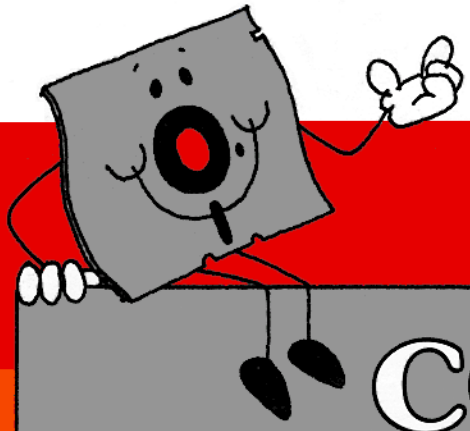


Radio Shack®

TRS-80®

COLOR COMPUTER DISK SYSTEM

Owners Manual & Programming Guide

Radio Shack
TRS-80
COLOR DISK MANUAL

CUSTOM MANUFACTURED IN U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

Important Notice: Your computer must be off when you connect the disk interface. Otherwise, you could damage the system.

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack 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. RADIO SHACK 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 RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or 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, participating Radio Shack franchisee or 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, 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 WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

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. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK 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 one host computer and access that Software through one or more terminals if the Software permits this function.

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.

The FCC wants you to know:

This equipment generates and uses radio frequency energy. If it is 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.

If necessary, the user should consult the dealer or an experienced radio / television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the United States Government Printing Office, Washington, DC 20402, Stock No. 004-000-0035-4.

Printed in the United States of America
10 9 8 7 6 5 4 3 2

WHY A DISK IS FAST

A disk is for storing your information. The precise term for it is a "mini-diskette," but in this book we'll just call it a disk. It is far superior to tape, the other alternative.

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

ABOUT THIS BOOK

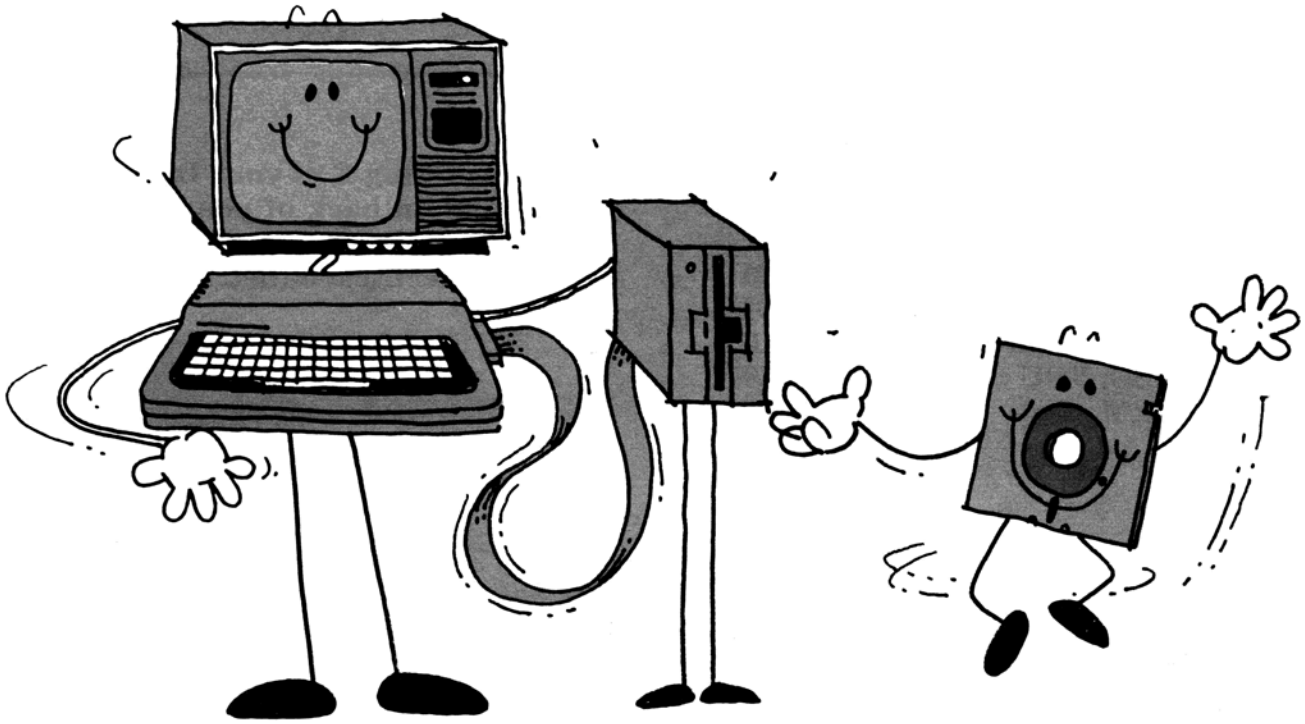
This book shows how to read and write on a disk. When we wrote it, we had three different groups of people in mind.

The first group includes all of you accomplished Radio Shack programmers. We are referring, of course, to those of you who learned to program by reading *Getting Started with Color BASIC* and *Going Ahead with Extended Color BASIC*. You'll find Sections I and II of this book another delightful experience. If you're especially ambitious, you'll also enjoy Section III.

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 Section I, "The Disk." You don't need to know anything about programming to understand it.

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 the "BASIC Summary" at the end of the book. Everything is summarized there with page number references, for the things you want to read more about.

TABLE OF CONTENTS

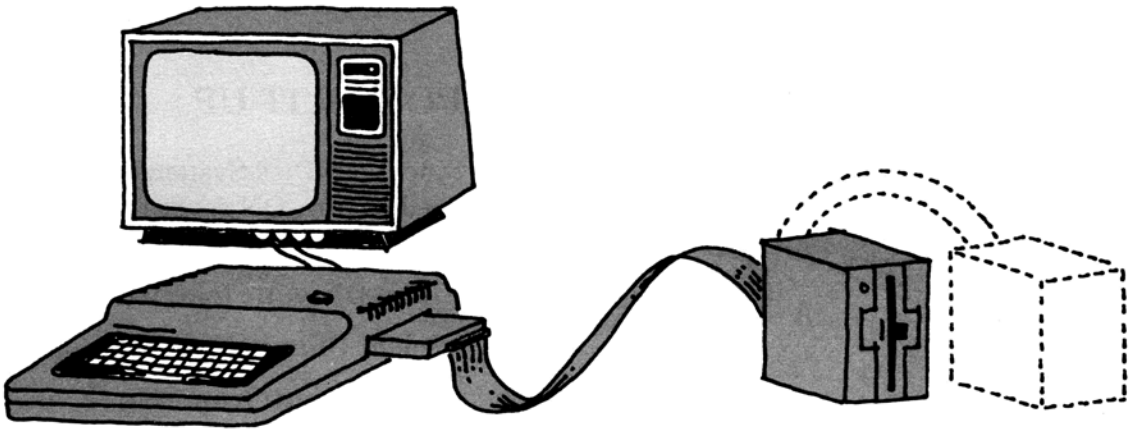


TO GET STARTED

Before you install your Disk System, you need to connect your Color Computer to the T.V. If you haven't done it yet, refer to the Color Computer Operation Manual.

A. CONNECT DISK SYSTEM

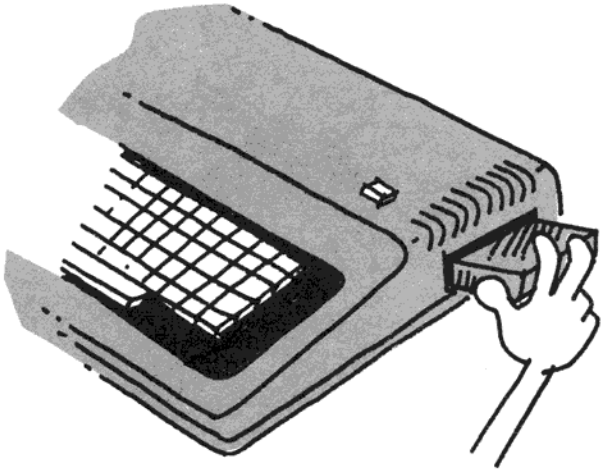
Your Disk System is easy to connect. Do it *before* you turn on your Computer by simply plugging in all the parts:



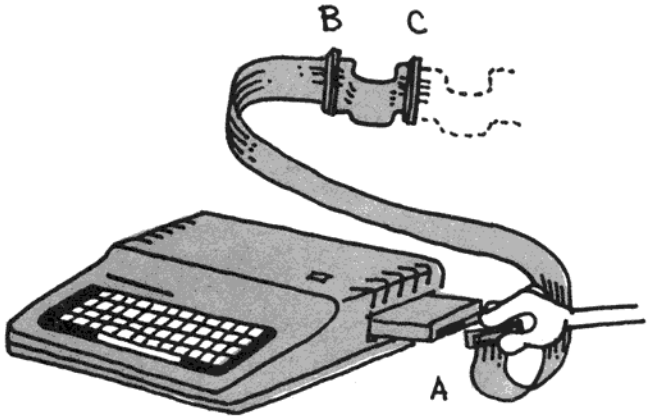
Note: *the dotted lines represent the connection of additional add-on drive .*

1. **Connect the Disk Interface to the plug in the opening of your Computer.**

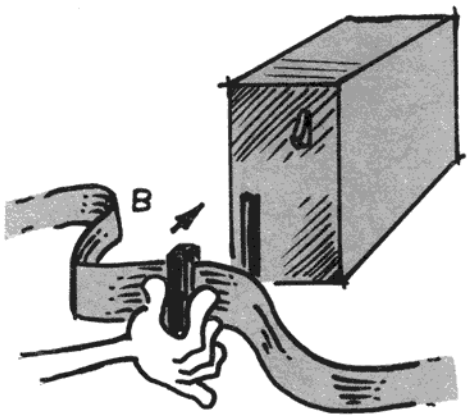
   IMPORTANT NOTICE: YOUR COMPUTER MUST BE OFF WHEN YOU CONNECT THE DISK INTERFACE. OTHERWISE, YOU COULD DAMAGE THE SYSTEM.



2. **Connect Plug A of the Disk Cable to the Disk Interface.**



3. **Connect Plug B of your Disk Cable to the plug on back of your Disk Drive. Plug in the power cord to a standard (120 V) electrical outlet.**



If You Have Additional Disk Drives

If you have more than one disk drive, do step 3 differently. Connect the 26-3023 Drive to the inside plug (Plug B). If you have more 26-3023 Drives and an expanded cable, connect these Drives to inside plugs also. The 26-3029 Drive must be connected to the last plug in the series.

You'll also need to number your Drives. Number them from the inside out, starting with Drive 0. The Drive connected to Plug B is drive number 0, the Drive connected to Plug C is drive number 1, etc.

B. POWER IT UP

Since your Disk System has several parts, you need to turn ON several buttons to power-up the entire system:

- **Turn ON your television set.**
- **Select Channel 3 or 4.**
- **Set the antenna switch on the T.V. to COMPUTER.**
- **Turn ON the Computer. (The power button is on the back left-hand side of your keyboard.)**

- **Turn ON the Disk Drives. (The power buttons are on the rear.)**

Have you turned ON all the buttons? This message should appear on your screen:

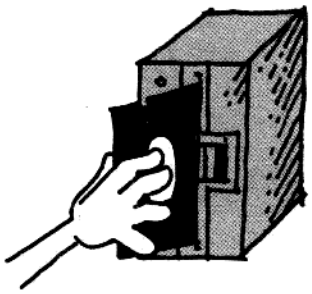
```
DISK EXTENDED COLOR BASIC v.r.
COPYRIGHT (C) 1981 BY TANDY
UNDER LICENSE FROM MICROSOFT
```

(*v.r.* is two numbers specifying which version and release you have.)

If not, turn off the Computer, check your connections, and power it up again.

C. INSERT A DISK

After powering the system up, you can insert a disk. If you plan to go through *Section I,* use the blank, unformatted disk which comes with your disk system. Otherwise, you can insert your "application program" disk. (If you have more than one drive, insert, the disk in drive 0).



- **Open the DRIVE DOOR.**
- **Position the disk with the notch on top, as we show in the picture above.**
- **Gently insert the disk until it stops.**
- **Close the DRIVE DOOR.**

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

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 *Section I.* 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 the disk, keep it in its storage envelope**
- **Do not turn the system ON or OFF with the disk in the drive.**
- **Keep disks away from magnetic fields (transformers, AC motors, magnets, TVs, radios, etc.)**
- **Handle disks by the jacket only. Don't touch any of the 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.**
- **Store disks upright in a vertical file.**

> ***Note:*** *Your disk drives should be on the* right *side of your television set.*

IMPORTANT NOTE!

If you have an earlier model of the Color Computer, the disk system might cause interference on your screen. If so, bring the computer to a Radio Shack Repair Center for additional grounding connections. (There will be no charge for this service.)

