510
1330 CLS
1340 PRINT @ 66, "ACCOUNT NUMBER:":PRINT @
     85,MID$(CHK$(I),33,3)"
1350 IF MID$(CHK$(I),13,20)= " "THEN 1410
1360 PRINT @ 98, "DATE OF CHECK:"PRINT @ 117,
     LEFT$(CHK$(I),8)
1380 PRINT @ 162, "PAID TO:":PRINT @ 181,
     MID$(CHK$(I),13,20)
1390 PRINT @ 194, "AMOUNT OF CHECK:":PRINT @ 212,
     USING "$$###.##"; -CVN(RIGHT$(CHK$(I),5)
1400 GOTO 1440
1410 PRINT @ 98, "DATE OF DEPOSIT:"PRINT @
     117,LEFT$(CHK$(I),8)
1420 PRINT @ 130, "DEPOSIT NUMBER:":PRINT @
     149,MID$(CHK$(I),9,4)
1430 PRINT @ 162, "AMOUNT OF DEPOSIT:":PRINT @
     180, USING "$$###.##";
     CVN(RIGHT$(CHK$(I),5)
1440 PRINT @ 290, "(PRESS <ENTER> TO SEE NEXT"
1450 PRINT @ 322, "RECORD OR <R> TO RETURN TO"
1460 PRINT @ 354, "'SELECTIONS')"
1470 A2$ = INKEY$
```

```
1480 IF A2$ = "R" THEN 1520
1490 IF A2$ = CH$(13) THEN 1510
1500 GOTO 1470
1510 NEXT I
1520 CLOSE #1
1530 RETURN
1540 '
1550 ' terminate program
1560 '
1570 END
```

## Sample Program #3 / Membership List

Want to store the names and telephone numbers of all your club members? This program puts them all in a disk file in alphabetical order. Add a few lines to it, and it can store their addresses and phone numbers also.

```
10 ' Create list and alphabetize.
20 '
30 ' The object of this program is to create a file
40 ' of alphabetically arranged names and phone
50 ' numbers. The names and numbers are first
60 ' input into an array, ARRAY$(I), then put into
70 ' alphabetical order, and finally put into a
80 ' disk file called NAMES/NOS. The file is 35
90 ' bytes long, all of it allotted to one
100 ' variable, INFO$. When you add to it,
110 ' it will automatically be alphabetized.
120 ' You can use it with the Search a
130 ' List program (Sample Program #4).
140 CLEAR 1050
150 DIM ARRAY$(30)
160 OPEN "D", #1, "NAMES/NOS", 35
170 FIELD #1, 35 AS INFO$
180 '
190 ' First the file is checked to see if there are
200 ' any records currently in it.
210 '
```

```
220 IF LOF(1) = 0 THEN I=1:GOTO 310
230 FOR I=1 TO LOF(I)
240 GET #1, I
250 ARRAY$(I) = INFO$
260 NEXT I
270 '
280 ' The new names and numbers are input and then
290 ' concatenated into 1 string, ARRAY$(I)
300 '
310 CLS
320 PRINT @ 64
330 INPUT "LAST NAME";L$
340 INPUT "FIRST NAME";F$
350 INPUT "MIDDLE INITIAL";M$
360 INPUT "AREA CODE";A$
370 INPUT "PHONE NUMBER";P$
380 ARRAY$(I) = LEFT$(L$+","+F$+" "+M$+".",
    24)+A$+P$"
390 PRINT @ 288,"MORE DATA (Y/N)?"
400 AN$ = INKEY$
410 IF AN$ = "Y" THEN I=I+1 :GOTO 310
420 IF AN$ = "N" THEN 470
430 GOTO 400
440 '
450 ' ARRAY$(I) is put in alphabetical order.
460 '
470 FOR J=1 TO I
480 FOR K=J TO I
490 IF ARRAY$(J) < ARRAY$(K) THEN 530
500 TEMP$ = ARRAY$(J)
510 ARRAY$(J) =ARRAY$(K)
520 ARRAY$(K) = TEMP$
530 NEXT K
540 NEXT J
550 '
560 ' The list is transferred into NAMES/NOS.
570 '
580 FOR N=1 TO I
590 LSET INFO$ = ARRAY$(N)
```

```
600 PUT #1,N
610 NEXT N
620 CLOSE #1
630 END
```

## Sample Program #4/Search for a Name

Since the file you created in Sample Program #3 is already in alphabetical order, you can immediately find the name you want. This program shows how.

```
10 ' Search a list
20 '
30 ' (NOTE: This program requires that a file
40 ' called NAMES/NOS exist. See Sample
50 ' Program #3.) This program searches that
60 ' file. It is a direct access file called
70 ' NAMES/NOS, is 35 bytes long and is formatted
80 ' as follows: 24 bytes for name, 3 bytes
90 ' for area code, 8 bytes for phone number.
100 ' It uses interactive searching.
110 '
120 OPEN "D", #1, "NAMES/NOS",35
130 FIELD #1, 24 AS NAMES,3 AS AREA$,8 AS PHONE$
140 CLS
150 PRINT @ 99, "ENTER NAME IN THIS FORM:"
155 PRINT " JENKINS,FRED A."
160 PRINT : LINE INPUT NM$
170 '
180 ' Initialization of variables
190 '
200 N1$ = NM$
210 IF LEN(NM$) < 24 THEN 800
220 IF LEN(NM$) > 24 THEN 820
230 FIRST = 1
240 MID = INT((LOF(1)+1)/2)
250 LAST = LOF(1)
260 CNT = 0
270 '
```

```
280 ' Program checks last record first because
290 ' it won't be checked in the regular search
300 '
310 GET #1,LAST
320 IF NAME$ = NM$ THEN 450
330 '
340 ' Program compares NM$ with NAME$ from record
350 ' MID until NM$ is found or enough records
360 ' are seen to show that it isn't in the file
370 '
380 GET #1,MID
390 IF CNT > (LOF(1)+1)/2 THEN 710
400 IF NAME$ < NM$ THEN 570
410 IF NAME$ > NM$ THEN 640
420 '
430 ' When NM$ is found, it is printed out
440 '
450 CLS
460 PRINT @ 104,NAME$
470 PRINT @ 136,"(";AREA$;")";PHONE$
480 PRINT @ 195,PRESS <ENTER> TO CONTINUE."
490 PRINT @ 227,"ELSE PRESS <0> TO QUIT"
500 AN$ = INKEY$
510 IF AN$ = "0" THEN CLOSE #1:END
520 IF AN$ = CHR$(13) THEN 140
530 GOTO 500
540 '
550 ' Subprogram for when NAME$ < NM$
560 '
570 FIRST = MID
580 MID = (MID+LAST)/2
590 CNT = CNT+1
600 GOTO 380
610 '
620 ' Subprogram for when NAME$ > NM$
630 '
640 LAST = MID
650 MID = (MID+FIRST)/2
660 CNT = CNT+1
```

```
670 GOTO 380
680 '
690 ' Subprogram for when NM$ is not found
700 '
710 CLS
720 PRINT @ 100,N1$;" NOT FOUND
730 PRINT @ 132, "TO TRY AGAIN PRESS <ENTER>"
740 AN$ = INKEY$
750 IF AN$ = "" THEN 740
760 GOTO 140
770 '
780 ' Subprogram for modifying NM$ to a 24-byte
790 ' string
800 NM$ = NM$+" "
810 GOTO 210
820 NM$ = LEFT$(NM$,24)
830 GOTO 220
```

## Sample Program #5 / Update the List

Update anything you want in the file you created in Sample Program #3. You can do it easily with this program.

```
10 ' Edit Your Names file
20 '
30 ' This program edits the NAMES/NOS file
40 ' from Sample Program #3.
50 ' It can retain a record, change a variable
60 ' in that record, or delete the record.
70 '
80 CLS
90 PRINT @ 106, "SELECTIONS:"
100 PRINT @ 168,"1) EDIT RECORD"
110 PRINT @ 200,"2) DELETE RECORD"
120 PRINT @ 232,"3) END JOB"
130 PRINT @ 298,"PRESS(1 2 OR 3)"
140 AN$ = INKEY$
150 IF AN$="" THEN 140
```

```
160 ON VAL(AN$) GOSUB 180,590,850
170 GOTO 80
180 OPEN "D",#1,"NAMES.NOS",35
190 FIELD #1,24 AS NAME$,3 AS AREA$, 8 AS PHONE$
200 FOR I=1 TO LOF(1)
210 GET #1,I
220 CLS
230 PRINT @ 68,"RECORD NUMBER:";I
240 PRINT @ 100, "NAME: "NAME$
250 PRINT @ 132, "AREA CODE :";AREA$
260 PRINT @ 164, "PHONE NUMBER; "; PHONE$
270 PRINT @ 228,"EDIT THIS RECORD? (Y/N)"
280 AN$ = INKEY$
290 IF AN$ = "Y" THEN 320
300 IF AN$ = "N" THEN 560
310 GOTO 280
320 PRINT @ 260, "EDIT NAME? (Y/N)"
330 AN$=INKEY$
340 IF AN$ = "N" THEN NM$ = NAME$:GOTO 400
350 IF AN$ = "Y" THEN 370
360 GOTO 330
370 LINE INPUT " NEW NAME ";NM$
380 IF LEN(NM$) < 24 THEN NM$ = NM$+" ":GOTO 380
    ELSE 390
390 IF LEN(NM$) > 24 THEN NM$ = LEFT$(NM$,24)
400 PRINT @ 292,"EDIT AREA CODE? (Y/N)"
410 AN$ = INKEY$
420 IF AN$ = "Y" THEN 450
430 IF AN$ = "N" THEN A$ = AREA$ : GOTO 460
440 GOTO 410
450 INPUT " NEW AREA CODE";A$
460 PRINT @ 324, "EDIT PHONE NUMBER? (Y/N)"
470 AN$ = INKEY$
480 IF AN$ = "Y" THEN 510
490 IF AN$ = "N" THEN P$ = PHONE$ : GOTO 520
500 GOTO 470
510 INPUT," NEW PHONE NUMBER";P$
520 LSET NAME$ = NM$
530 LSET AREA$ = A$
```