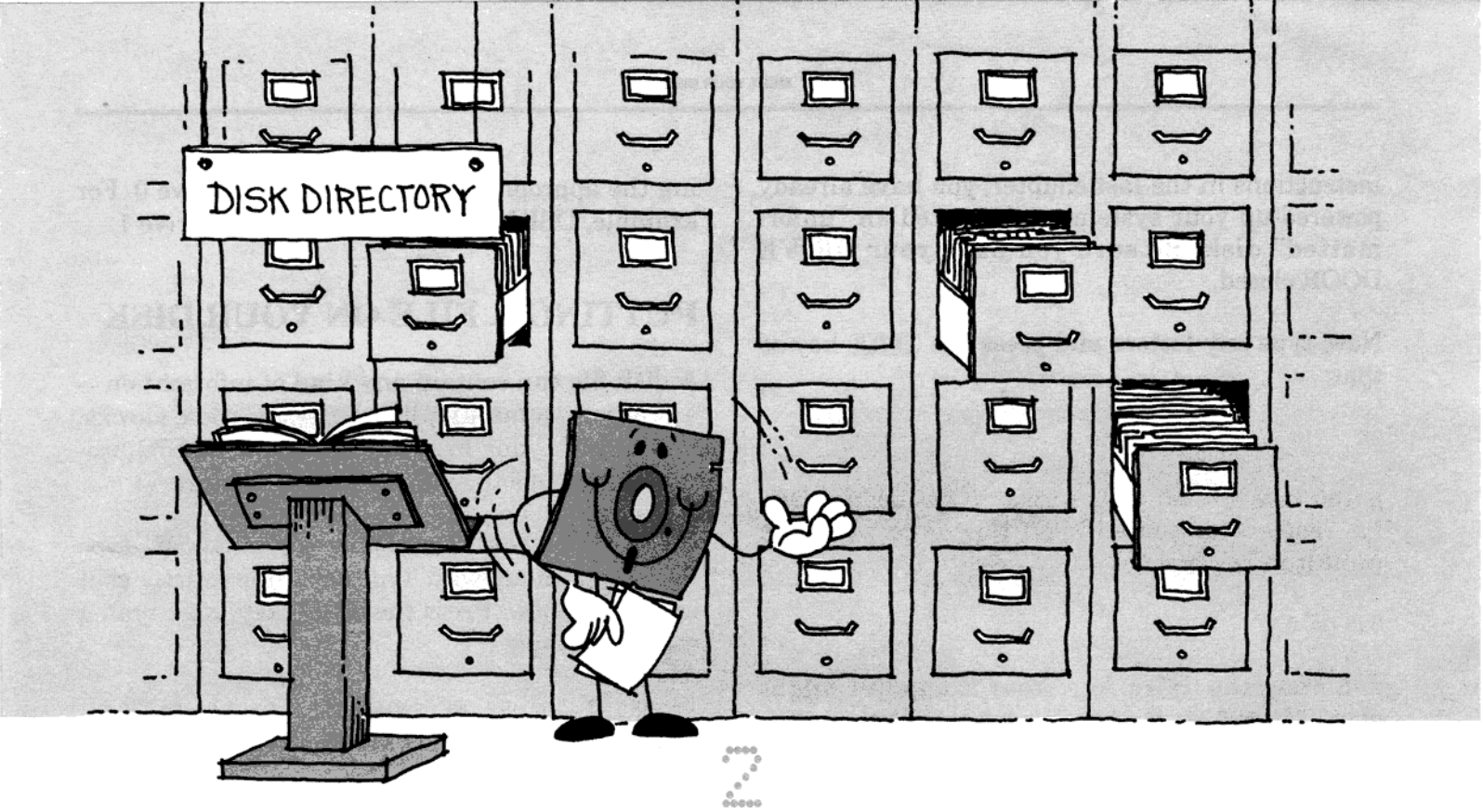
SECTION I

THE DISK

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

This makes disks easy to work with. In this section, we'll show you how your Computer organizes everything on your disk, and how you can take advantage of this.

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



MEET YOUR DISK

A LOOK INSIDE OF IT

Although your disk looks like a 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 will 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 what's important. It forms a code which 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 ... we're not finished yet... each sector is divided into bytes and each byte is divided into bits.

Note: *To be precise, there are 35 tracks on a disk, 18 sectors in each track, 256 bytes in each sector, and 8 bits in each byte.*

After creating this filing system, the Computer puts a master directory on the disk. There, it indexes where everything is stored. Whenever it wants to find something — a program, a mailing list, your letters — it uses the directory to find the tracks and sectors where it is stored. It can then go directly to that spot.

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

Putting this filing system on your disk is called "formatting" it. The last thing we had you do in *Chapter 1* was to insert an "unformatted" disk. Before you can use it, 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? If you went through the

instructions in the last chapter, you have already powered-up your system and inserted an "unformatted" disk. Be sure you have your DRIVE DOOR closed.

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 type what you want it to do. Type:

```
DSKINIØ
```

and press the (ENTER) key. Your Computer might print `?SN ERROR`. If so, don't let this bother you. This "error" simply means you typed the command incorrectly. Type it again.

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

After typing `DSKINIØ` (ENTER), you'll hear some noises from your disk drive and its red light will come on. Sounds promising . . .

After about 40 seconds of noises, your Computer will then print `OK`. It has 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 can use it.

Later on, you might not remember if a disk has been 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:*** *It does no harm to reformat a disk. This is a common way to erase everything on it.*

If you have more than one disk drive, you can format a disk in one of the other drives by substituting the appropriate drive number for drive 0. For example, DSKINI1 formats the 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. 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 it is shown below. Press the (ENTER) key after typing each line. Type:

```
1Ø PRINT "STORE ME IN A DISK FILE" (ENTER)
2Ø PRINT "AND YOU'LL NEVER LOSE ME" (ENTER)
```

Finished? Now that you've typed the program into your Computer's "memory," you can put it on a disk. To do this, we'll call it a file and name the file "SIMPLE/PRO" (all files have a name). To store it, type:

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

Once you press the (ENTER) key, your disk drive will whirr and grind some and the red light on it will come on. Your Computer is:

- finding a place on the disk to store "SIMPLE/PRO"
- telling the directory where "SIMPLE/PRO" will be stored.
- storing "SIMPLE/PRO" on your disk.

> ***Note:*** *The Computer stores "SIMPLE/PRO" the same way it stores everything else — in 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 you have stored on the disk.

When your Computer finishes storing "SIMPLE/PRO," it prints 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 it is, skip down to the next heading — "Loading a File from Disk."

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

```
RUN (ENTER)
```

Your Computer PRINTs:

```
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 it, rearrange it, combine it, or any of the other things you want done with it.

Later on, you'll probably want to put other things, such as your mailing list, in memory. To do this, you'll need to write or purchase a program written especially for that application. This "application program" will get the Computer to put the information you type into memory.

The important thing to remember about memory is that turning off your Computer erases it. Once memory has been erased, there's no way to recover it. The only way to keep a permanent copy of what you've typed into memory is by storing it on a disk (or tape).

LOADING A FILE FROM DISK

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

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

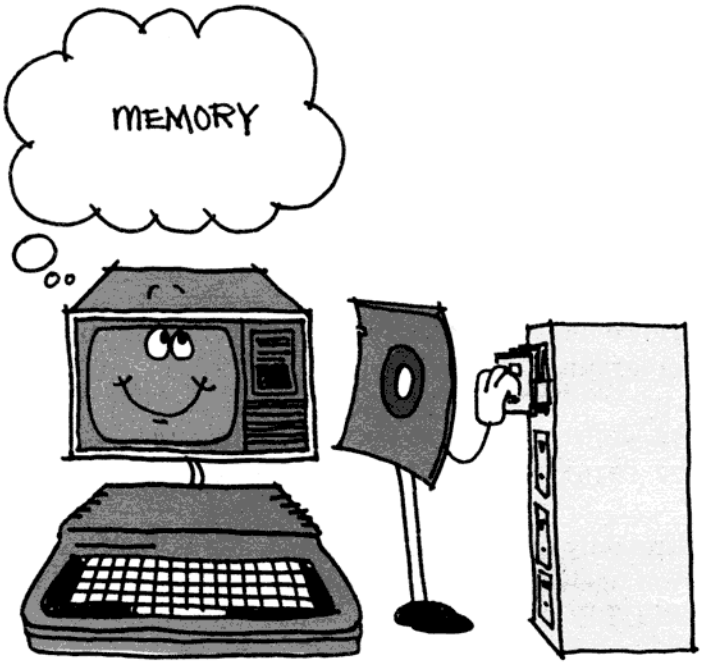
Although NEW erased the program from memory, "SIMPLE/PRO" is still safely stored on your disk. You can put "SIMPLE/PRO" back into memory anytime you want by "loading" it from disk. To do this, type LOAD "SIMPLE/PRO" (ENTER).

Again, you'll hear some promising noises from your disk drive. The Computer is:

- reading the directory to find where "SIMPLE/PRO" is stored.
- going to that location on the disk and reading the contents of "SIMPLE/PRO."
- putting "SIMPLE/PRO" into its memory.

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

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



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've

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 it in a different file by typing `SAVE "CHANGE" (ENTER)` ...

Hear the whirring and grinding from your disk drive? You have two disk files now: "SIMPLE/PRO" and "CHANGE." What do you think each of them contains? Try LOADing and then LISTing both of them.

> ***Note:*** *You don't need to type* `NEW (ENTER)` *before LOADing a new program into memory. The Computer will automatically erase everything you presently have in memory before LOADing the new program.*

??

"CHANGE" 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've used a second filename:

"CHANGE"

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

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

1. The name may have no more than eight characters.

2. If you give it an extension, the extension may have no more than three characters.

3. There must be a slash (/) or a period (.) between the name and the extension.

Fair enough? Good.

> ***Note:*** *You may use any characters in the filename except a colon (:) or a zero (0). You can only use a slash (/) or a period (.) to separate the name from the extension.*

FILENAMES WHEN YOU HAVE MORE THAN ONE DRIVE

If you have more than one disk drive, you can add the drive number to your filename. (Remember, you numbered all of your drives in *Chapter 1*). For example:

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

LOADs "SIMPLE/PRO" from the disk in drive number 1. Or

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

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

CHECKING THE MASTER DIRECTORY

As we've said earlier, a disk has a master directory which the Computer can use to find out what's on the disk. If the Computer can use it, you can use it, too. Type `DIR` (ENTER).

The Computer prints information on all the files you have stored on your disk. If the only files you've stored so far are "SIMPLE/PRO" and "CHANGE," the Computer prints this:

```
SIMPLE     PRO   0   B   1
CHANGE     BAS   0   B   1
```

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

The Computer prefers for all filenames to have an extension. If you do not give a file an extension when you store it, the Computer will automatically assign one of these extensions:

"BAS" if it's a BASIC program
"DAT" if it's data (such as names, numbers, etc.)
"BIN" if it's a machine-language program)

Note: A machine-language program is a highly technical program which talks directly to the Computer.

The next three columns contain information which is primarily for the use of technical programmers. Interested? Then read on...

The third column lists the type of file it is:

0 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
B Binary

We'll explain the meaning of this 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 of these "granules").

If you have disks inserted and formatted in other drives, you can check their directories also. For instance `DIR1` (ENTER) displays the directory of the disk in drive number 1.

Impressed? You'll be even more impressed when you see how fast you can SAVE and LOAD long programs. But before you get too involved, please read the next chapter. It'll help ensure that your experience with your Disk System is smooth and enjoyable.

Note: To stop the directory from scrolling, 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.**

A GARBLED UP 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 only use a felt-tipped pen when labeling it.

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

Since the disk is made up of magnetic charges, putting it next to another magnetic device, such as your television set, could completely 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 this command:

```
UNLOAD (ENTER)
```

before you switch disks. This way the Computer can put its closing information on the proper disk. 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 switching disks. UNLOAD closes all open files.*

BACK IT UP

All of this might sound a little gloomy to you, even if you are a careful person. This is why we've included a command called BACKUP. BACKUP will enable you to make a duplicate or "backup" copy 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 which contains important programs or data. This way you won't have to worry about losing them.

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, when the Computer begins having its problems reading and writing to the disk, you can use your backup copy.

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

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

2. Your "destination" disk—This is the disk which you want to be your duplicate copy. Use a blank disk or, if you've been using your disk system for a while, use any disk which contains files you won't need anymore.

   > ***Note:*** *Everything previously on your destination disk will be erased. It will be replaced 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, shut the door, and type `DSKINIO` (ENTER).

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

Backup with One Disk Drive

If you have only one disk drive, it will take you about five minutes to make a backup copy. Insert your "source" disk in your disk drive and shut the DRIVE DOOR. Type `DIR` (ENTER) to see which files you will be copying.

Now start the backup procedures. Type:

```
BACKUP 0 (ENTER)
```

After making some noise while it reads a portion of your "source" disk, the Computer will print:

```
INSERT DESTINATION DISKETTE AND PRESS (ENTER)
```

Take the "source" disk out and insert the "destination" disk. Shut the DRIVE DOOR. Then press (ENTER). You'll hear some more noise while the Computer "writes" some things on the "destination" disk. Then it will print:

```
INSERT SOURCE DISKETTE AND PRESS (ENTER)
```

The Computer will have you continue switching disks until you have copied everything from your source disk. During this process, make sure you insert the correct disk and insert it properly. When you've finished, the Computer will print the OK message on your screen.

To make sure BACKUP worked, you can insert your "destination" disk and type `DIR` (ENTER).

Backup with More Than One Disk Drive

If you have more than one disk drive, backing up a disk is much easier. It will take about two minutes.

Insert your "source" disk in drive 0 and your "destination" disk in drive 1 (*Chapter 1* shows how to label your drives). Then type:

```
BACKUP 0 TO 1 (ENTER)
```

You will hear some noise as the Computer backs up the contents of the disk in drive 0 to the disk in drive 1. When it's finished, it will print the `OK` message. You can then make sure BACKUP worked by typing `DIR1` (ENTER).

You can use different drives, if you want. For instance:

```
BACKUP 1 TO 0 (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 you're backing up a disk, it's probably because you've inserted the disk incorrectly or 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, you will need to try BACKUP with another disk.

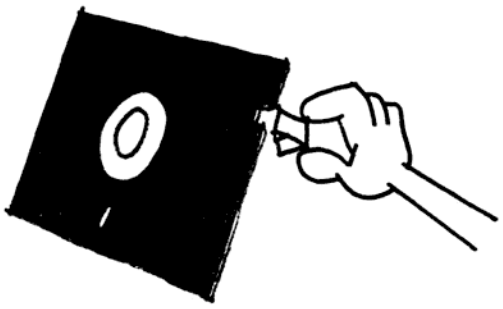
After determining the problem, press the RESET button to get out of BACKUP. Then start the BACKUP procedure all over again.

> ***Note:*** *The RESET button is on the right-hand rear of your Computer (when you're facing it).*

"WRITE" PROTECT IT

"Write-protecting" is one more way to protect your disk files. Let's assume you have a disk which contains some valuable information — such as a good program — which 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.

Putting a little gummed label on the WRITE-PROTECT NOTCH will enable the Computer to read the disk, but not to write on it. Any gummed label will do. There is one which comes with your new, unformatted disk:



SALVAGE IT

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

Although this might salvage the disk, it will not salvage the *contents* of the disk. By reformatting the disk, you will erase everything on it. However, it will save you the expense of purchasing a new disk.

If you get an IO error while trying to reformat it (see "Error Messages" at the end of this chapter), the disk has probably reached its limit. If you have a "bulk-eraser," you can try "bulk-erasing" the disk and reformatting it. Otherwise, throw it away and use another one.

> ***Note:*** *If you have more than one disk drive, 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 can do this flawlessly.

There might be times when you want to be absolutely 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 will notify you, whenever it is writing on a disk, if there are any flaws in what it is writing. The only catch is that it will take twice as long for the Computer to write.

For example, let's assume you now make a BACKUP copy of your disk. The Computer will take twice as long doing this, but will notify you if there is a flaw in the BACKUP copy.

This VERIFY command will remain ON until you turn it off. To do this, type:

```
VERIFY OFF (ENTER)
```

WHEN THINGS GO WRONG

Your Computer realizes nobody's perfect. When you make a mistake, it'll try to notify you immediately and tell you what kind of "error" you made.

You've probably already been notified that you made a "SN ERROR." If you haven't, type `DIIR` (ENTER) deliberately mispelling DIR.

SN means "Syntax" error. It's the Computer's way of telling you that "DIIR" doesn't make sense to it. 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 (discussed in the next chapter) to a filename which Already Exists.

`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 3. You will 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 (DSKINI0 or BACKUP 0)

`FN` —You used an unacceptable format to name your file. The last Chapter explains which File Names are acceptable to the Computer.

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

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

`IO` —The Computer is having trouble Inputting or Outputting information to the disk.

(1) Make sure there is a disk inserted properly in the indicated drive and 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 using a different one or reformatting it. (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 the Radio Shack Repair 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 more than one disk drive, 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 is caused by a program which mixes "strings" with "numbers." However, you might get this error if you don't enclose a filename in quotation marks.

`VF` —You will only get the 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 IO for instructions on what to do.

`WP` —You are trying to store information on a disk which is Write Protected. Either take the label off the write protect notch or use a different disk. If your disk is not Write Protected, then there is an input/output problem. See IO for instructions on what to do about this.

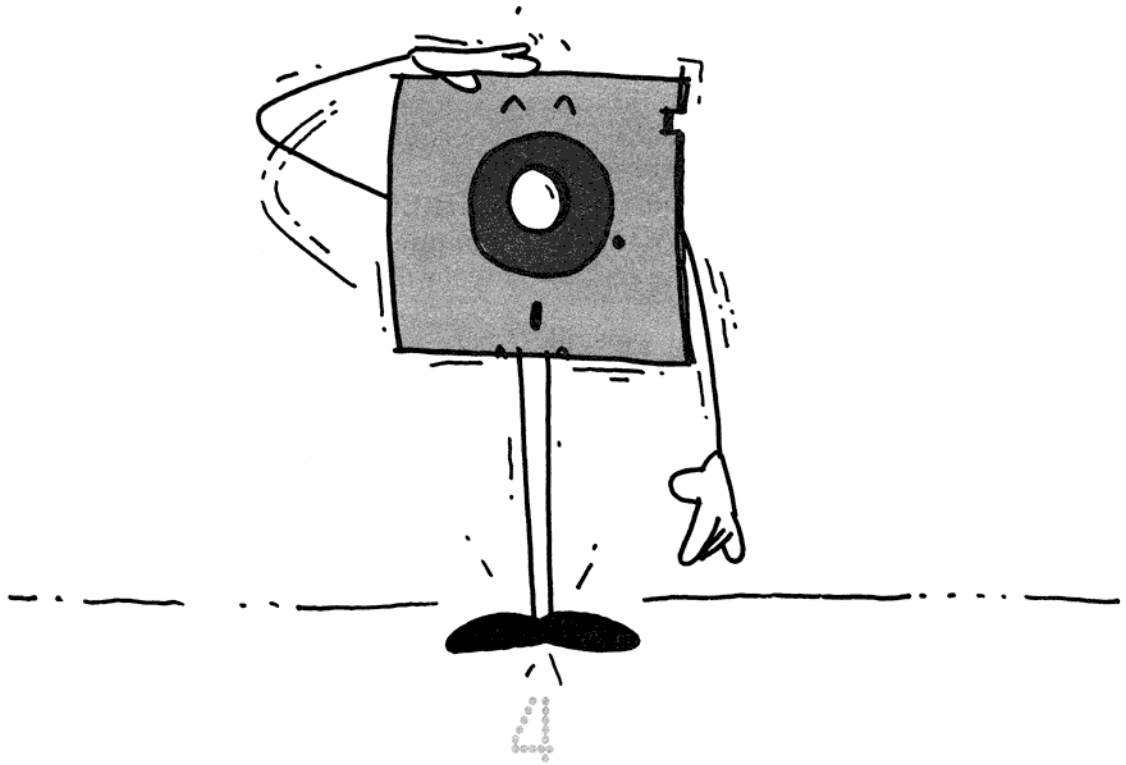
All other errors you might get are errors in the program you are using. If you did not write the program and get one of these errors, you need to contact the people who wrote it. If you did write it, check *Appendix G,* where you'll find an explanation of all the error messages.

Caring for your disk might seem a little awkward at first. It should. You've spent most of your life protecting your papers and now you're dealing with a different medium.

After awhile, 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 keeping your disk "ungarbled," you'll never want to go back to pencils and paper again (*we hope*).

☑ *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 backup a disk?***



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 rename a file. If you've taken your formatted disk out, re-insert it.

> ***Note:*** *Can't remember if your disk's formatted? Check the directory by typing* `DIR` (ENTER) *(or* `DIR0` *or* `DIR1` *if you have more than one drive).*

Type this to put a file on your disk:

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

Check the directory to see that the program file is stored on your disk under the name "ORIGINAL/NAM"... Now rename it. Type:

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

Hear the disk drive working? Check your DIRectory again. If you'd like, LOAD and LIST "NEW/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 and then try to rename it. Type:

```
10 PRINT "FILE NUMBER TWO" (ENTER)
SAVE "AFILE" (ENTER)
RENAME "AFILE" TO "BFILE" (ENTER)
```

The Computer gives you an NE error. This means the Computer can't find the file.

When you RENAME a file, you must type in the *complete* name of the file so that the Computer can find it. This includes the name and the extension. As we discussed in *Chapter 2*, whenever you SAVE a file the Computer will make sure it has an extension. If you don't assign it one, the Computer will.

You can check the directory to find out the extension of "AFILE." Then RENAME it. Type:

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

If you're renaming 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" would not have an extension. This would cause a problem when you try to LOAD "BFILE," since all files you LOAD must have an extension.

This might seem to conflict with what we said above. You were able to SAVE "AFILE" without assigning it an extension because the Computer automatically assigned it one when it saved it. RENAME works differently. The Computer won't automatically assign an extension to a program you rename.

> ***Note:*** *There is one way to LOAD "BFILE" without an extension. This is by indicating that there is no extension by typing* `LOAD "BFILE/"` (ENTER). *This is awkward. That's why we suggest, when renaming a file, you always assign it an extension.*

Multi-Disk Drives

You can RENAME a file on another disk drive, simply by typing the appropriate drive number. Insert a formatted disk in drive 1 (if it's not already inserted). Store a file on it:

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

and RENAME it by typing:

```
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(0) (ENTER)
```

The Computer prints the number of FREE "granules" remaining on your disk.

There are 68 granules in all. If the Computer tells you that you have only one granule FREE, you'd better do one of the following: start using another disk or "KILL" some of your disk files.

KILLing a disk file does just 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 FREE space remaining 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 KILL it. The Computer insists you type the *complete* filename as one extra precaution. It doesn't want to KILL a file you don't want destroyed.

> ***Note:*** *Want to get very technical? The data will still exist on the disk after you KILL a file. However, the Computer won't* know *it's there because KILL deletes all reference to it in the disk's directory. Therefore, you'll no longer be able to access the data and the Computer will be able to write over it with a new file.*

Multi-Disk Drives

You can use FREE and KILL on other disk drives, as you can with 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.

SPECIAL MULTI-DRIVE COMMANDS

In the rest of this chapter, we'll talk about two commands which you can use if you have a multi-drive system. If you don't have one, go on to "Chapter Checkpoint" at the end of this chapter.

The first one copies a disk file. You should, at this point, have a program file stored in the disk in drive 0 named "NEW/NAM." Make a COPY of it. Type:

```
COPY "NEW/NAM:0" TO "NEW/NAM:1" (ENTER).
```

If you want, you can rename the file when you copy it. For instance, `COPY "NEW/NAM:1" TO "ANOTHER/NAM:0"` (ENTER) copies "NEW/NAM" from the disk in drive 1 to the file "ANOTHER/NAM" on the disk in drive 0.

The second command changes the drive number the Computer goes to if you do not specify one. Up to now, this has been drive 0. For example, by typing `SAVE "ANYTHING/EX"` (ENTER), the Computer will assume you want to use drive 0. It will then SAVE this 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 changing this DRIVE assumption, the Computer will respond differently to the same command. By typing `SAVE "ANYTHING/EX"` (ENTER), the Computer will store "ANYTHING/EX" on the disk in drive 1. You would now need to type `SAVE "ANYTHING/EX:0"` (ENTER) to SAVE it 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 more than one disk drive and do not specify the drive number, which drive will 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.

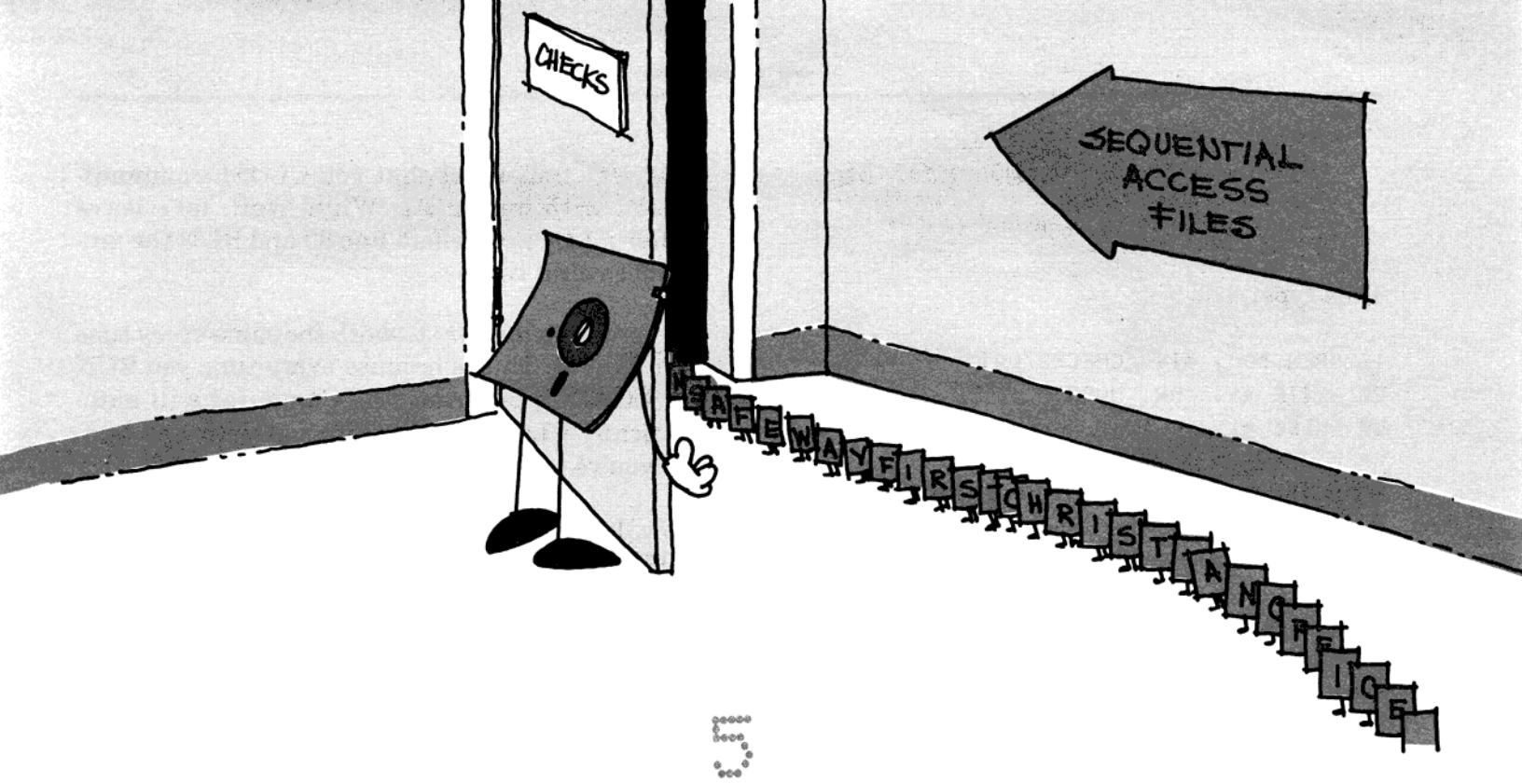
SECTION II

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 will enjoy writing your own.

In this section, we'll show you how to write a BASIC program which stores data on disk. We are assuming you already know some BASIC. If you don't, read *Section I* of *Getting Started with Color BASIC*. It will give you all the background you need.



ONE THING AT A TIME
(Sequential Access to a File)

A tape is simple. There's only one way to put data on it and one way to read it off. A disk is more complex. There are several ways to "file" your data on it.

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

In showing how to store things on disk, we'll 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 will index the location of the disk file in the *disk's directory.* For example, if you want to store the names of your friends, you could put them in a *disk file* named "FRIENDS." Your *disk's directory* would then index 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 will be 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'll start with a short, simple program which writes the first check, "DR. HORN," on the disk. Insert a formatted disk in your disk drive. (If you have more than one disk drive, 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.)* 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'll hear the motor of the disk drive and see the red light. The Computer is at work doing several tasks.

First, it OPENs communication to the disk so you can send your checks out to it. Then, it finds an empty location 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 the "O", #1, and "CHECKS/DAT":

1. #1 is a special "buffer" area in memory called buffer #1. It communicates with the disk drive. Line 10 OPENs this buffer. (If you've been using tape, you might remember that buffer # – 1 communicates with the tape recorder.)

2. "O" is the letter "O," not a zero. It stands for *output*. It tells the Computer that buffer #1 will be sending *out* data to the disk.