```
540 LSET PHONE$ = P$
550 PUT #1,I
560 NEXT I
570 CLOSE #1
580 GOTO 900
590 OPEN "D",#1,"NAMES/NOS",35
600 FIELD #1,24 AS NAME$,3 AS AREA$, 8 AS PHONE$
610 OPEN "D",#2,"TEMP/FIL",35
620 FIELD "#2,24 AS TNAME$,3 AS TAREA$,8 AS
    TPHONE$
630 FOR I=1 TO LOF(1)
640 GET #1,I
650 CLS
660 PRINT @ 68,"RECORD #";I
670 PRINT @ 100, "NAME: ";NAME$
680 PRINT @ 132, "AREA CODE: ";AREA$
690 PRINT @ 164, "PHONE NUMBER: "; PHONE$
700 PRINT @ 228, "DELETE THIS RECORD? (Y/N)"
710 AN$ = INKEY$
720 IF AN$ = "Y" THEN 800
730 IF AN$ = "N" THEN 750
740 GOTO 710
750 LSET TNAME$ = NAME$
760 LSET TAREA$ = AREA$
770 LSET TPHONE$ = PHONE$
780 J=J+1
790 PUT #2,J
800 NEXT I
810 CLOSE
820 KILL "NAMES/NOS"
830 RENAME "TEMP/FIL" TO "NAMES/NOS"
840 RETURN
850 END
900 DIM ARRAY$(30)
910 OPEN"D",#1,"NAMES/NOS",35
920 FIELD #1, 35 AS INFO$
930 OPEN"D",#2,"TEMP/FIL",35
940 FIELD #2, 35 AS NEW$
950 FOR I=1TOLOF(1)
```

```
960 GET #1,I
970 ARRAY$(I)=INFO$
980 NEXT I
990 FOR J=1 TO LOF(1)
1000 FOR K=J TO LOF(1)
1010 IF ARRAY$(J)<ARRAY$(K) THEN 1050
1020 TEMP$=ARRAY$(J)
1030 ARRAY$(J)=ARRAY$(K)
1040 ARRAY$(K)=TEMP$
1050 NEXT K
1060 LSET NEW$=ARRAY$(J)
1070 PUT #2, J
1080 NEXT J
1090 GOTO 810
```

## Sample Program #6 / Grading Tests

This program is ideal for teachers. It creates several disk files of students and their test scores. You can then immediately find averages and the standard deviation for the entire class or for each student.

```
10 ' Test program
20 '
30 ' This program inputs several files -- a names
40 ' file and several test files. You can access
50 ' the files and process the test scores to
60 ' find averages and standard deviations.
70 ' The files are sequential access.
80 '
90 '
100 ' Main module of program
110 '
120 DIM NAME$(30),GRADE(6,30)
130 CLS
140 PRINT @ 107,"SELECTIONS"
150 PRINT @ 164,"1) CREATE A 'NAMES' FILE"
160 PRINT @ 196,"2) ADD A NEW TEST FILE"
170 PRINT @ 228,"3) PROCESS SCORES"
```

```
180 PRINT @ 260,"4) END"
190 PRINT @ 331, "1,2,3 OR 4"
200 AN$=INKEY$
210 IF AN$="" THEN 220
220 ON VAL(AN$) GOSUB 290,430,640,1430
230 GOTO 130
240 '
250 ' This subroutine builds a NAMES file.
260 '
270 OPEN "O",#1,"NAME/FIL"
280 CLS
290 PRINT @ 96,"ENTER NAME OF STUDENT:"
300 LINE INPUT NAME$
310 WRITE #1,NAME$
320 PRINT @ 196,"(PRESS <ENTER> TO ENTER":PRINT @
    228,"ANOTHER NAME, PRESS <Q>"PRINT @ 260,"TO
    QUIT)"
330 AN$=INKEY$
360 IF AN$="" THEN 330
370 IF AN$<>"Q" THEN 300
380 CLOSE #1
390 RETURN
400 '
410 ' This subroutine builds test files.
420 '
430 CLS
440 PRINT @ 64
450 INPUT "NUMBER OF NEW TEST FILE";TF$
460 IF TF$ ="" THEN 450
470 TF$ = "TEST" + TF$
480 OPEN "I",#1,"NAME/FIL"
490 OPEN "O",#2,TF$
500 IF EOF(1) THEN 560
510 INPUT #1,NAME$
520 PRINT "NAME:"NAME$
530 INPUT "SCORE";SCORE
540 WRITE #2,SCORE
550 GOTO 500
560 CLOSE #1,#2
```

```
570 RETURN
580 '
590 ' This subroutine inputs the NAMES file and
600 ' the desired test files, processes them
610 ' either on a class basis or an individual
620 ' basis, and then prints out the results.
630 '
640 OPEN "I",#1,"NAME/FIL"
650 IF EOF(1) = -1 THEN 690
660 Y = Y + 1
670 INPUT #1,NAME$(Y)
680 GOTO 650
690 YEND = Y
700 CLOSE #1
710 CLS
720 PRINT @ 96
730 INPUT " HOW MANY TESTS ARE THERE";N
740 FOR X=1 TO N
750 TF$ = "TEST" + RIGHT$(STR$(X),1)
760 OPEN "I",#1,TF$
770 FOR Y=1 TO YEND
780 INPUT #1,GRADE(X,Y)
790 NEXT Y
800 CLOSE #1
810 NEXT X
820 CLS
830 PRINT @ 130,"INDIVIDUAL TOTALS OR CLASS"
840 INPUT " TOTALS(I/C)";AN$
850 IF AN$ = "I" THEN 900
860 IF AN$ = "C" THEN 1130
870 '
880 ' This part processes the scores by the student.
890 '
900 FOR Y=1 TO YEND
910 CLS
920 PRINT @ 105,NAME$(Y)
930 PRINT @ 137,"SCORES:"
940 STUTOT = 0
950 FOR X=1 TO N
```

```
960 PRINT TAB(10) GRADE (X,Y)
970 STUTOT = STUTOT + GRADE(X,Y)
980 NEXT X
990 AVE(Y) = STUTOT / N
1000 NUM = 0
1010 FOR X=1 TO N
1020 NUM = (AVE(Y) - GRADE(X,Y)) 2 + NUM
1030 NEXT X
1040 SD = SQR(NUM / N)
1045 FM$="%"+STRING$(5,32)+"%##.##"
1050 PRINT USING FM$;"AVERAGE:";AVE(Y)
1055 FM$="%"+STRING$(17,32)+"%##.##"
1060 PRINT USING FM$;"STANDARD DEVIATION:";SD
1070 PRINT "PRESS <ENTER> TO SEE NEXT NAME"
1080 AN$ = INKEY$
1090 IF AN$ = CHR$(13) THEN 1100 ELSE 1080
1100 NEXT Y
1110 CLS
1120 PRINT @ 105, "NO MORE NAMES"
1130 GOTO 1350
1140 '
1150 ' This portion processes the scores by the
1160 ' test number for the whole class.
1170 '
1180 INPUT " WHICH TEST NUMBER";X
1190 CLS
1200 PRINT @ 4, "DATA FOR TEST NUMBER ";X
1210 PRINT "NAME";TAB(25)"SCORE"
1220 TTOT = 0
1230 FOR Y=1 TO YEND
1240 TTOT = TTOT + GRADE(X,Y)
1250 PRINT TAB(1) NAME$(Y);TAB(25)GRADE(X,Y)
1260 NEXT Y
1270 AVE = TTOT / YEND
1280 NUM = 0
1290 FOR Y=1 TO YEND
1300 NUM = NUM (AVE - GRADE(X,Y)2
1310 NEXT Y
1320 SD = SQR(NUM / (YEND - 1))
```

```
1325 FM$="%"+STRING$(16,32)+"%#%%##.##"
1330 PRINT:PRINT USING FM$; "AVERAGE FOR TEST
     #";X;":";AVE
1335 FM$="%"+STRING$(17,32)+"%##.##"
1340 PRINT USING FM$;"STANDARD DEVIATION: ";SD
1350 PRINT:PRINT " PRESS <ENTER> FOR MORE"
1360 PRINT " PROCESSING, <Q> TO QUIT"
1370 AN$ = INKEY$
1380 IF AN$ = CHR$(13) THEN 820
1390 IF AN$ = "Q" THEN 1400 ELSE 1370
1400 RETURN
1410 '
1420 ' This subroutine terminates the program.
1430 '
1440 END
```

## Sample Program #7 / Create-a-Game

The following four programs display three settings: a house and two rooms. Each setting is stored on disk as a program file.