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

In line 20, the Computer sends out the words "DR. HORN" to buffer #1 which WRITEs it on the disk.

Then, in line 30, the Computer CLOSEs communication with buffer #1. In doing this, it:
- sends *out* all the data remaining in buffer #1 to the disk file.
- notes in the disk's directory where "CHECKS/DAT" ends.

> ***Note:*** *A buffer temporarily stores data so the Computer can input and output data to the disk in blocks of 249 characters (bytes). Since buffer #1 only contains 8 characters ("DR. HORN"), they would not be sent out to the disk without closing the file.*

It is very 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 will automatically CLOSE communication with any buffers you've left OPENed.

Now, let's assume you switch disks and RUN or LOAD a program. The Computer will automatically CLOSE communication with buffer #1. In doing this, it will send out its closing information to the new disk (thinking it's the old one). This will very possibly garble 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.*

You can verify that the Computer has done this by checking the disk's directory. You remember how to do that. (Type `DIR` (ENTER))

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

READING THE DISK FILE

To get the Computer to read this data from the disk back into its 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
```

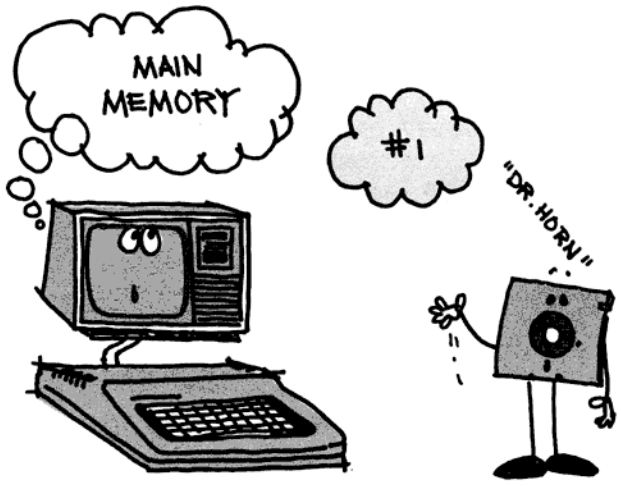
This is actually just the reverse of the *output* program...

Line 100 again OPENs communication to buffer #1. This time communication is OPEN for "I" — *input*. 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$. Line 120 PRINTs A$.

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

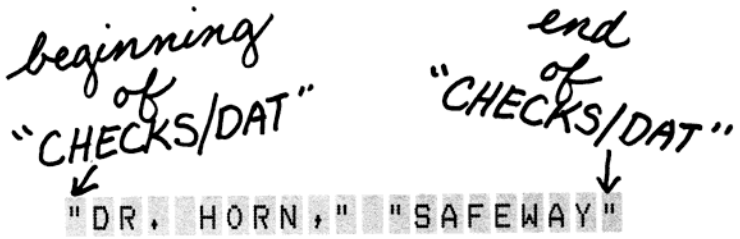
At this point, we've used an *output* program and an *input* program. Let's combine them into one program. Type:

```
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
```

Now 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 line 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.

This 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... EOF checks to see if you've reached the end of buffer #1 (the number in parentheses). If you have, EOF(1) equals a −1. If you haven't, EOF equals 0.

By adding 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 more details:

CHECKS		
PAYABLE TO	AMOUNT	EXPENSE
DR. HORN	45.78	MEDICAL
SAFEWAY	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 it.

A GOOD TIGHT PROGRAM

What if you need to store a whole list of checks? Continue to plod along with this program, and it'll soon be unbearable.

Here, we have a tight program which asks you to INPUT all your data, stores it on disk, and reads it back into memory. Erase memory and 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 it. Input any checks. When you want to quit, simply press (ENTER) in answer to the `CHECK PAYABLE TO :` prompt. For example:

```
CHECK PAYABLE TO :? GOODY BANK (ENTER)
AMOUNT :$? 230.97 (ENTER)
EXPENSE :? CAR (ENTER)
CHECK PAYABLE TO :? (ENTER)

YOUR CHECKS ARE STORED ON DISK
PRESS <ENTER> TO READ THEM? (ENTER)
GOODY BANK 230.97 CAR
```

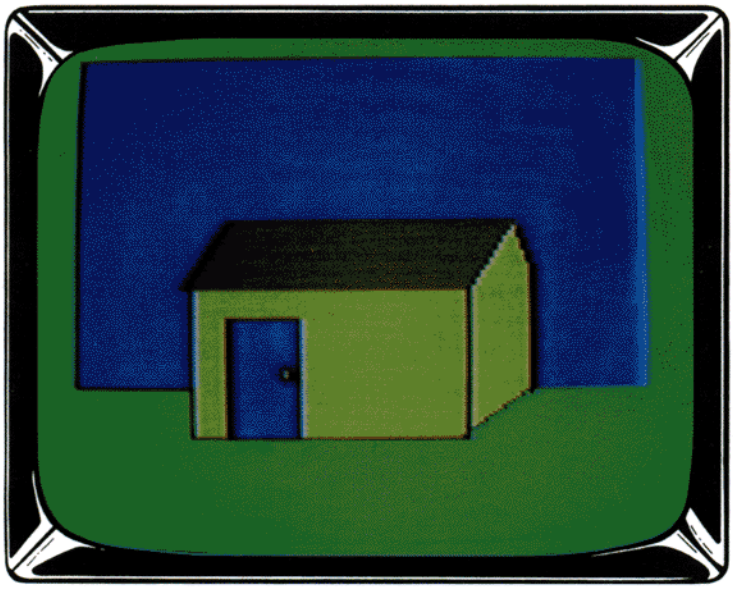
PROGRAMMING EXERCISE #5.1

Write a program which will print only those checks which were for CAR expenses.

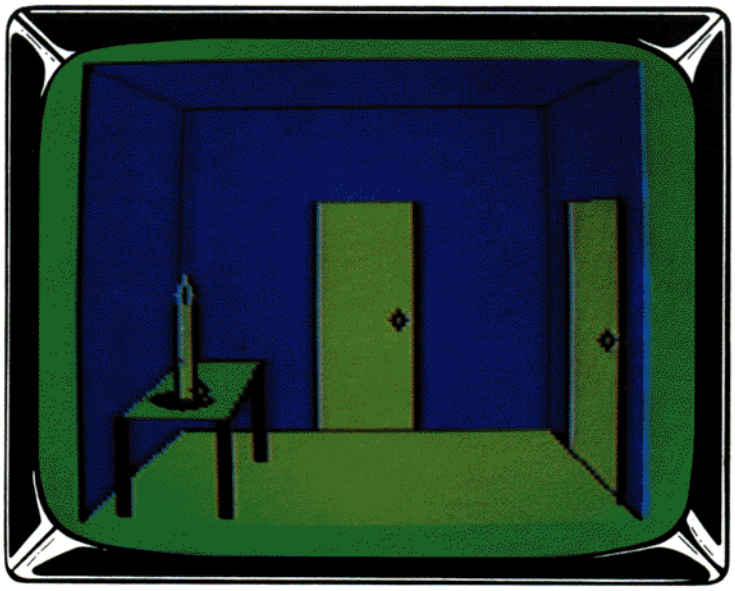
The answers to all the "Programming Exercises" are in Appendix A.

☑ CHAPTER CHECKPOINT

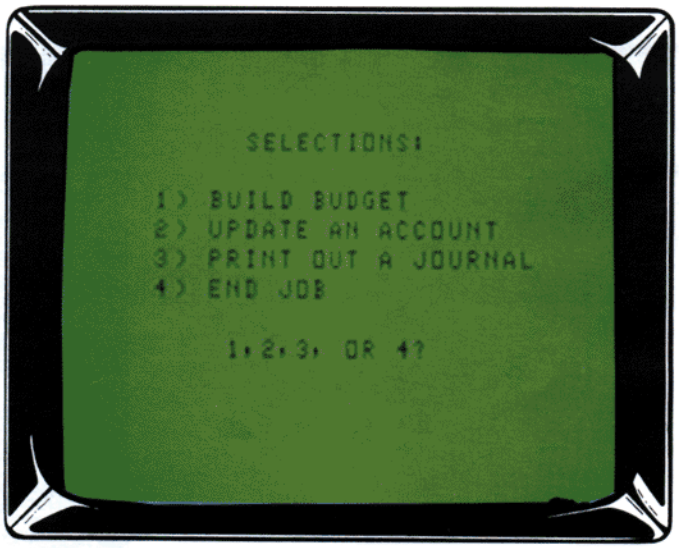
1. ***What is a buffer #1?***
2. ***Why must you OPEN a disk file?***
3. ***Why must you CLOSE it?***
4. ***What is the difference between a file OPEN for input and output?***

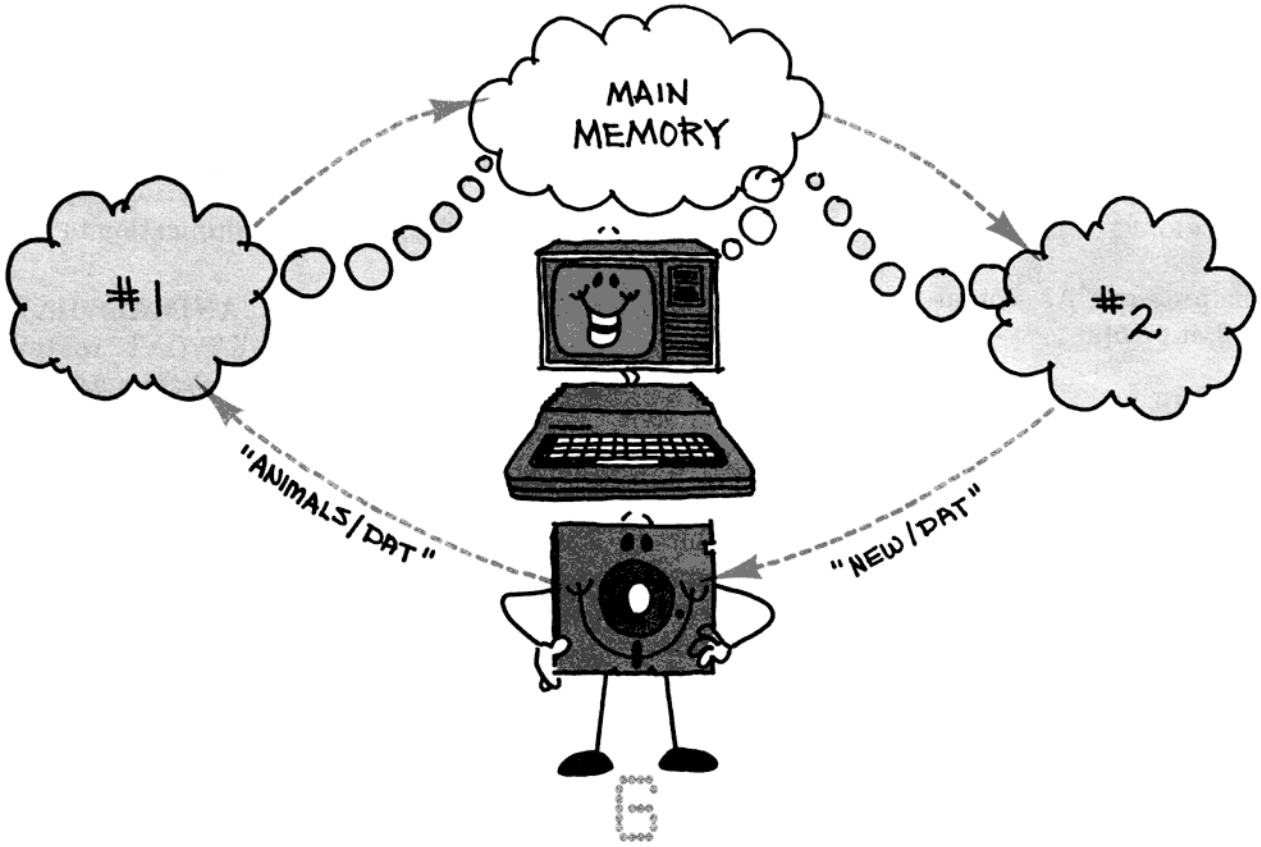


Try saving many different graphics programs on disk and calling them from one main program. Sample Program 7 in Appendix C *shows how.*



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





CHANGING IT ALL AROUND
(Updating a Sequential Access File)

Everything you put on the disk and take off of it goes through a spot in memory called a buffer. When we told you how to put data on tape in *Getting Started With Color BASIC*, we didn't talk about these buffers. We didn't need to. There is only one buffer which communicates with the tape recorder — buffer # – 1.

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

The reason we brought this subject up is that we want to demonstrate how to change some of the data in your file. To do this, it is very helpful to use two buffers.

> ***Note:*** *In* Chapter 10, *we'll demonstrate how to take advantage of more of these 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. Type NEW (ENTER) and then type:

```
10 OPEN "I", #1, "ANIMALS/DAT"
20 IF EOF(1) = -1 THEN 110
30 INPUT #1, A$
40 CLS : PRINT @ 106, "DATA ITEM :" A$;
100 GOTO 20
110 CLOSE #1
```

Then you need to add lines which will allow you to 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 for output. Add these lines:

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

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

```
10 OPEN "I", #1, "ANIMALS/DAT"
15 OPEN "O", #2, "NEW/DAT"
20 IF EOF(1) = -1 THEN 110
30 INPUT #1, A$
40 CLS : PRINT @ 106, "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$
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$, which will replace A$. If you input X$, line 90 *outputs* it. Line 90 *outputs* X$ to buffer #2, which, in turn, WRITEs it to "NEW/DAT."

Line 110 CLOSEs communication to buffer #1 and line 120 CLOSEs communication to #2.

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

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

Now the old "ANIMALS/DAT" file is deleted from the disk and the "NEW/DAT" file has been renamed to "ANIMALS/DAT." To see what this updated file contains, SAVE this program if you want, erase memory, and type and RUN:

```
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
```

Understand? Try these exercises:

PROGRAMMING EXERCISE #6.1

Write a program which will allow 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 which will allow you to delete animals from "ANIMALS/DAT."