```
10 ' "DISPLAY/BAS"
20 '
30 ' This program shows how you can
40 ' access another program from your main
50 ' program. It uses a main program called
60 ' DISPLAY/BAS and three graphics programs
70 ' called HOUSE/BAS, FOYER/BAS, and
80 ' STAIRS/BAS. (Naturally, they must be on disk
90 ' before you can run this program.)
100 '
110 CLS
120 PRINT @ 106, "SELECTIONS:"
130 PRINT @ 170, "1) HOUSE"
140 PRINT @ 202, "2) FOYER"
150 PRINT @ 234, "3) STAIRS"
160 PRINT @ 266, "4) END JOB"
170 PRINT @ 330, "1,2,3, OR 4"
```

```
180 AN$ = INKEY$
190 IF AN$ = "" THEN 180
200 IF AN$ = "4" THEN 250
210 CLS
220 PRINT @ 98,"TO RETURN FROM THIS SELECTION"
230 PRINT @ 130,"PRESS ANY KEY"
240 FOR I=1 TO 40:NEXT I
250 ON VAL(AN$) GOTO 260,270,280,290
260 LOAD "HOUSE/BAS",R
270 LOAD "FOYER/BAS",R
280 LOAD "STAIRS/BAS",R
290 END

10 ' "HOUSE/BAS"
20 '
30 PMODE 3,1
40 PCLS
50 SCREEN 1,0
60 DRAW "BM66,108;D48;R32;U48;L32"
70 DRAW "BM66,68,R132;BM46,96,R132;MB50,156;
   R128"
80 DRAW "BM50,96;D60;BM178,96,;D60;BM206,88;
   D50"
90 DRAW "BM0,136;R50;BM206,136;R50"
100 LINE (46,96)-(66,68),PSET
110 LINE (178,96)-(198,68),PSET
120 LINE (198,68)-(206,88),PSET
130 LINE (174,156)-(206,136),PSET
140 CIRCLE (92,130),5,0
150 PAINT (0,0),3,4
160 PAINT (0,149),1,4
180 PAINT (55,105),2,4
190 PAINT (194,96),2,4
200 PAINT (82,128),3,4
210 AN$ = INKEY$
220 IF AN$ = "" THEN 210
230 LOAD "DISPLAY/BAS",R
```

```
10 ' "FOYER/BAS"
20 '
30 PMODE 3,1
40 PCLS
50 SCREEN 1,0
60 DRAW "BM104,60;D92;R48;U92;L48"
70 DRAW "BM44,20;R168;D132;L132;BL4;L12;BL4;
   L16;BM44,102;U82"
80 DRAW "BM220,60;D100;BM244,58;D126"
90 DRAW "BM42,102;D38;RB;U38;L8"
100 DRAW "BM16,148;D40;R4;U40"
110 DRAW "BM64,148;D40;L4;U40"
120 DRAW "BM80,124;D40;L4;U36"
130 CIRCLE (144,108),4
140 CIRCLE (238,117),4
150 CIRCLE (45,140),15,4,.3,0,.7
160 CIRCLE (45,140),15,4,.3,.95,1
170 CIRCLE (53,136),4
180 LINE (0,192)-(16,176),PSET
190 LINE (20,172)-(44,152),PSET
200 LINE (256,192)-(212,152),PSET
210 PAINT (28,8),3,4
220 LINE (0,0)-(44,20),PSET
230 LINE (256,0)-(212,20),PSET
240 LINE (220,60)-(244,59),PSET
250 LINE (16,148)-(44,124),PSET
260 LINE (16,148)-(64,148),PSET
270 LINE (64,148)-(80,124),PSET
280 LINE (80,124)-(52,124),PSET
290 PAINT (10,10),3,4
300 PAINT (60,32),3,4
310 PAINT (240,20),3,4
320 PAINT (128,64),2,4
330 PAINT (228,70),2,4
340 PAINT (62,156),4,4
350 PAINT (78,150),4,4
360 PAINT (18,156),4,4
370 PAINT (68,128),1,4
380 PAINT (128,156),2,4
```

```
390 PAINT (40,140),4,4
400 PAINT (48,120),2,4
410 CIRCLE (46,98),5,2,2
420 AN$ = INKEY$
430 IF AN$ = "" THEN 420
440 LOAD "DISPLAY/BAS",R

10 ' "STAIRS/BAS"
20 '
30 PCLS
40 PMODE 3,1
50 SCREEN 1,0
60 DRAW "BM60,20;R140;D120;L40;U32;L4;D52;R4;
   U20;BM160,160;L128;U150"
70 DRAW "BM4,62;D130;BM28,166;U102;BM144,148;
   R12"
80 DRAW "BM40,72;D24;R36;U24;L36;BM44,76;D16;
   R28;U16;L28"
90 LINE (32,12)-(92,12),PSET
100 LINE (92,12)-(100,20),PSET
110 LINE (0,0)-(60,20),PSET
120 LINE (200,20)-(225,0),PSET
130 LINE (200,140)-(255,192),PSET
140 LINE (0,192)-(32,160),PSET
150 LINE (4,62)-(28,64),PSET
160 PAINT (120,4),2,4
170 PAINT (20,20),2,4
180 PAINT (230,20),2,4
190 PAINT (120,40),2,4
200 PAINT (60,16),3,4
210 PAINT (20,64),3,4
220 PAINT (158,124),4,4
230 PAINT (42,74),4,4
240 LINE (28,8)-(144,148),PSET
250 LINE (64,12)-(156,122),PSET
260 LINE (68,12)-(156,116),PSET
270 DRAW "BM144,148:U38"
280 FOR I=0 TO 9
290 DRAW "BM-8,28;U38"
```

```
300 NEXT I
310 DRAW "BM56,40,U28,31,U18;MB40,22;U10"
320 PAINT (56,84),2,4
330 CIRCLE (56,86),10,3,.4,0,.5
340 LINE (51,86)-(63,86),PSET
350 DRAW "BM56,84;L4;E7;D8"
360 FOR I=1 TO 32
370 CIRCLE (120,176),I*2,I/4,.25
380 NEXT I
382 DRAW "BM232,176;U100;R21;D100"
383 CIRCLE (232,180),15,4,1,.5,0
384 CIRCLE (232,178),6,4,2,.55,.1
385 CIRCLE (232,80),15,4,1,0,.5
386 CIRCLE (232,82),6,4,2,.1,.55
390 AN$ = INKEY$
400 IF AN$ = "" THEN 390
410 LOAD "DISPLAY/BAS",R
```

## Sample Program #8 / Budgeting

This program organizes your finances and prints out a journal on your printer. You need a line printer with a line length of at least 80 characters to run it.

```
10 ' BUDGET PROGRAM
20 '
30 ' This program builds three direct access
40 ' files: a listing of a balanced budget; a
50 ' listing of transactions, and a listing of the
60 ' updated budget. It allows for carryover from
70 ' the previous period's budget. You can print
80 ' out a list of the budget, expenses, and
90 ' balances. As written, this program requires
100 ' a printer. With slight modification, it
110 ' can be used without a printer.
120 '
130 '
140 ' Main module program
150 '
```

```
160  CLS
170  PRINT @ 106, "SELECTIONS:"
180  PRINT @ 165, "1) BUILD BUDGET"
190  PRINT @ 197, "2) UPDATE AN ACCOUNT
200  PRINT @ 229, "3) PRINT OUT A JOURNAL"
210  PRINT @ 261, "4) END JOB"
220  PRINT @ 329, "1,2,3, OR 4?"
230  AN$=INKEY$
240  IF AN$="" THEN 230
250  ON VAL(AN$) GOSUB 360,910,1410,1940
260  GOTO 160
270  '
280 ' This subroutine builds the Budget file
290 ' (called BUDGET/ORT), and builds or updates
300 ' the file BUDGET/UPD. It lets you input the
310 ' start date and the amount you have to
320 ' divide among accounts. Tentative amounts
330 ' are entered for each account and a running
340 ' balance advises you of the amount left.
350 '
360 OPEN "D",#1, "BUDGET/ORG",5
370 OPEN "D",#2, "BUDGET/UPD",5
380 FIELD #1,5 AS OAMT$
390 FIELD #2,5 AS UPDAMT$
400 GOSUB 1780
410 IF LOF(2) = 0 THEN 470
420 FOR I=1 TO 9
430 GET #2,I
440 AMT(I) =CVN(UPDAMT$)
450 PTOT=PTOT + AMT(I)
460 NEXT I
470 CLS
480 PRINT @ 130, "DATE(MM/DD/YY):"
490 PRINT @ 96
500 INPUT "DATE(MM/DD/YY):":DATE$
510 PRINT @ 162, "PROJECTED INCOME FROM :"
520 INPUT " SALARY:";SAL
530 INPUT " OTHER:";OTHER
```

```
540 BTOT=SAL + OTHER
550 CLS
560 PRINT @ 9, "CURRENT BUDGET"
570 PRINT " acct# description balance"
580 SUMBUD = 0
590 FOR I=1 TO 9
600 PRINT USING "#####% % %                    %
    ####.##-";ACNO(I);" ";DESC$(I);AMT(I)
610 SUMBUD = SUMBUD + AMT(I)
620 NEXT I
630 PRINT @ 86,USING "$####.##-";AMT(1)
640 PRINT @ 387,USING "%                    %$$#####.
    ##"; "remaining money: ":BTOT - (SUMBUD -
    PTOP)
650 PRINT @ 451, "ENTER ACCT# OF ITEM TO BE"
660 INPUT " CHANGED (000 TO QUIT)";AN
670 IF AN = 0 THEN 750
680 CLS
690 N = AN / 100
700 PRINT @ 105,ACNO (N)
710 PRINT @ 138,DESC$(N)
720 PRINT @ 170,"$";AMT(N)
730 PRINT :INPUT " NEW AMOUNT";AMT(N)
740 GOTO 550
750 DATE = VAL(LEFT$(DATE$,2) + MID$(DATE$,4,2)
    + RIGHT$(DATE$,2))
760 LSET OAMT$ = MKN$(DATE)
770 PUT #1,1
780 FOR I=1 TO 9
790 LSET OAMT$ = MKN$(AMT(I))
800 LSET UPDAMT$ = MKN$(AMT(I))
810 PUT #1,I+1
820 PUT #2,I
830 NEXT I
840 CLOSE
850 RETURN
860 '
870 ' This subroutine builds a transaction file
880 ' called TFILE/DAT that contains any updates
```

```
890 ' to the budget, and updates BUDGET/UPD
900 '
910 OPEN "D",#1,"BUDGET/UPD",5
920 OPEN "D",#2,"TFILE/DAT",36
930 FIELD #1,5 AS UPDAMT$
940 FIELD #2,3 AS ACON$,8 AS DATE$,20 AS DESC$,5
    AS TAMT$
950 FOR I=1 TO 9
960 GET #1,I
970 AMT(I) = CVN(UPDAMT$)
980 NEXT I
990 GOSUB 1780
1000 CLS
1010 SUMBUD =0
1020 PRINT @ 9;"CURRENT BUDGET"
1030 PRINT " acct$ description balance"
1040 FOR I=1 TO 9
1050 PRINT USING "#####% % %
     %#####.##-";ACNO(I);" ";DESC$(I);AMT(I)
1060 SUMBUD = SUMBUD - AMT (I)
1070 NEXT I
1080 PRINT @ 86,USING "$####.##-";AMT(I)
1090 PRINT @ 387,USING "%              %$$###.##";
     "total balance:";SUMBUD
1100 PRINT @ 451,"ENTER ACCT# OF ITEM TO BE
1110 INPUT " UPDATED (000 TO QUIT)";AN
1120 IF AN = 0 THEN 1310
1130 CLS
1140 N = AN / 100
1150 PRINT @ 95,AN
1160 PRINT DESC$(N)
1170 PRINT USING "% %$              $###,##";
     "CURRENT BALANCE";AMT (N)
1180 PRINT:INPUT "DATE(MM/DD/YY)";DT$
1190 PRINT "DESCRIPTION OF TRANSACTION:"
1200 INPUT DS$
1210 PRINT "AMOUNT OF TRANSACTION:"
1220 PRINT "(NEGATIVE NUMBER FOR A CREDIT)"
1230 INPUT TRANS
```

```
1240 AMT(N) = AMT(N) -TRANS
1250 LSET ACNO$ = RIGHT$(STR$(AN).3)
1260 LSET DATE$ = DT$
1270 LSET DESC$ = DS$
1280 LSET TAMT$ = MKN$(TRANS)
1290 PUT #2, LOF (2)+1
1300 GOTO 1000
1310 FOR I+1 TO 9
1320 LSET UPDAMT$ = MKN$(AMT(I))
1330 PUT #1,I
1340 NEXT I
1350 CLOSE
1360 RETURN
1370 '
1380 ' This subroutine prints out a listing of
1390 ' the budget, transactions, and balances.
1400 '
1410 OPEN "D", #1, "BUDGET/ORG",5
1420 FIELD #1,5 AS AMT$
1430 OPEN "D",#2, "TFILE/DAT",36
1440 FIELD #2, 3 AS TACNO$, 8 AS TDATE$, 20 AS
     TRDESC$, 5 AS TMT$
1450 GOSUB 1780
1460 CLS
1470 PRINT $ 172, "PRINTING"
1480 GET #1,1
1490 DATE$ = STR$(CVN(AMT$))
1500 IF LEN(DATE$) < 6 THEN DATE$ = " " + DATE$
1510 DATE$ = LEFT$(DATE$,2) + "/" + MID$,DATE$,
     3,2) + "/" + RIGHT$(DATE$, 2)
1520 PRINT #-2, TAB(30)"BUDGET FOR THE PERIOD"
1530 PRINT #-2, TAB(31) "STARTING ";DATE$
1540 PRINT #-2,PRINT #-2
1550 PRINT #-2,TAB(28) "ACCOUNT OR"
1560 PRINT #-2,TAB(10)"ACCOUNT";TAB(27)
     "TRANSACTION"
1570 PRINT #-2,TAB(10)"NUMBER";TAB(20)"DATE";
     TAB(27) "DESCRIPTION";TAB(47)
     "TRANSACTION"; TAB(61)"BALANCE"
```