Ready for the big time? Our next exercise is a program many of you will want — a mailing list program. We'll start you out with these lines which 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 it by solving this Programming Exercise. It'll be 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:

1. ***See the names, addresses, and phone numbers of your club's members.***
2. ***Change the addresses of some of the members.***
3. ***Add new members.***
4. ***Delete some of the members.***

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

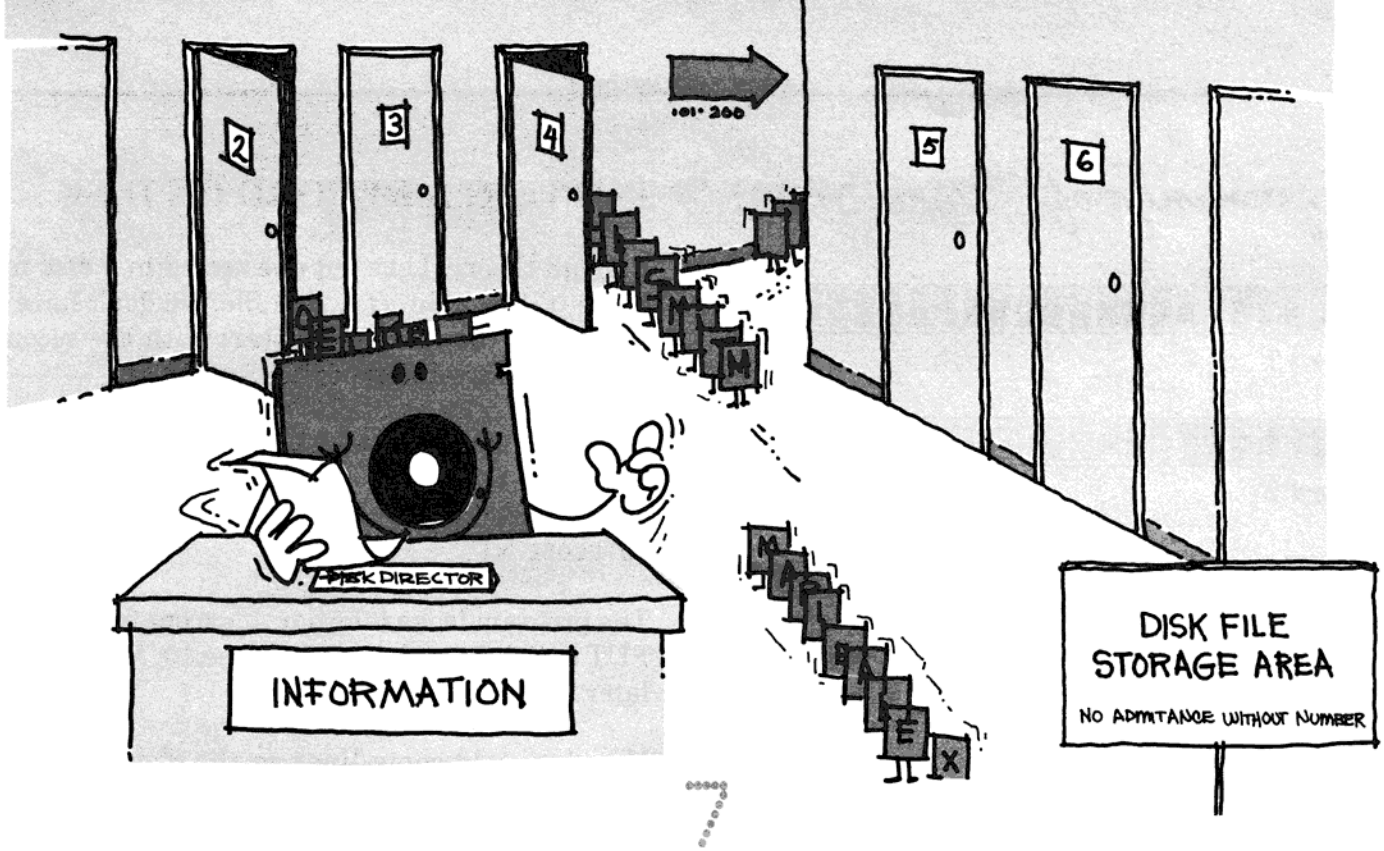
The process would still be the same. You would have to input each of the 500 members from one file and then output them all to another file. All of this just to change one record. There must be an easier way!

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 will make them take up more space in your disk. The choice is yours. We'll talk about direct access in the next chapter.

Note: *We've demonstrated short example programs. 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 and output data to the same buffer at the same time?***
2. ***Can you input data from a file OPENed for "O"—output?***



A MORE DIRECT APPROACH
(Direct Access to a File)

Up to now, we haven't been concerned with how your data is stored on the disk. For example, you might have put this in a disk file:

beginning of "NAMES/DAT"

```
"MARIE ALEXANDER," "J. DO
E," "MARK JONES," "BILL S
MITH"
```

end of "NAMES/DAT"

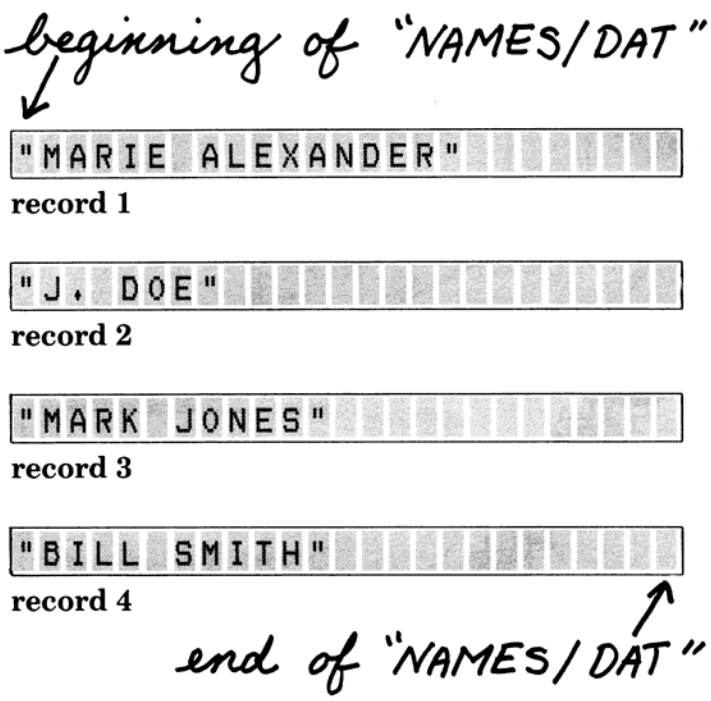
What if you want to change "J. DOE," to "ELLIOTT HOBBS"? You could not ask the Computer to go directly to "J. DOE." The Computer does not know where it is.

All the files we've created so far have been "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 talked 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 something which we call "records." You can then write a program which stores each record in a sector and allows you to put data in the records. The next page shows how your new disk file will look:



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

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

A direct access file has one shortcoming. Each record is the size of a sector—256 bytes. Since one of these bytes holds one character of data, each record is large enough to hold 256 characters.

This means that our drawing above is a little misleading. If we illustrated all the empty space in each record, they would each have to be nearly ten times as long. We simply don't have enough room on the page.

If you're a beginner, all this empty space probably won't bother you. An empty disk can hold up to 612 records—each 256 bytes long. Later on, when you become more comfortable with programming, you'll probably want to pack more records into a disk file. You can then progress to *Chapter 9*, where we will demonstrate how to make smaller records.

PUTTING A RECORD ON DISK

Enough theory! Let's put one record in a disk file. Since it'll be a direct access file, we don't have to start with the first. We'll 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. More on that later . . .

Now let's add some lines so the Computer will read 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? Let's 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'll 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 . . .

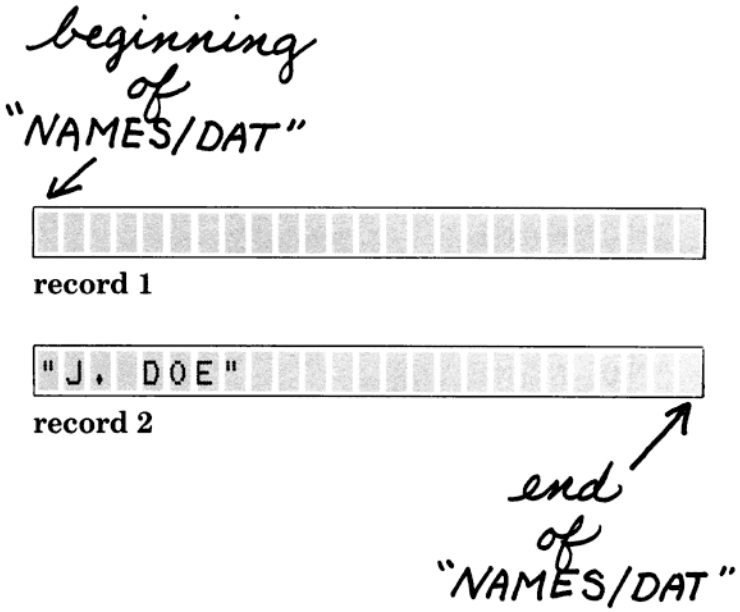
Line 10 OPENs buffer #1 which will communicate with a disk file named "NAMES/DAT." As we said in the last two chapters, buffer #1 is one of the 15 "buffer" areas which can communicate with your disk.

Communication is being OPENed for "D." "D" stands for direct access. Unlike 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" will remain 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 did this 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 our drawing shows only two records in the file. GET record 4. Type:

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

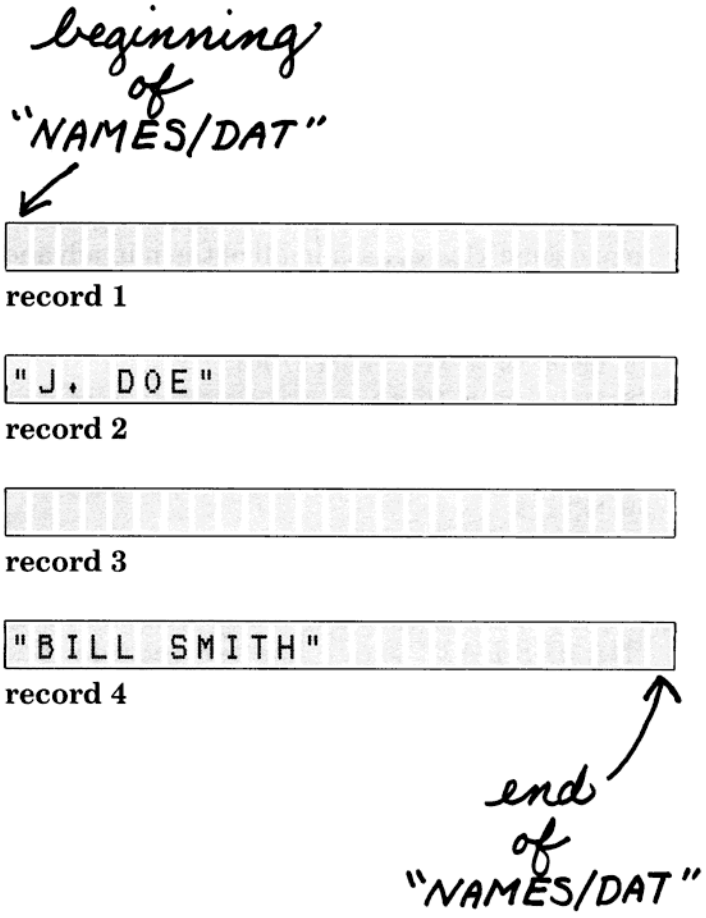
and RUN... The Computer gives you an IE (Input past the End of the File) error. This is because the last record the program PUT in the file was record number 2. Hence, record 2 became the end of the file.

> ***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 will have these four records:



PROGRAMMING EXERCISE 7.1

Change lines 32 and 34 so that your Computer will use 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 and RUN:

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

Since 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, it will either get the "garbage" or give you an OS (Out of String Space) error. The OS error simply means the garbage consumes more than 200 bytes (characters).

Since your empty records will 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":

beginning of "NAMES/DAT"

record 1

record 2

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 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$ "-- IS THE NAME IN RECORD" R
```

```
80 INPUT "TYPE NEW NAME ELSE PRESS
   <ENTER>"; A$
90 IF A$ = "" THEN 20
100 WRITE #1, A$
110 PUT #1, R
120 GOTO 20
130 CLOSE #1
```

RUN it. 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, type a 0 as the RECORD NO.)

READING ALL THE RECORDS

At this point, you might like the Computer to print all of the records in your "NAMES/DAT" file with their appropriate record numbers. SAVE your program, if you want, erase memory, type, and RUN:

```
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 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 will not 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 which buffer #1 (the number in parenthesis) is communicating with. It tells the Computer what the last record number in that file is.

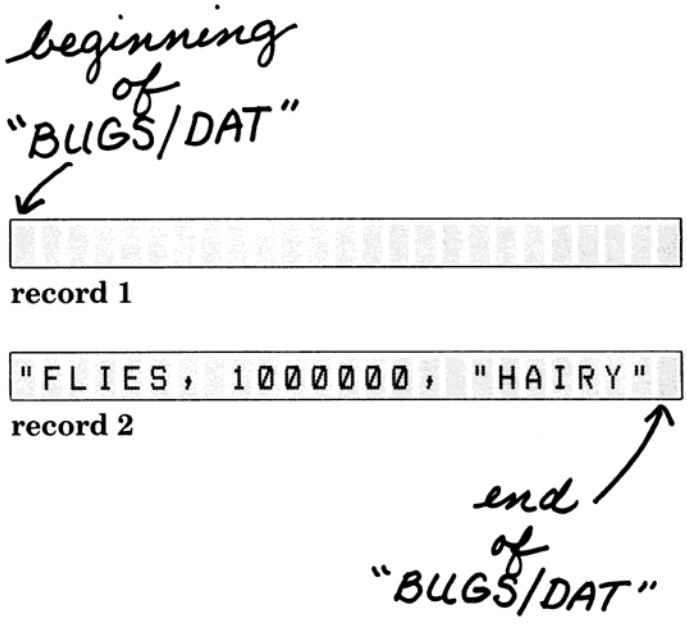
MORE POWER TO A RECORD

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

Erase memory, 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:



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 and RUN...

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

Since this line asks the Computer to INPUT only the first field of data in buffer #1, it INPUTs only "FLIES."

PROGRAMMING EXERCISE 7.2

What do you think the Computer would print if you ran the program, using this for line 36? Why?

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

PROGRAMMING EXERCISE 7.3

Change the program which stores the "NAMES/DAT" file so that each record will contain five fields of data:

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

☑ *CHAPTER CHECKPOINT*

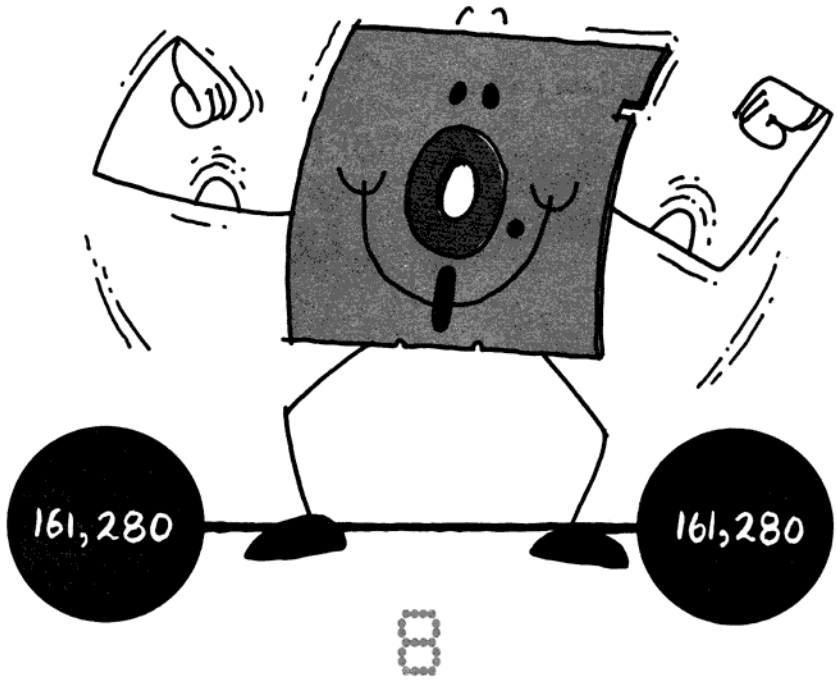
1. ***What are records? Why must you use them to access data directly?***
2. ***What are fields?***
3. ***What is the difference between a sequential access and a direct access file?***
4. ***Why is it quicker to update a direct access file?***

SECTION III

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 all of you ambitious people 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.



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." One of these bytes can hold one character. Thus, the word STRAW will consume five bytes of disk space.

An empty disk contains 161,280 bytes. 4,608 of them house the directory. This leaves you 156,672 for your disk files.

> ***Note:*** *A disk contains 35 tracks. Each track contains 18 256-byte sectors, or 18 x 256 = 4,608 bytes. One of the tracks is for the directory. This leaves 156,672 bytes (4,608 bytes per track x 34 tracks).*

Does this mean you can use the entire 156,672 bytes for data? Possibly. There are two factors which will determine this.

The first 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 will contain a multiple of 2,304 bytes. If your file contains 2,305 bytes of data, for example, the Computer will allocate 2 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 which affects how much data 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.

In the next two chapters, we're going to compare eight different types of programs. Each will store the same data — 5, "PEN," −16, and "PAPER" — in a disk file named "OFFICE/DAT." The amount of overhead and empty space each program will put in "OFFICE/DAT" will vary greatly.

WRITING ON THE DISK

Program 1 uses WRITE to put this 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 what lines 20 and 30 wrote on your disk. Type these two lines exactly as they are above, but leave off the #1 in each line. This will prevent the Computer from writing the data on your disk (via buffer #1). The Computer will write it on your screen instead. Type:

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

Look very carefully at what the Computer WRITEs. Every blank space and punctuation mark counts.

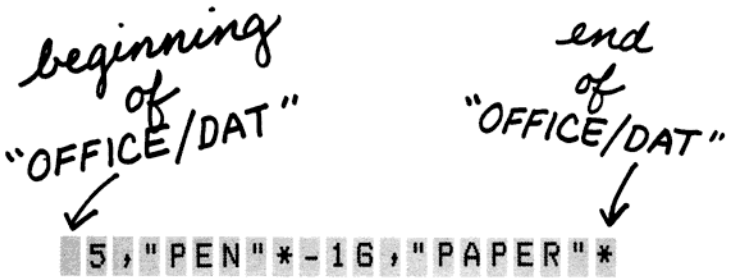
Notice the way the Computer WRITEs the two strings (PEN and PAPER). It puts quotation marks around them. It WRITEs the numbers (5 and -16) differently. If the number's negative, the Computer puts a minus sign in front of it. If it's positive, the Computer simply puts a blank space in front of it.

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

```
 5, "PEN"
OK

-16, "PAPER"
OK
```

When writing on the disk, the Computer actually WRITEs each (ENTER) character exactly as you typed it. This illustration shows what Program 1 WRITEs on your disk. (We used asterisks to 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 (see* Appendix D*) which the Computer converts to a binary number.*

Count the characters. Make each (ENTER) (represented by an asterisk), comma, and quotation mark count for one character each. Don't forget the blank space preceding 5. What you should come up with is 21 characters. Program 1 puts 21 bytes in "OFFICE/DAT."

Since the Computer allocates file space in clusters, "OFFICE/DAT" will actually consume 1 granule of disk space or 2,304 bytes. However, for the purpose of comparison, we'll only look at the 21 bytes which 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
```

It did input your data items. However, it did not input the quotation marks, commas, and blank spaces which we told you were interspersed with your data.

To actually see what Program 1 wrote on your disk, you can use a "LINE INPUT Program." First

SAVE the "INPUT Program" you now have in memory. (You'll be using it later.)

Now change it into a "LINE INPUT Program." Delete line 50 and change lines 40 and 60. Type:

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

and RUN ... 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 it on your screen.

The program then INPUTs and PRINTs −16, "PAPER" — the second and final line in the file.

We can easily alter this program so that it will count how many bytes are in the file. Add these lines and RUN it:

```
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 total number of characters (bytes) 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 it. It will be useful in comparing what Programs 2, 3, and 4 put in your disk file.

PRINT — FOR A CHANGE

So far, we've used 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 do this. However, PRINT is much more tricky to use. If you're not used to it, don't bother learning all this. Skip to Program 4.

... Still with us? KILL 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'd 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 which will send your data to the disk file. To see what Program 2 PRINTs, type:

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

Notice the Computer did not enclose the strings — PEN and PAPER — in quotes, as WRITE did. This will be important to know later.

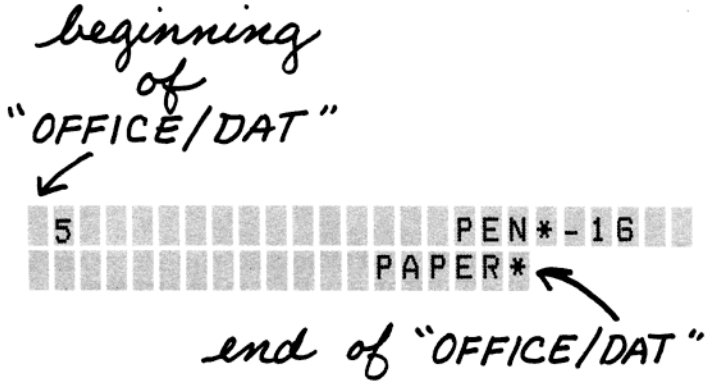
Now look at the blank spaces. We'll start with the first one — the one before the 5. This means the

same thing it did with WRITE. 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 −16.

How about all the additional spaces? Remember, from *Getting Started With Color BASIC,* what a comma in the PRINT line does? It causes the Computer to PRINT your data in columns, inserting spaces between the columns.

The Computer will PRINT every single one of these blank spaces in your disk file:



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

> ***Note:*** *Unclear about what commas do 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 PRINT inserts in your disk file are a waste of space. They are. The way to get around this waste is to use semi-colons. You might again recall, from *Getting Started With Color BASIC,* that semi-colons in a PRINT line compress your data. Type:

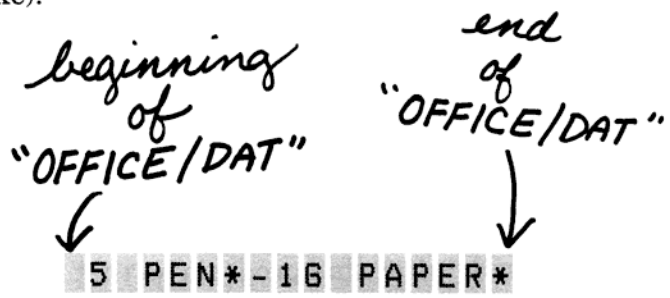
```
PRINT 5; "PEN" (ENTER)
PRINT -16; "PAPER" (ENTER)
```

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

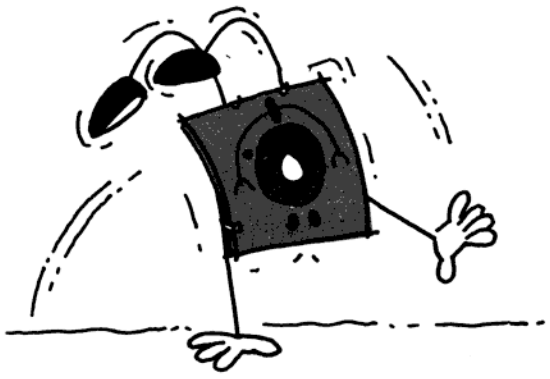
PROGRAM 3
17 bytes

```
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. 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

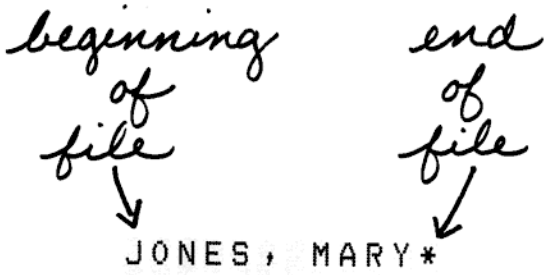
There are certain types of PRINT lines which are tricky. (We did warn you, didn't we?) Type:

```
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:

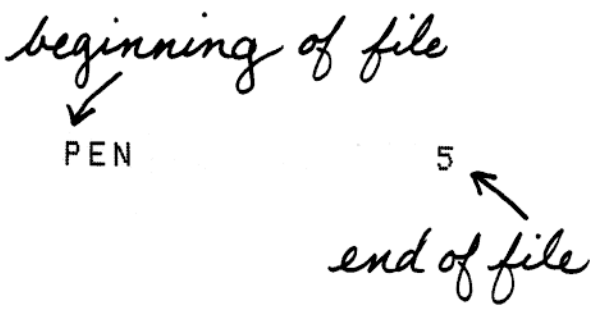
The Computer would read PENPAPER back into memory as one item. *(Reason: there is not a "delimeter" — a comma, quotation mark, or space — to separate PEN from PAPER).*

The line `PRINT #1, "JONES, MARY"` would print this in your disk file:



The Computer would read JONES, MARY back as two items: JONES and MARY. *(Reason: The Computer interprets the comma as a delimeter).*

The line `PRINT #1, "PEN", 5,` would print this in your disk file:



The Computer would read PEN 5 (with all the blank spaces) back into memory as one item. *(Reason: although the Computer normally interprets blank spaces as a delimeter, it will not interpret them in this way when they follow a string and precede a number).*

For more information on using PRINT in disk programs, see the *TRS-80 Model I, Model II,* or *Model III Disk System Owner's Manual.*

AN ATTRACTIVE DISK FILE

PRINT USING is another word you can substitute for WRITE. We discussed PRINT USING in *Going Ahead With Extended Color BASIC).* Type:

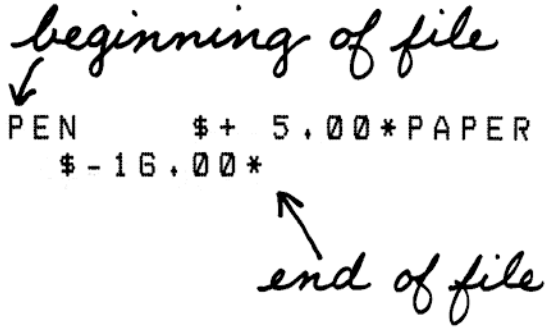
```
PRINT USING "%      %$+##.##"; "PEN", 5
  ENTER
PRINT USING "%      %$+##.##"; "PAPER", -16
  ENTER
```