```
1580 FOR I=2 TO LOF(1)
1590 GET #1,I
1600 PRINT #-2
1610 PRINT #-2,TAB(10)ACNO(I-1);TAB(17)DATE$;
     TAB(27)DESC$(I-1);TAB(61)CVN;AMT$
1620 BAL=CVN(AMT$)
1630 IF LOF(2)<1 THEN 1710
1640 FOR J=1 TO LOF(2)
1650 GET #2,J
1660 IF ACNO(I-1) <> VAL(TACNO$)THEN 1700
1670 BAL=BAL-CVN(TMT$)
1680 IF CVN(TMT$) < 0 THEN CR$="CR" ELSE CR$=""
1690 PRINT #-2,TAB(17)TDATE$;TAB(27)TRDESC$;
     TAB(47)ABS(CVN(TMT$));CR$:TAB(61)BAL
1700 NEXT J
1710 NEXT I
1720 CLOSE
1730 RETURN
1740 '
1750 ' This subroutine sets values of account
1760 ' numbers, ACNO(I) and account descriptors,
1770 ' DESC$(I).
1780 FOR I=1 TO 9
1790 ACNO(I) = I* 100
1800 NEXT I
1810 DESC$(1) = "FOOD"
1820 DESC$(2) = "RENT"
1830 DESC$(3) = "CAR"
1840 DESC$(4) = "UTILITIES"
1850 DESC$(5) = "INSURANCE"
1860 DESC$(6) = "TAXES"
1870 DESC$(7) = "CLOTHING"
1880 DESC$(8) = "ENTERTAINMENT"
1890 DESC$(9) = "MISCELLANEOUS"
1900 RETURN
1910 '
1920 ' The program terminates here.
1930 '
1940 END
```

## Appendix D

# ASCII Character Codes

The following is a list of ASCII codes for the characters on your keyboard. The first column contains the character. The second lists the code in decimal notation. The third converts the code to a hexadecimal (base 16) number.

| CHARACTER | DECIMAL CODE | HEXADECIMAL CODE |
|---|---|---|
| space bar | 32 | 20 |
| ! | 33 | 21 |
| " | 34 | 22 |
| # | 35 | 23 |
| $ | 36 | 24 |
| % | 37 | 25 |
| & | 38 | 26 |
| ' | 39 | 27 |
| ( | 40 | 28 |
| ) | 41 | 29 |
| * | 42 | 2A |
| + | 43 | 2B |
| , | 44 | 2C |
| – | 45 | 2D |
| . | 46 | 2E |
| / | 47 | 2F |
| 0 | 48 | 30 |
| 1 | 49 | 31 |
| 2 | 50 | 32 |
| 3 | 51 | 33 |
| 4 | 52 | 34 |
| 5 | 53 | 35 |

| CHARACTER | DECIMAL CODE | HEXADECIMAL CODE |
|---|---|---|
| 6 | 54 | 36 |
| 7 | 55 | 37 |
| 8 | 56 | 38 |
| 9 | 57 | 39 |
| : | 58 | 3A |
| ; | 59 | 3B |
| < | 60 | 3C |
| = | 61 | 3D |
| > | 62 | 3E |
| ? | 63 | 3F |
| @ | 64 | 40 |
| A | 65 | 41 |
| B | 66 | 42 |
| C | 67 | 43 |
| D | 68 | 44 |
| E | 69 | 45 |
| F | 70 | 46 |
| G | 71 | 47 |
| H | 72 | 48 |
| I | 73 | 49 |
| J | 74 | 4A |
| K | 75 | 4B |
| L | 76 | 4C |
| M | 77 | 4D |
| N | 78 | 4E |
| O | 79 | 4F |
| P | 80 | 50 |
| Q | 81 | 51 |
| R | 82 | 52 |
| S | 83 | 53 |
| T | 84 | 54 |
| U | 85 | 55 |
| V | 86 | 56 |
| W | 87 | 57 |

| CHARACTER | DECIMAL CODE | HEXADECIMAL CODE |
|---|---|---|
| X | 88 | 58 |
| Y | 89 | 59 |
| Z | 90 | 5A |
| ↑ * | 94 | 5E |
| ↓ * | 10 | 0A |
| ← * | 8 | 08 |
| → * | 9 | 09 |
| BREAK | 03 | 03 |
| CLEAR | 12 | 0C |
| ENTER | 13 | 0D |

* In shift mode, the codes for these characters are as follows: CLEAR is 92 (hex 5C); ↑ is 95 (hex 5F); ↓ is 91 (hex 5B); ← is 21 (hex 15); and → is 93 (hex 5D). Refer to the following table.

The following characters are not on your keyboard but can be displayed by "printing" the associated code. In BASIC you can type:

**PRINT CHR$ (*code*)** ENTER

*code* represents a character's associated code.

| CHARACTER | DECIMAL CODE | HEXADECIMAL CODE |
|---|---|---|
| [ | 91 | 5B |
| \ | 92 | 5C |
| ] | 93 | 5D |
| ↑ | 94 | 5E |
| ← | 95 | 5F |
| @* | 96 | 60 |
| *  | 123 | 7B |
| *  | 124 | 7C |
| *  | 125 | 7D |
| *  | 126 | 7E |
| *  | 127 | 7F |

* For the Color Computer 3, different characters are displayed between the low-resolution text screen (WIDTH 32) and high-resolution text screen (WIDTH 40 & WIDTH 80) modes.

# Lower-Case Codes

These are the ASCII codes for lower-case letters. You can produce these characters by pressing the (SHIFT) and (Ø) keys simultaneously to get into an upper-/lower-case mode. With the Color Computer and Color Computer 2, lower-case letters appear on the screen as upper-case letters in reverse type (green type on a black background). This is also the means of display used on a Color Computer 3 in WIDTH 32 mode. On a Color Computer 3 in WIDTH 40 or WIDTH 80, true lower-case letters can be displayed.

| CHARACTER | DECIMAL CODE | HEXADECIMAL CODE |
|---|---|---|
| a | 97 | 61 |
| b | 98 | 62 |
| c | 99 | 63 |
| d | 100 | 64 |
| e | 101 | 65 |
| f | 102 | 66 |
| g | 103 | 67 |
| h | 104 | 68 |
| i | 105 | 69 |
| j | 106 | 6A |
| k | 107 | 6B |
| l | 108 | 6C |
| m | 109 | 6D |
| n | 110 | 6E |
| o | 111 | 6F |
| p | 112 | 70 |
| q | 113 | 71 |
| r | 114 | 72 |
| s | 115 | 73 |
| t | 116 | 74 |
| u | 117 | 75 |
| v | 118 | 76 |
| w | 119 | 77 |
| x | 120 | 78 |
| y | 121 | 79 |
| z | 122 | 7A |

## Appendix E

# Memory Maps

# For the Color Computer and Color Computer 2

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| 0-255 | 0000-00FF | System Direct Page RAM | |
| 256-1023 | 0100-03FF | Extended Page RAM | |
| 1024-1535 | 0400-05FF | Video Text Memory | |
| 1536-2440 | 0600-0988 | Additional System RAM | This is used exclusively by Disk BASIC. |
| 2441-*top of RAM*<br><br>*top of RAM* is 16383 for 16K systems, 32767 for 32K systems | 0989-*top of RAM*<br><br>*top of RAM* is 3FFF for 16K systems, 7FFF for 32K systems | These Random Access Memory locations are allocated dynamically and contain the following: | |
| | | 1. Random File Buffer Area | Total buffer space for random access files. On startup, 256 bytes are reserved for this. This value can be reset by the FILES statement. |
| | | 2. File Control Blocks (FCBs) | Control data on each user buffer. On startup, 843 bytes are reserved for this. This value can be reset by the FILES statement: (*number of buffers set by FILES* + 1) x 281 bytes. |

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| | | 3. Graphics Video Memory | Space reserved for graphics video pages. On startup 6144 bytes, or four pages, are reserved for this. This value can be reset by the PCLEAR statement: *number of pages reserved by PCLEAR* × 1,536 bytes per page. (Note: All pages must start at a 256-byte page boundary, a memory location divisible by 256.) |
| | | 4. BASIC Program Storage<br>5. BASIC Variable Storage<br>6. Stack | Space reserved for BASIC programs and variables. On startup, 6455 bytes (16K systems) or 22,839 bytes (32K systems) are reserved for this. The value can be reset by different settings of random file buffers, FCBs, graphics video memory string space or user memory. |

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| | | 7. String Space | Total space for string data. On startup, 200 bytes are reserved, but this value can be reset by the CLEAR statement. |
| | | 8. User Memory | Total space for user machine-language routines. No space is reserved for this on startup, but you can use the CLEAR statement to reserve some. |
| 32768-40959 | 8000-9FFF | Extended Color BASIC ROM | Read Only Memory |
| 40960-49151 | A000-BFFF | Color BASIC ROM | Read Only Memory |
| 49152-57343 | C000-DFFF | Color Disk BASIC ROM | Read Only Memory |
| 57344-65279 | E000-FEFF | Unused | |
| 65280-65535 | FF00-FFFF | Input/Output | |

# For the Color Computer 3

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| 458752-459007 | 70000-700FF | System Direct Page RAM | |
| 459008-459775 | 70100-703FF | Extended Page RAM | |
| 459776-460287 | 70400-705FF | Video Text Memory | |
| 460288-461192 | 70600-70988 | Additional System RAM | This is used exclusively by Disk BASIC. |
| 461193-*top of RAM* | 70989-*top of RAM* | These Random Access Memory locations are allocated dynamically and contain the following: | |
| | | 1. Random File Buffer Area | Total buffer space for random access files. On startup, 256 bytes are reserved for this. This value can be reset by the FILES statement. |
| | | 2. File Control Blocks (FCBs) | Control data on each user buffer. On startup, 843 bytes are reserved for this. This value can be reset by the FILES statement: (*number of buffers set by FILES* + 1) x 281 bytes. |

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| | | 3. Graphics Video Memory | Space reserved for graphics video pages. On startup, 6144 bytes, or four pages, are reserved for this. This value can be reset by the PCLEAR statement: *number of pages reserved by PCLEAR* × 1,536 bytes per page. (Note: All pages must start at a 256-byte page boundary, a memory location divisible by 256.) |
| | | 4. BASIC Program Storage<br>5. BASIC Variable Storage<br>6. Stack | Space reserved for BASIC programs and variables. On startup, 6455 bytes are reserved for this. The value can be reset by different settings of random file buffers, FCBs, graphics video memory string space or user memory. |

| DECIMAL | HEX | CONTENTS | DESCRIPTION |
|---|---|---|---|
| | | 7. String Space | Total space for string data. On startup, 200 bytes are reserved, but this value can be reset by the CLEAR statement. |
| | | 8. User Memory | Total space for user machine-language routines. No space is reserved for this on startup, but you can use the CLEAR statement to reserve some. |
| 491520-499711 | 78000-79FFF | Extended Color BASIC ROM | Read Only Memory |
| 499712-507903 | 7A000-7BFFF | Color BASIC ROM | Read Only Memory |
| 507904-516095 | 7C000-7DFFF | Color Disk BASIC ROM | Read Only Memory |
| 516096-523775 | 7E000-7FDFF | Super Extended BASIC | |
| 523776-524031 | 7FE00-7FEFF | Secondary Vector | |
| 524032-524287 | 7FF00-7FFFF | Input/Output | |

## Appendix F

# Specifications

| | |
|---|---|
| Type of Disks | 5 1/4-inch floppy diskettes<br>Radio Shack Cat. No. 26-305,<br>26-405 (package of three),<br>or 26-406 (package of ten) |
| Disk Organization<br>(Formatted Disk) | *Double-sided<br>Double-density<br>40 tracks (80 tracks, if both sides are used)<br>18 sectors per track<br>256 data bytes per sector<br>Directory on Track 17 (Disk Extended Basic) |
| Operating Temperature | 55 to 85 degrees Fahrenheit<br>12.8 to 29.4 degrees Celsius |
| BTU/H | 90 |
| Memory Capacity | |
| Unformatted | 500 kilobytes per disk<br>6.2 kilobytes per track |
| Soft Sector<br>(I/O Sector/Track) | 322 kilobytes per disk<br>4.6 kilobytes per track |
| Data Transmission Speed | 250 kilobits per second |
| Access Time | |
| Track to Track | 6 m/sec. |
| Average | 100 m/sec. |
| Settling Time | 15 m/sec. |
| Number of Indexes | 1 |
| Disk Controller | WD1773 |

**Note:** Double-sided, double-density diskettes can be used under OS-9 Level Two operation. We recommend Radio Shack Cat. No. 26-411 (package of three) or 26-412 (package of ten).

* When using OS-9.

## Appendix G

# Error Messages

| *Error* | *No.* | *Description* |
|---|---|---|
| /0 | | **Division by zero.** You asked the computer to divide a number by 0, which is impossible. You might also get this error message if you do not enclose a filename in quotation marks. |
| AE | 33 | **File already exists.** You are trying to rename or copy a file to a filename that already exists. |
| AO | | **Attempt to open a data file that is already open.** |
| BR | 27 | **Bad record number.** You have used an impossible record number in your PUT or GET line. The number is either too low (less than 1) or too high (higher than the maximum number of records the computer can fit on the disk). Use a different record number in the PUT or GET line, or assign a smaller record length in the OPEN line. |
| BS | | **Bad subscript.** The subscripts in an array are out of range. Use DIM to dimension the array. For example, if you have A(12) in your program, without a preceding DIM line that dimensions array A for 12 or more elements, you get this error. |
| CN | | **Can't continue.** If you use the command CONT, and you are at the end of the program, you get this error. |
| DD | | **Attempt to redimension an array.** An array can be dimensioned only once. For example, you cannot have DIM A(12) and DIM A(50) in the same program. |
| DF | 28 | **Disk full.** The disk you are trying to store your file on is full. Use another disk. |

| *Error* | *No.* | *Description* |
|---|---|---|
| DN | | **Drive number or device number error.**<br>Drive number error. You are using a drive number higher than 3. You also get this error if you do not specify a drive number when using DSKINI or BACKUP. If you have only one drive, specify Drive 0 with these two commands (**DSKINIØ** or **BACKUP Ø**).<br>Device number error. You are using more buffers than the computer has reserved. Use FILES to reserve more. You might also get this error if you use a non-existing buffer number (such as Buffer # – 3) or omit the buffer (such as **FIELD 1 AS A$** rather than **FIELD #1, 1 AS A$**). |
| DS | | **Direct statement.** There is a direct statement in the data file. This could be caused if you load a program that has no line numbers. |
| ER | 37 | **Write or input past end of record** (direct access only). You are attempting to put more data in the record than the record can hold or trying to input more data than it contains. |
| FC | | **Illegal function call.** This error happens when a number that you use with a BASIC word is out of range. For example, **SOUND (260,260)** and **CLS(10)** cause this error. Using **RIGHT$ (S$, 20)** when there are only ten characters in S$ would also cause it. Other examples include using a negative subscript, such as A(–1), or using a USR call before defining the address. |
| FD | | **Bad file data.** This error occurs when you print data to a file or input data from the file, using the wrong type of variable for the corresponding data. For example, using **INPUT #1, A,** when the data in the file is a string, causes this error. |
| FM | | **Bad file mode.** You have specified the wrong file mode (O, I, or D) in your OPEN line. For example, you are attempting to get a record from a file opened for input using I (instead of D), or you are trying to write data to a file opened for output. |
| FN | 31 | **Bad filename.** You used an unacceptable format to name your file. |

| *Error* | *No.* | *Description* |
|---|---|---|
| FO | 34 | **Field overflow**. The field length is longer than the record length. |
| FS | 32 | **Bad file structure**. There is something wrong with your disk file. Either the data was written incorrectly or the directory track on the disk is bad. See the IO error description for instructions. |
| ID | | **Illegal direct statement**. You can use INPUT only as a line in the program, not as a command line. |
| IE | | **Input past end of file**. Use EOF or LOF to check to see when you reach the end of the file. When you do, close the file. |
| *HP | | **High-resolution print error**. This error occurs if you attempt to execute a high-resolution text function on a low-resolution text screen or to execute a low-resolution text function on a high-resolution text screen. |
| *HR | | **High-resolution graphics error**. This error occurs if you attempt to execute a high-resolution graphics statement without first setting up a high-resolution screen, using the HSCREEN statement. |
| IO | | **Input/Output error**. The computer is having trouble inputting or outputting information.<br>(1) Be sure that a disk is properly inserted in the indicated drive and that the drive latch is closed.<br>(2) If you still get this error, there might be something wrong with your disk. Try re-inserting the disk. If this fails, reformat the disk or use different one. (Remember that reformatting a disk erases its contents.)<br>(3) If you still get this error, you probably have a problem with the computer system itself. Call a Radio Shack Computer Center.<br>This error can also be caused by input/output problems with another device, such as a tape recorder. |

* Color Computer 3 only.

| Error | No. | Description |
|---|---|---|
| LS | | **String too long.** A string can contain a maximum of 255 characters. |
| NE | 26 | **The computer can't find the disk file you want.** Check the disk's directory to see if the file is there. If you have two disk drives, you might not have included the appropriate drive number in the filename. If you are using COPY, KILL, or RENAME, you might have left off the extension. |
| NF | | **NEXT without FOR.** NEXT is being used without a matching FOR statement. This error also occurs when you have the NEXT lines reversed in a nested loop. |
| NO | | **File not open.** You cannot transfer data to or from a unopened file. |
| OB | 29 | **Out of buffer space.** Use FILES to reserve more space. |
| OD | | **Out of data.** You executed a read with insufficient data for the computer to read. Perhaps you left out a DATA statement. |
| OM | | **Out of memory.** All available memory is used or reserved. |
| OS | | **Out of string space.** There is not enough space in memory to do your string operations. Use CLEAR at the beginning of your program to reserve more string space. |
| OV | | **Overflow.** The number is too large for the computer to handle. |
| RG | | **RETURN without GOSUB.** A RETURN line in your program has no matching GOSUB. |
| SE | 35 | **Set to non-fielded string.** The field in which you are attempting to left- or right-justify data has not yet been fielded. Check the FIELD line. |
| SN | | **Syntax error.** This error can result from a misspelled command, incorrect punctuation, an unmatched open parenthesis, or an illegal character. Retype the program line or command. |

| *Error* | *No.* | *Description* |
|---|---|---|
| ST | | **String formula too complex.** A string operation is too complex for the computer to handle. Break the operation into shorter steps. |
| TM | | **Type mismatch.** This error occurs when you try to assign numeric data to a string variable (`A$ = 3`) or string data to a numeric variable (`A = "DATA"`). This can also occur if you do not enclose a filename in quotation marks. |
| UL | | **Undefined line.** In the program, you have a GOTO, GOSUB, or other branching line that asks the computer to go to a nonexisting line number. |
| VF | 36 | **Verification.** You only get this error when you have the VERIFY command on and are writing to a disk. The computer is informing you that there is a flaw in what it wrote. See the IO error description for instructions. |
| WP | 30 | **Write-protected.** You are trying to store information on a disk that is write-protected. Either remove the label from the write-protect notch, or use a different disk. If your disk is not write-protected, then there is an input/output problem. See the IO error description for instructions. |

## Appendix H

# Disk BASIC Summary

This appendix contains a summary of the BASIC commands you can use with your Color Computer. In addition to these Disk BASIC commands, you can continue to use the Extended BASIC commands, which are built into your computer.

Beside each BASIC word is the chapter to which you can refer for more information.

The first line of each command's description gives the format to use in typing the command. The italicized words represent *parameters*, variables that you replace with specific values when you enter the command.

These are the meanings of some of the parameters:

*filename*
All information stored on a disk must have a *filename*. The *filename* should be in this format:

*name/extension: drive number*

The *name* is mandatory. It can have 1-8 characters.
The *extension* is optional. It can have 1-3 characters.
The *drive number* is optional. If you do not use it when opening a disk file, the computer uses Drive 0 (or the drive that you specify in the DRIVE command).

*number*
This can be a number (such as 1 or 5.3), a numeric variable (such as A or BL), a numeric function (such as ABS(3)), or a numeric operation (such as 5+3 or A−7).

*string*
This can be a series of characters (such as "B" or "STRING"), a string variable (such as A$ or BL$), a string function (such as LEFT$(S$, 5)), or a string operation (such as "M"+A$).

*data*
This can be *number* or *string*.

| CHAPTER | BASIC WORD |
|---|---|

Chapter 3

BACKUP *source drive* TO *destination drive*
Duplicates the contents of the disk in the *source drive* to the disk in the *destination drive*. If you have only one drive, specify it as the *source drive*, the computer prompts you to switch disks as it makes the backup copy. Executing this command erases memory.