You can get the Computer to print these same images on your disk with this program. KILL "OFFICE/DAT," erase memory, and 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
```

which prints this in your disk file:



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

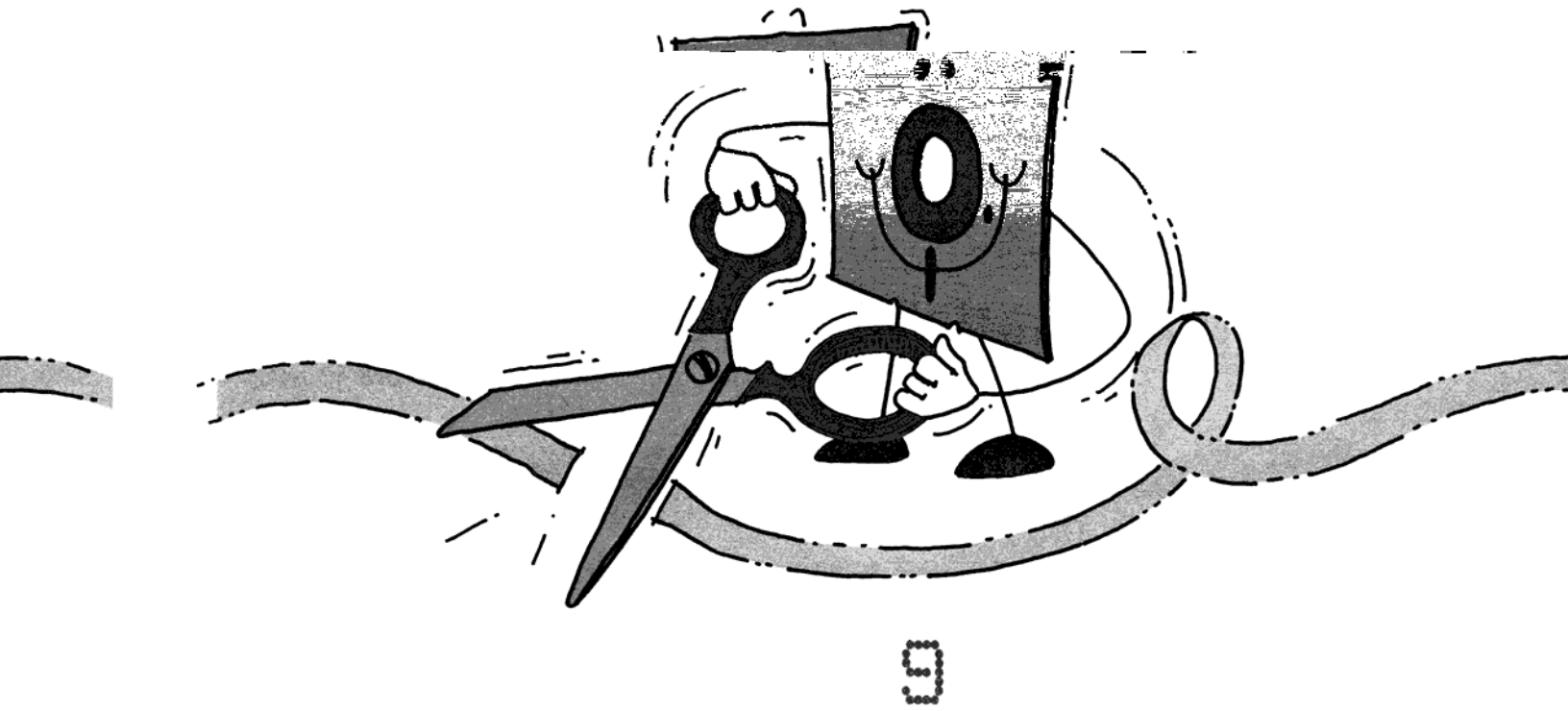
Now the data is already in an attractive print format. You can input and print it using a simple line input program. Erase memory, type and RUN:

```
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 of the files we've created in this chapter are sequential access. The next chapter compares four more programs which will 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?***
4. ***What is the difference betw… LINE INPUT?***
5. ***What does a comma in a PR… the Computer to do?***
6. ***What does a semi-colon in… cause it to do?***
7. ***How does the Computer PR…***



9

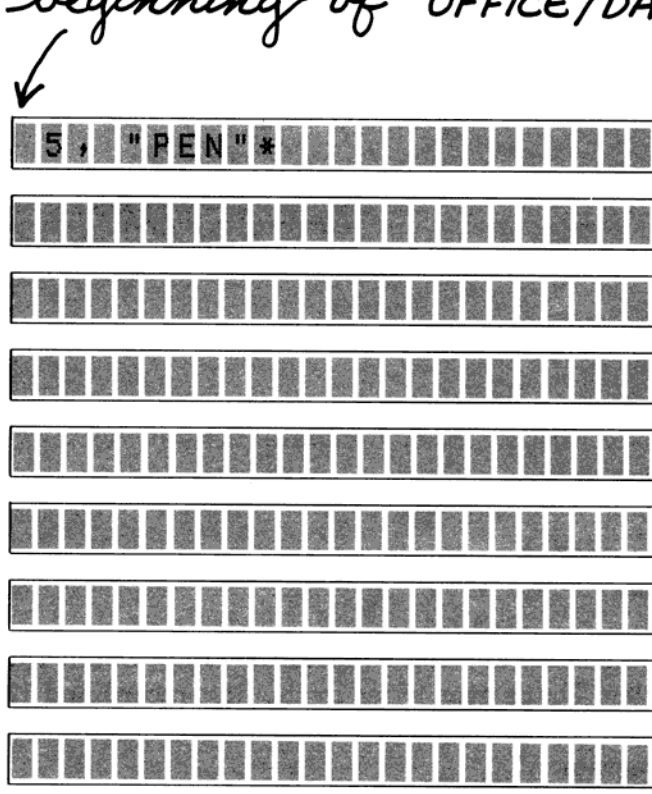
TRIMMING THE FAT OUT OF DIRECT ACCESS
(Formatting a Direct Access File)

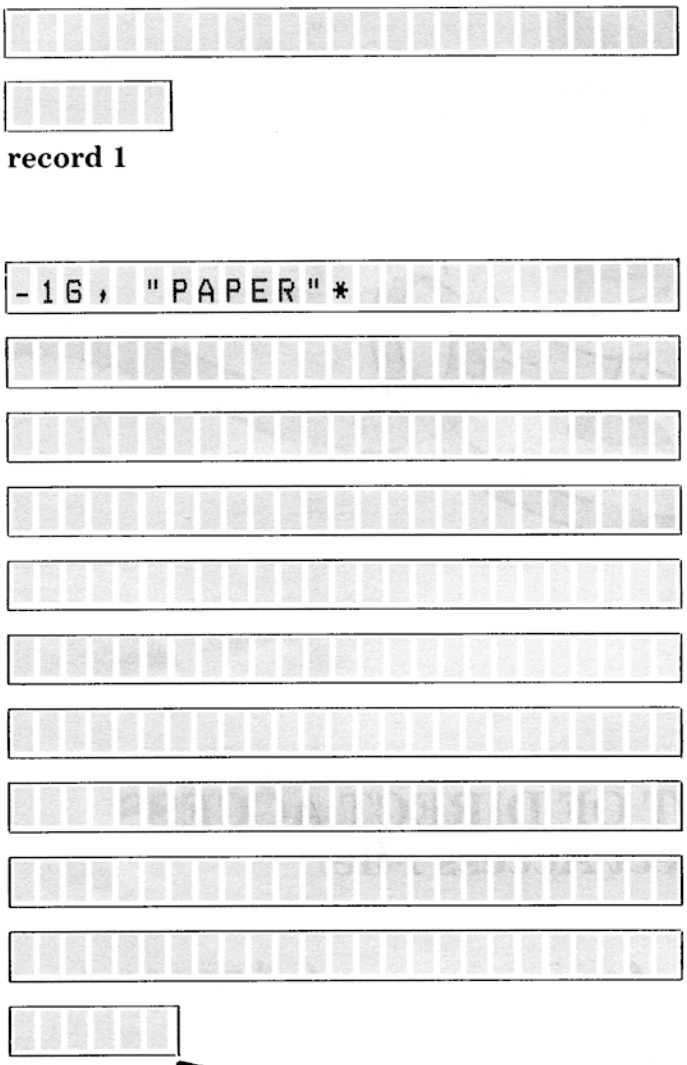
Direct access files often contain a lot of empty space. For example, our first program is very similar to Program 1 from the last chapter. The WRITE lines are identical. However, because it is direct access, it will put 512 bytes in "OFFICE/DAT":

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. Program 5 puts two records in the "OFFICE/DAT" file. Therefore, it will consume 2 x 256, or 512 bytes:





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

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

is the same as what Program 1 wrote. Count the bytes. That's nine bytes in the first record and 12 in the second. You'll need to know this for our next program.

Note: We could have used PRINT or PRINT USING rather than WRITE. The Computer would have then PRINTed your data inside each record using the PRINT or PRINT USING format.

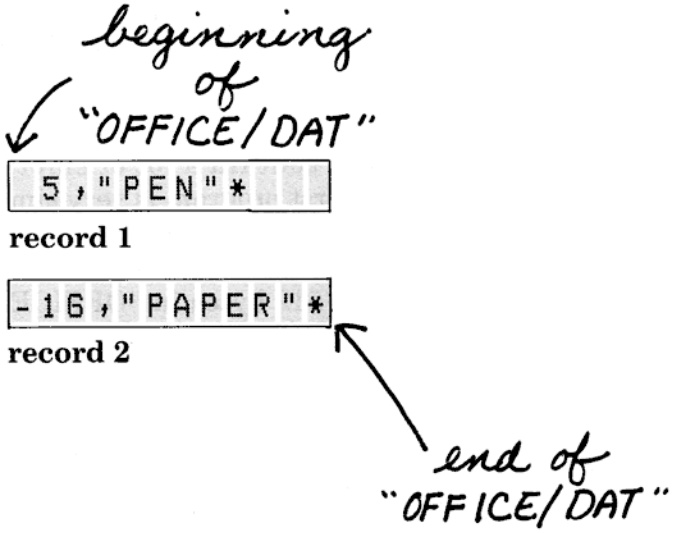
TRIMMING THE FAT

Program 6 is the same as Program 5, except that we inserted a number 12 at the end of line 10.This 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 really whittles this file down:



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 them, they will all be 256 bytes.

In this program, we made each record 12 bytes, the size of the largest record. Type and RUN Program 6, if you'd like. (Be sure to erase memory and KILL your old "OFFICE/DAT" file first.) After RUNning Program 6 you can use this 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 in a record which follow the (ENTER) character.*

EFFICIENCY, EFFICIENCY...

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

PROGRAM 7
16 bytes

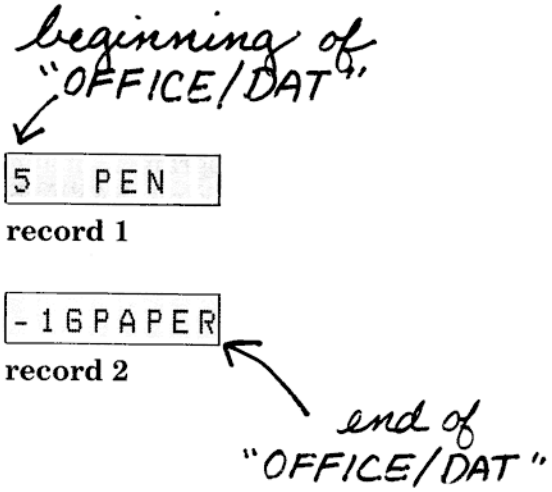
```
10 OPEN "D", #1, "OFFICE/DAT", 8
20 FIELD #1, 3 AS A$, 5 AS B$
30 LSET A$ = "5"
40 LSET B$ = "PEN"
50 PUT #1, 1
60 LSET A$ = "-16"
70 LSET B$ = "PAPER"
80 PUT #1, 2
90 CLOSE #1
```

There are two new words in this program which we'll talk about later. Let's see what the program does first. SAVE it. Then erase memory and input the file with this program:

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, your program will work the same as any direct access program. The difference is what FIELD and LSET put in each record:



Only the bare essentials. 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$. These two fields will be the same size in every record. A$ will always be 3 bytes and B$ will always be 5 bytes.

Now that we've established this, we can put data in each field. Line 30 LSETs 5 in the A$ field (SETs the character 5 to the Left of A$). Since the character 5 only consumes 1 byte and there are 3 bytes in the A$ field, there are 2 empty spaces at the end of 5.

Notice we had to convert the number 5 to a string by putting quotes around it. You cannot LSET a *number*. You must convert it to a *string*.

Line 40 LSETs the word PEN in the B$ field. Again, this leaves 2 empty spaces at the end of A$, since PEN is 3 bytes.

Line 50 PUTs all this in record 1. Then, the same process is repeated for record 2.

Now let's look at the "Fielded INPUT Program." Notice we used a FIELD line. RUN the program without line 20 and see what happens...

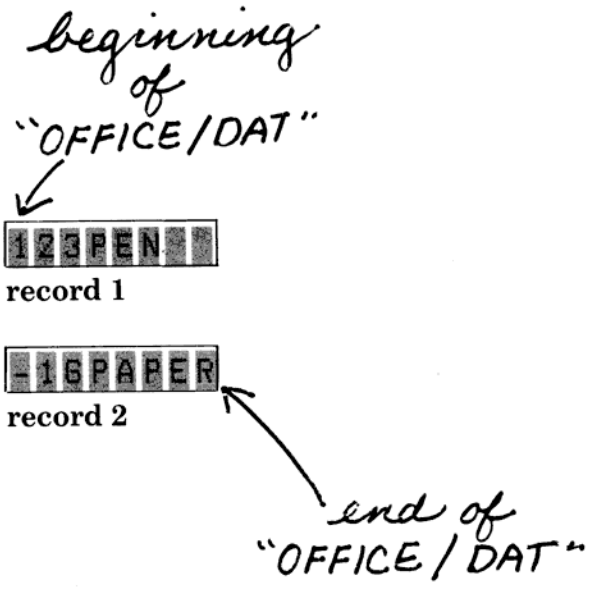
Without a FIELD line, the Computer does not know where the two fields are. Whenever you input FIELDed records, use a FIELD line in your input program.

Can you guess what the Computer would do if you tried to LSET a long string, such as "123456789," into one of the fields? LOAD Program 7, change line 30, and RUN the program. (First, SAVE the "Fielded INPUT Program" with line 20 reinstated.):

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

Now load and RUN the "Fielded INPUT Program."

A$ is only 3 bytes. Therefore, the Computer only LSETs the first 3 bytes of "123456789." It chops the remaining characters off:



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 will be putting a lot of numbers in your disk file. Every number might be a different length:

$$-5.237632 \quad 31 \quad 673285$$

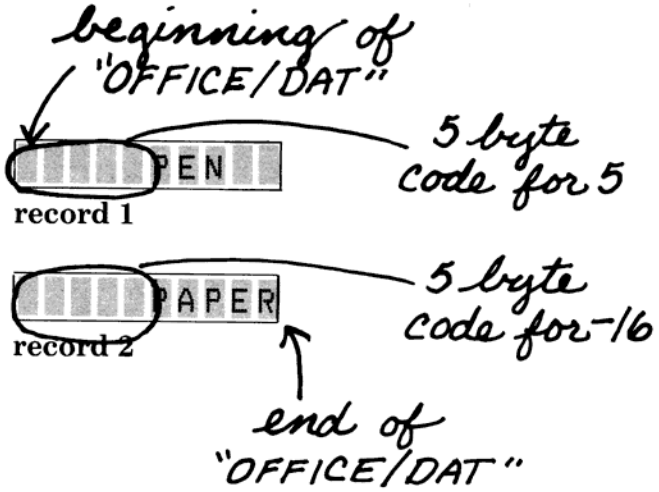
However, it is very important that the Computer not chop any of the digits off. This might entirely change the number's value.