```
BACKUP Ø TO 1     BACKUP Ø
```

Chapter 5

CLOSE *#buffer,...*
Closes communication to the *buffers* specified. (See OPEN for information *buffer* numbers). If you omit the *buffer*, the computer closes all open files.

```
CLOSE #1     CLOSE #1, #2
```

Chapter 4

COPY *"filename1"* TO *"filename2"*
Copies the contents of *filename1* to *filename2*. Each *filename* must include an *extension*. (See the filename information at the beginning of this appendix.) Executing this command erases memory.

```
COPY "FILE/BAS" TO "NEWFILE/BAS"

COPY "ORG/DAT:Ø" TO "ORG/DAT:1"
```

Chapter 9

CVN(*string variable*)
Converts a 5-byte coded *string* (created by MKN$) back to the number it represents.

```
X=CVN(A$)
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 2

DIR *drive number*
Displays a directory of the disk in the drive you specify. If you omit *drive number*, the computer uses Drive 0. (Unless you use the DRIVE command to change this default drive.) This is a typical directory display:

```
MYPROG     BAS    Ø B 3
YOURPROG   BAS    Ø A 1
HERDATA    DAT    1 A 5
USPROG     BIN    2 B 2
```

The first column contains the name of the file. The second column contains its extension. The third gives the file type (0 = BASIC program, 1 = BASIC data file, 2 = machine-language file, 3 = editor source file). The fourth column tells the format in which the file is stored (A = ASCII, B = binary). The fifth column contains the length of the file, in granules.

```
DIRØ    DIR
```

Chapter 2

DOS
With the OS-9 system diskette in Drive 0, the DOS command boots the OS-9 operating system.

```
DOS
```

Chapter 4

DRIVE *drive number*
Changes the drive default to the drive you specify. If you do not use the DRIVE command, the computer uses Drive 0.

```
DRIVE 1
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 11

DSKI$ *drive, number, track, sector, string variable1, string variable2*
Inputs data from a certain *sector* within a certain *track* on the disk in the specified drive. The first 128 bytes of data are input into *string variable1*, the second 128 bytes into *string variable2*.

```
DSKI$ Ø, 12, 3, M$, N$
```

Chapter 2

DSKINI *drive number*
Formats a disk in the drive you specify. Executing this command erases memory.

```
DSKINIØ     DSKINI1
```

Chapter 11

DSKO$ *drive number, track, sector, string1, string2*
Writes string data on the *sector, track,* and *drive number* you specify. *string1* is written to the first 128 bytes of the sector; *string2* is written to the second 128 bytes. Used improperly, this command can garble the contents of the disk.

```
DSKO$ Ø, 2, 1, "FIRST DATA", "SECOND
DATA"
```

Chapter 5

EOF(*buffer*)
Returns a value of 0 if there is more data to read in the *buffer* and a value of −1 if there is no more data in it. (See OPEN for the *buffer* numbers.)

```
IF EOF(1) = -1 THEN CLOSE #1
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 9

FIELD *#buffer, field size* AS *field name,...*
Organizes the space within a direct access *buffer* into *fields*. (See OPEN for the *buffer* numbers.) You specify the *size* and *name* of each *field*.

```
FIELD #1, 10 AS A$, 12 AS B$, 5 AS C$
```

Chapter 10

FILES *number of buffers, size*
Tells the computer how many buffers to reserve in memory, and the total number of bytes to reserve for these buffers (*size*). If you do not use FILES, the computer reserves enough memory space for two buffers (Buffers 1 and 2), and reserves a total of 256 bytes for those buffers.

```
FILES 1, 1000     FILES 5
```

Chapter 4

FREE (*drive number*)
Returns the number of free granules on the disk in the drive you specify.

```
PRINT FREE(0)
```

Chapter 7

GET *#buffer, record number*
Gets the next record or the record you specify, and puts it in the *buffer*. (See OPEN for information on *buffer* numbers).

```
GET #1, 5     GET #2, 3
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 5

INPUT *#buffer, variable name,...*
Inputs data from the *buffer* you specify and assigns each data item in the *buffer* to the *variable name* you specify. (See OPEN for information on *buffer* numbers.)

```
INPUT #1, A$, B$
```

Chapter 4

KILL *"filename"*
Deletes the *filename* you specify from the disk directory. (See the filename information at the beginning of this appendix.) You must include the *extension* with the *filename*.

```
KILL "FILE/BAS"    KILL "FILE/DAT:1"
```

Chapter 8

LINE INPUT *#buffer, data*
Inputs a line (all the *data* up to the (ENTER) character) from the *buffer* you specify. (See OPEN for information on *buffer* numbers).

```
LINE INPUT #1, X$
```

Chapter 2

LOAD *"filename"*, R
Loads the BASIC program file you specify into memory from a disk. If you include the R, the computer runs the program immediately after loading it. If your *filename* does not have an *extension*, the computer assumes that the extension is BAS. (See the filename information at the beginning of this appendix.) Executing this command erases memory.

```
LOAD "PROGRAM", R

LOAD "ACCTS/BAS:1"
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 11 LOADM *"filename", offset address*
Loads a machine-language program file from disk. You can specify an *offset address* to add to the program's *loading address*. If your filename does not have an *extension*, the computer assumes that the extension is BIN. (See the filename information at the beginning of this appendix.)

```
LOADM "PROG/BIN", 3522
```

LOC(*buffer*)
Returns the current record number of the buffer you specify. (See OPEN for information on *buffer* numbers.)

```
PRINT LOC(1)
```

Chapter 7 LOF(*buffer*)
Returns the highest numbered record of the buffer you specify. (See OPEN for information on *buffer* numbers.)

```
FOR R = 1 TO LOF(1)
```

Chapter 9 LSET *field name = data*
Left-justifies the *data* within the field you specify. If the *data* is larger than the field, the computer *truncates* (chops off) characters on the right side.

```
LSET A$ = "BANANAS"     LSET B$ = T$
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 10

MERGE "*filename*", R
Loads a program *file* from disk, and merges it with the existing program in memory. If you include R, the computer immediately runs the program after merging it. (See the filename information at the beginning of this appendix.) To merge a program file, you must first save it with the A (ASCII) option.

```
MERGE "SUB/BAS"    MERGE "NEW", R
```

Chapter 9

MKN$(*number*)
Converts the specified number to a 5-byte coded string, for storage in a formatted disk file.

```
LSET B$ = MKN$(5367891Ø)
```

Chapter 5

OPEN "*mode,*" #*buffer, filename, record length*
Opens a place in memory, called a *buffer*, which transfers data to and from a device.

The communication *modes* and the types of data transfer they allow are as follows:

| Mode | Allows |
|---|---|
| I | Data input from a sequential access file. |
| O | Data output to a sequential access file. |
| D | Data transfer to or from a direct access file. |

| CHAPTER | BASIC WORD |
|---|---|

The *buffers* are:

| Buffer | Communicates With |
|---|---|
| −2 | The printer. |
| −1 | The tape recorder. |
| 0 | The screen or printer. (It is not necessary to open this buffer.) |
| 1-15 | The disk drive(s). |

The *filename* must be in the format described at the beginning of this appenidx. If you do not give the *filename* an *extension*, the computer gives it the *extension* DAT.

If you are opening communication to a direct access file, you can also specify the *record length*. If you do not, the computer makes the *record length* 256 bytes.

```
OPEN "D", #1, "FILE", 15

OPEN "I", #2, "CHGE/DAT"
```

Chapter 8

PRINT *#buffer, data list*
Prints the *data* to the *buffer*. (See OPEN for information on *buffer* numbers.) You can use a comma or a semicolon to separate each item in the *data list*.

```
PRINT #1, "DATA"
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 8

PRINT *#buffer,* USING *format; data list*
Prints data to the *buffer* using the *format* you specify. The *format* is a string that specifies either a format for numerical data or a format for string data.

The commands that you can use in a *format* for numerical data are:

| Command | Function |
|---|---|
| # | Holds a space for one digit. |
| . | Prints a decimal point. |
| , | Prints a comma immediately preceding every third digit (counting to the left from the decimal point). |
| ** | Fills leading spaces with asterisks. |
| $ | Prints a leading dollar sign. |
| $$ | Prints a floating dollar sign. |
| + | Prints the sign of the number. To print the sign in front of the number, place the plus sign at the beginning of the format string. To print the sign following the number, place the plus sign at the end of the format string. |
| ^^^^ | Prints the number in exponential format. |
| — | Prints a minus sign after the number, if the number is negative. Place the minus sign at the right end of the format string. |

```
PRINT #1, USING "##.#"; 53.76

PRINT #2, USING "##$##.##-"; -3.678
```

| CHAPTER | BASIC WORD |
|---|---|

The commands that you can use in a *format* for string data are:

| Command | Function |
|---|---|
| ! | Prints the first character of the string. |
| %*spaces*% | Sets the field for the string. The length of the field is the number of spaces plus 2. |

```
PRINT #1, USING "!"; "WHITE"

PRINT #1, USING "% %"; "YELLOW"
```

See the BASIC manual that came with your computer for more information on the formats.

Chapter 7

PUT *#buffer, record number*
Assigns a *record number* to the data in the *buffer*. If you do not specify a *record number*, the computer assigns it to the current record. (See OPEN for information on *buffer* numbers.)

```
PUT #2, 3     PUT #1, 4
```

Chapter 4

RENAME *"old filename"* TO *"new filename"*
Renames a disk file. You must specify the extension of both *filenames*.

```
RENAME "MFILE/DAT:1" TO "BFILE/DAT:1"
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 9

RSET *field name = data*
Right-justifies the *data* within the field you specify. If the *data* is larger than the field, the computer truncates (chops off) characters on the right side.

```
RSET M$ = "SOAP"
```

Chapter 2

RUN "*filename*", R
Loads *filename* from disk, and runs it. If you include R, all open files remain open. (See the format for *filenames* at the beginning of this appendix.)

```
RUN "FILE"     RUN "PROG/BAS", R
```

Chapter 2

SAVE "*filename*", A
Saves *filename* on disk. If you do not give *filename* an *extension*, the computer gives it the extension BAS. By using the A option, you save your program in ASCII format. (See the format for *filenames* at the beginning of this appendix.)

```
SAVE "PROG/BAS"     SAVE "TEST:1", A
```

Chapter 11

SAVEM "*filename*", *first address, last address, execution address*
Saves *filename*, a machine-language program residing in the memory location that begins at *first address* and ends at *last address*. You also specify the *address* at which the program is to be executed. If you do not give *filename* an *extension*, the computer gives it the extension BIN. (See the format for *filenames* at the beginning of this appendix.)

```
SAVEM "FILE/BIN:1", &H5200, &H5800,
&H5300
```

| CHAPTER | BASIC WORD |
|---|---|

Chapter 3

UNLOAD *drive number*
Closes any open files in the drive you specify. If you don't specify a *drive number*, the computer uses Drive 0 (or the *drive number* you specified with DRIVE).

```
UNLOAD Ø     UNLOAD
```

Chapter 3

VERIFY ON
VERIFY OFF
Turns the verify function on or off. When VERIFY is on, the computer verifies all disk writes.

Chapter 5

WRITE *#buffer, data list*
Writes the data to the *buffer* you specify. (See OPEN for information on *buffer* numbers.) Use a comma to separate each item in the data list.

```
WRITE #1, A$, B$, C
```

# Adding a Secondary Disk Drive Kit

**(Cat. No. 26-3135)**

The disk drive kit consists of the following parts:

- The disk drive
- The fan motor assembly
- The fan (with two small screws)
- Six small screws

Be sure the disk drive in this kit matches your primary drive. Pay special attention to the drive light and door latch. They should be positioned the same as those in the primary drive. If they do not match, see your Radio Shack dealer.

## Installation

The following information is provided for technicians only. The Secondary Disk Drive Kit is not intended for installation by the average user. We urge you to have this installation made by one of our technicians at your local Radio Shack Computer Center. (Doing so not only ensures expert installation, but also enables the technicians to quickly ensure that all the equipment is functioning properly.)

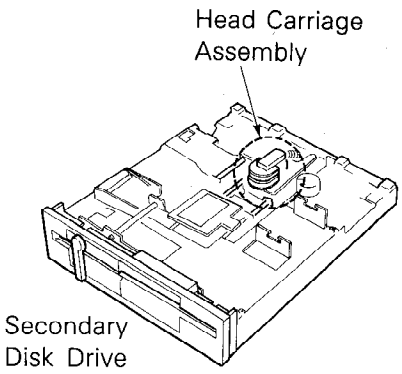
If, however, you do decide to install the kit yourself, follow these steps precisely.

**Warnings:**

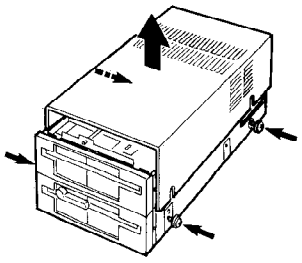
- **Turn off all equipment.** Turn off the computer and primary disk drive, and disconnect the power cord from the back of the computer. If the computer is on, you might damage the central processing unit, as well as your secondary disk drive.

- To avoid possible static charge buildup, touch a metal object to ground yourself before you begin.

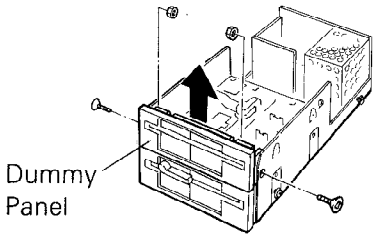
- Be **sure** to unplug the power cord from the disk drive before removing the case.



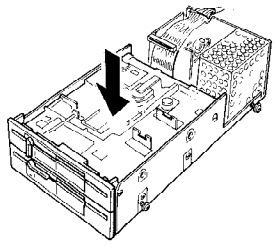

- Do not remove the protect sheet inserted into the disk drive until you finish installation and begin using the drive.
- Be sure not to touch the head carriage assembly during installation.



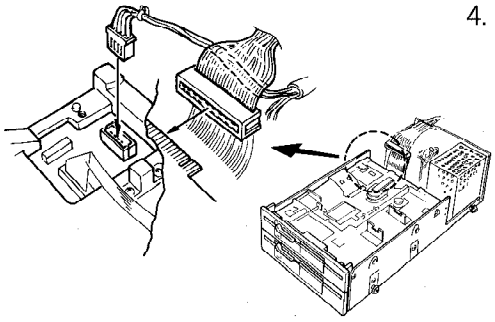
1. Loosen the four screws that fasten the cover, and pull the cover up to detach it.



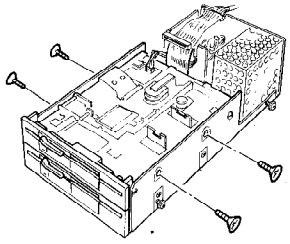

2. Remove the dummy panel by unfastening a screw and a nut from each side of the chassis.



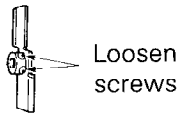
3. Carefully place the secondary floppy disk drive on the primary floppy disk drive, being sure both drives face the same direction.



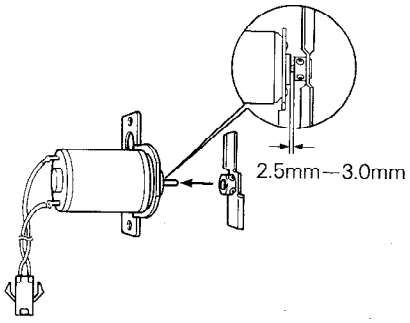
4. Connect the interface connector and the power supply. Be sure to route the power supply lead from behind the interface connector.



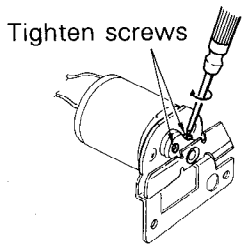
5. Attach the secondary floppy disk drive, using the four screws (dished-head screw M3 × 6) included with the kit.



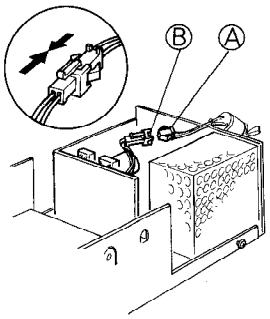

6. Loosen the two small screws affixed in the fan.




7. Affix the fan to the shaft of the motor. Be sure to leave a gap of 2.5 to 3.0 mm between the fan and the motor.

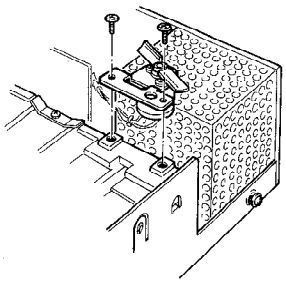



8. Apply screwlock to the threads of the screws you loosened in Step 6, and tighten them.

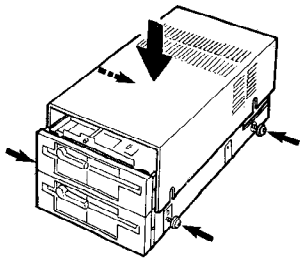



9. Connect the connector Ⓐ of the fan assembly to the connector Ⓑ that runs from power supply PCB assembly.

   These connectors can be easily disconnected by pushing down lever on the connector Ⓑ and pulling out the connector Ⓐ.

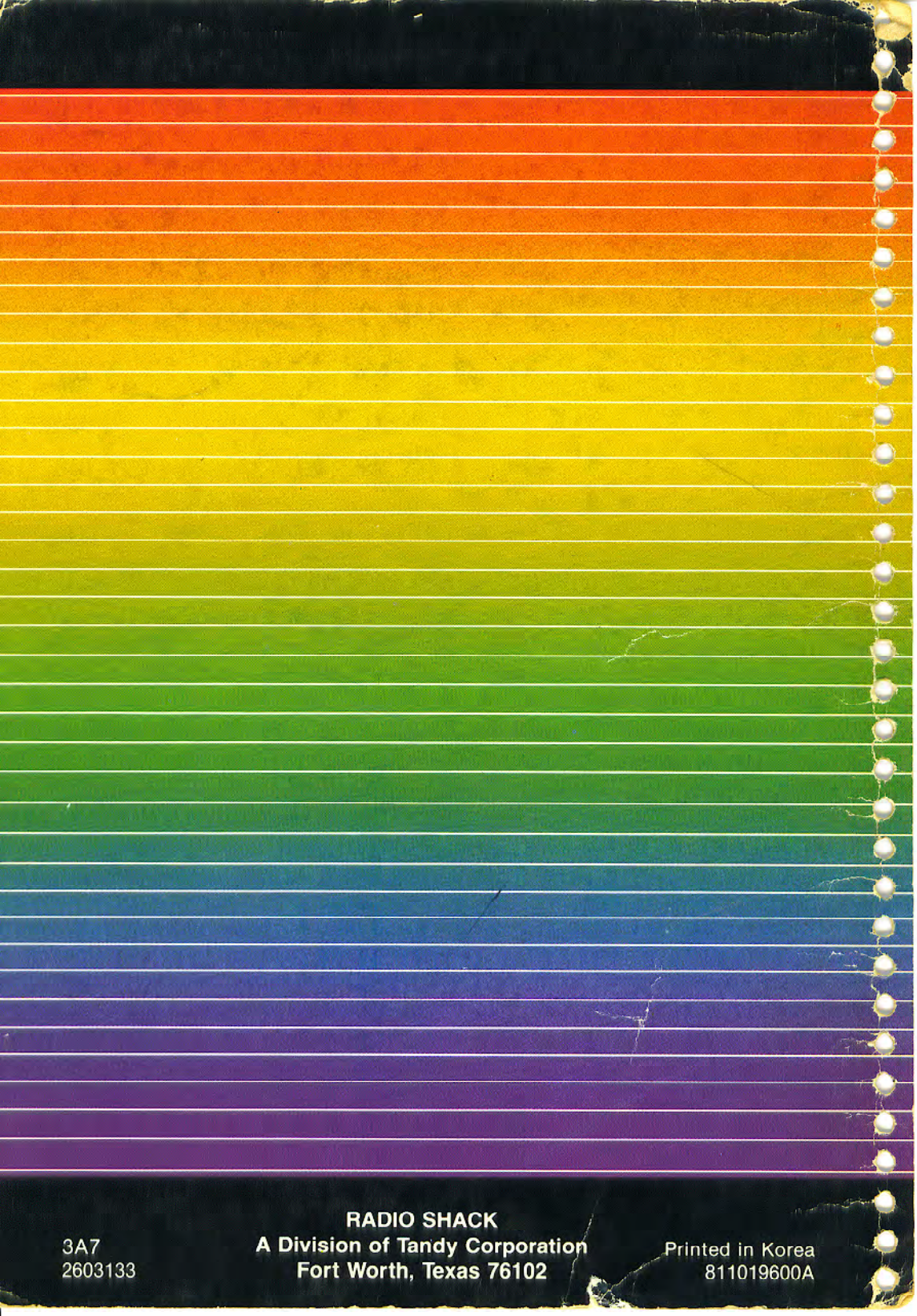


10. Mount the fan assembly on the secondary drive by using the accessory two screws, as illustrated. Be sure that the fan motor rotates without making contact with other parts.



11. Replace the cover, and fasten it with the four screws you loosened earlier.

# Index



3A7
2603133

**RADIO SHACK**
**A Division of Tandy Corporation**
**Fort Worth, Texas 76102**

Printed in Korea
811019600A