The word MKN$ will solve this problem:

PROGRAM 8
20 bytes

```
10 OPEN "D", #1, "OFFICE/DAT", 10
20 FIELD #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 difference between this program and program 7 is lines 10, 20, 30, and 60. This is what it stores in your disk file:

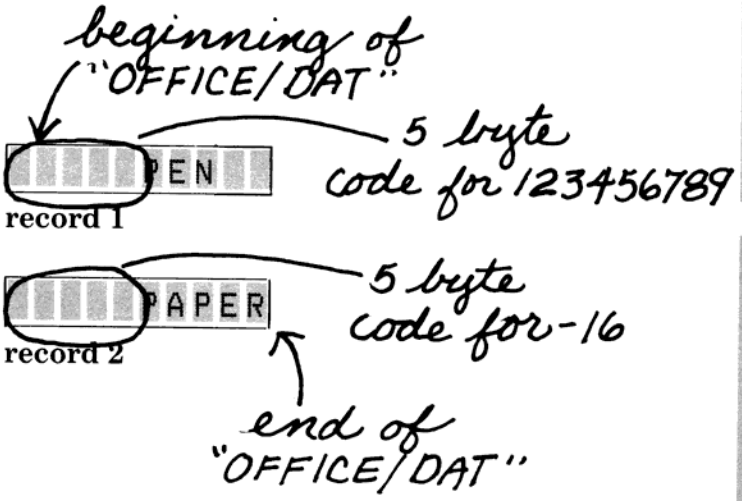


MKN$ converts a number to a coded string. Regardless of how long the number is, MKN$ will always convert it to a string that is five bytes long.

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

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

Erase memory, KILL "OFFICE/DAT," and type and RUN the program. This is what it stores in your disk file:



To read this program in, 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$); B$
```

and RUN it... CVN (in line 50) decodes A$ to the number it represents.

> ***Note:*** *The Computer only sees the first 9 digits of a number. It rounds the rest off.*

PROGRAMMING EXERCISE #9.3

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

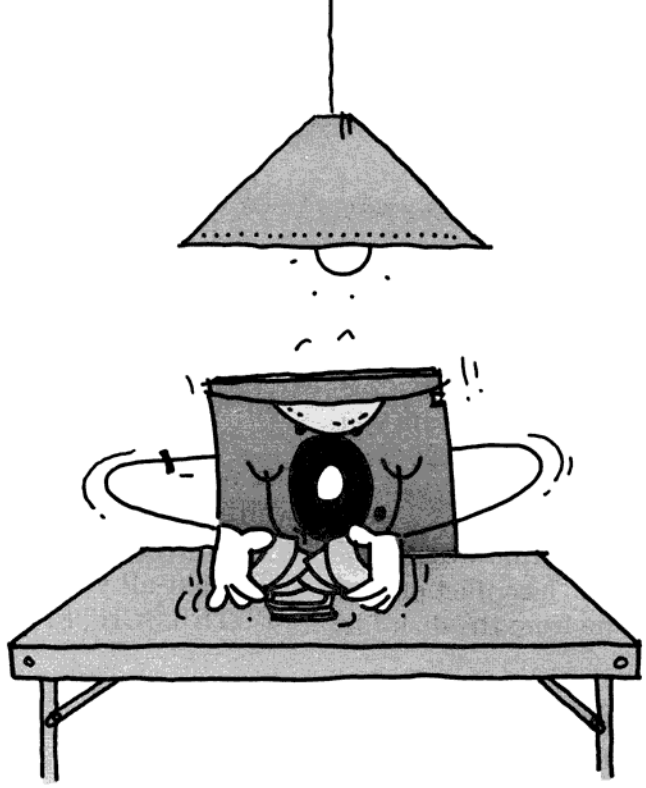
1. country — 10 bytes
2. population — 5 bytes

PROGRAMMING EXERCISE #9.4

Write a program which will input the file you created in Exercise #9.3.

☑ *CHAPTER CHECKPOINT*

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



10

SHUFFLING DISK FILES
(Merging programs, using many buffers)

Because storing and retrieving disk files is so easy, you will want to use them as much as you can. In this chapter, we're going to 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 program files with whatever 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'll explain why later... Erase memory. Now put this program in memory:

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

and combine it with one of the programs you SAVEd. Type:

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

LIST the program... The Computer has MERGEd "MONTHS/AGE" with the program you have in memory. Notice the line numbers are the same as they were in each individual program.

At this point, this is the program you have in memory:

```
5 INPUT "TYPE YOUR AGE"; N
10 REM AGE CONVERSION TO MONTHS
20 N = N * 12
```

```
30 A$ = STR$(N) + " MONTHS"
40 PRINT "YOU HAVE LIVED ABOUT" A$
```

MERGE "WEEKS/AGE" with it by typing `MERGE "WEEKS/AGE"` (ENTER). Then LIST the MERGEd program.

Notice that lines 10, 20, and 30 of the program you had in memory were 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 (two line 10s), the line from the disk file prevails.

Now we'll 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, it writes the BASIC words differently. To save space, it compresses each BASIC word into a one-byte "binary" code.

You can't MERGE a file which contains these binary codes. This is why we had you type the A when you SAVEd the two programs above. The A tells the Computer to write the ASCII codes for each BASIC word rather than the binary code.

By checking the directory, you can see if the data in your files are in ASCII or binary codes. If there is an "A" in the fourth column, it's all in ASCII codes. A "B" indicates that some of the words are in binary codes.

> ***Note:*** *Try typing* `MERGE "MONTHS/AGE", R` (ENTER). *The R tells the Computer to RUN the program after it's MERGEd.*

USING MORE BUFFER SPACE

When you start-up your disk system, it sets aside two buffer areas in memory for disk communication. You can use either or both of them for reading or writing data to a disk file.

Up to now, that's all we've used — buffers #1 and #2. But, as we've said earlier, you can use up to 15 disk buffer areas.

To use more than 2 buffers, you must first reserve space in memory for them. To do this, use the word FILES. For example, FILES 3 reserves 3 buffers.

Making use of all these buffers will greatly simplify your programs. For example, let's assume you own a computer school. To organize it, you first put all your students in a file named "COMPUTER/SCH." Erase memory, 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 this program to assign the students to a BASIC or assembly-language class. Erase memory and type this "Class Assignment 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 120
60 INPUT #3, ST$
70 PRINT: PRINT ST$
80 INPUT "(1) BASIC OR (2) ASSEMBLY
   LANGUAGE"; R
90 IF R > 2 THEN 80
100 WRITE #R, ST$
110 GOTO 50
120 CLOSE #1
130 CLOSE #2
140 CLOSE #3
```

RUN it. After assigning all the students to a class, you can print a class roster with this program. Erase memory, 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: *Substitute "ASSEMBLY/CLS" for "BASIC/CLS" in lines 20 and 30 to print the class roster of the assembly language class.*

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 OPENs 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 ("STUDENT/SCH") have been input, line 50 sends the Computer to lines 120-140, which CLOSEs the three buffers.

CROWDING THE BUFFER

There's one more thing you'll like about FILES. Erase memory, type, and RUN:

```
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
```

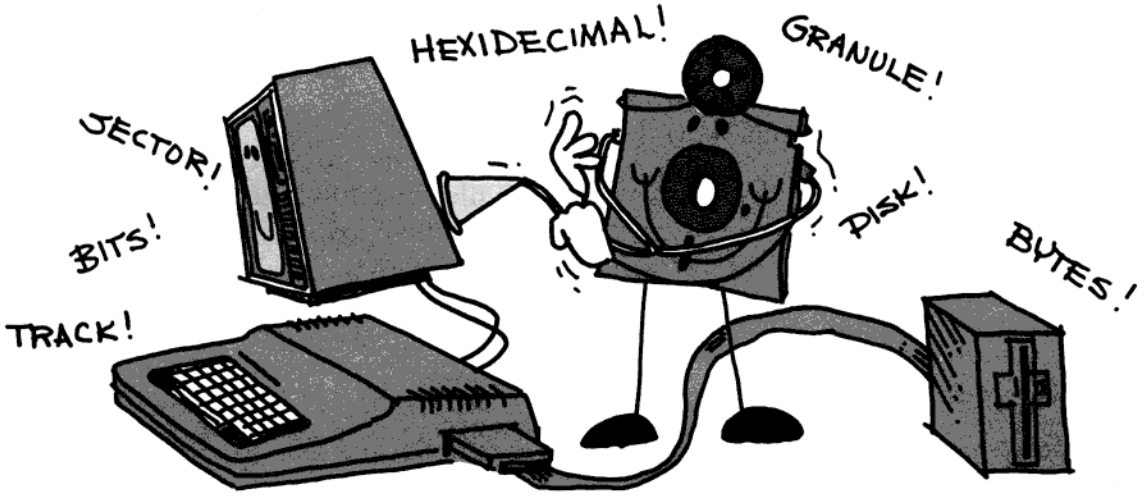
Want to input this paragraph? Add these lines and RUN:

```
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 which you will want to MERGE?***
2. ***When the two programs you're merging both have the same line numbers, which lines prevail?***
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?***



TECHNICAL INFORMATION (Machine-Language Input/Output)

In this chapter, we'll discuss the technical details which are happening "behind the scenes." You don't need to know this information when you are programming in BASIC. In fact, you won't even be aware that these details are happening.

However, if you plan to write machine-language disk programs or are simply interested in knowing all you can, you'll definitely want to read this chapter. We'll begin by discussing how the Computer organizes all the bytes on the disk. Then, we'll show 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 these bytes control the system. The great majority of them are for data.

Tracks

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

Byte #	Contents
0-31	System controls
32-6115	Sectors
6116-6249*	System controls

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

*the number of system control bytes at the end of each track might vary slightly due to slight speed variations.

Note: *One byte contains 8 bits. Each bit contains either a 1 or a 0. Normally, we express the contents of these 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 — 0100110. 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. 256 of them are for data. The remaining bytes are for system controls.

Byte #	Contents
0-55	System controls
56-311	Data
312-337	System controls

The hexadecimal contents of the system control bytes are:

Byte #	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 which 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 ½ 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 3 granules (6,912 bytes) of disk space for it.

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

Track 0, Sectors 1-9	—	Granule 0
Track 0, Sectors 10-18	—	Granule 1
Track 1, Sectors 1-9	—	Granule 2
•		•
•		•
•		•
Track 16, Sectors 10-18		Granule 33
Track 17, Sectors 1-18		Directory
Track 18, Sectors 1-9		Granule 34
•		•
•		•
•		•
Track 34, Sectors 10-18		Granule 67

Note: *The minimum size of a disk file is one granule or 2,304 bytes. A disk will hold 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 which 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 9 sectors of the directory containing directory entries (sectors 3-11) will hold up to 72 entries. Each entry is 32 bytes long and contains:

Byte #	Contents
0-7	Filename, left justified, blank-filled. If byte 0 = 0, the file has been deleted and the entry is available. If byte 0 = FF (hexadecimal), the entry (and all following entries) have not yet been used.
8-10	Filename extension, left justified, blank-filled.
11	File Type 0 = BASIC program 1 = BASIC data file 2 = Machine-language program 3 = Text Editor source file
12	ASCII flag 0 = the file is in binary format 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 on the first 68 bytes of the sector. The remaining bytes contain zeroes:

Byte #	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 will either contain a value of FF, 0-43, or C0-C9 (hexadecimal):

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, 10 is the next granule in the file.
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 that granule are part of the disk file. (Bits 7 and 8 both equal 1.)

SKIP FACTOR

The Computer reads or writes data to the disk one sector at a time. Between sector reads or writes, it does some processing.

The disk does not stop and wait for the Computer to do this processing. It spins continuously.

For example, the Computer might read Sector 1 first. But by the time it's finished processing Sector 1, the disk will have spun to Sector 6.

To allow for this time differential, the Computer sets a "skip factor" of 4 when it formats the disk. This notes on the disk that the computer should skip 4 "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

PHYSICAL SECTOR	LOGICAL SECTOR
13	7
14	18
15	11
16	4
17	15
18	8

Thus, after reading Sector 1, the Computer will skip "physical" sectors 2, 3, 4, and 5. The second "logical" sector it reads will be "physical" Sector 6.

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:

```
DSKINI0, 3
```

tells the Computer to skip 3 physical sectors between each logical sector.

> ***Note:*** *It's difficult to determine the optimum skip factor. We recommend you leave it 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 need to write instructions to the Color Computer's 6809 Microprocessor.

See "Using Machine-Language Subroutines" with Color BASIC in *Getting Started with Color BASIC* for the procedures to use in accessing a machine-language subroutine. See 6809 Assembly Language Programming, by Lance Leventhal (published by Osborne/McGraw-Hill) for the specific 6809 instructions.

Information on DSKCON

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

```
JSR [$C004]
```

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 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 to 3

DCTRK — Track Number
- 0 to 34

DCSEC — Sector Number
- 1 to 18

DCBPT — Buffer Pointer
- the address of a 256-byte buffer. For read sector, the data is returned in the buffer. For write sector, the data in the buffer is written on the disk.

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

DCSTA — Status
- Bit 7 = 1 Drive Not Ready
- Bit 6 = 1 Write Protect
- Bit 5 = 1 Write Fault
- Bit 4 = 1 Seek Error or Record Not Found
- Bit 3 = 1 CRC Error
- Bit 2 = 1 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

This program uses DSKCON to restore the head to track 0:

```
R 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 through 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 DISPLAY
  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 (its starting and ending addresses). You also need to specify the address where it should be executed. Use the hexadecimal numbers for all of these addresses.

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

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

To load it back into memory, you could use the LOADM command:

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

This would load "PROG MAC" back into memory locations 5000-5FFF. The Computer would begin executing it at location 500A.

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

```
LOADM "PROG/MAC", 1000
```

would load "PROG/MAC" into memory locations 6000-6FFF. The Computer would begin executing it at address 600A.

SPECIAL INPUT/OUTPUT COMMANDS

BASIC offers two special input/output commands. These commands input and output data directly to a particular sector. They do this through bypassing the entire disk's filing system.

The first, 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:

```
DSKI$ 0, 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 with:

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

Since DSKI$ will read any sector on the disk, it is the only BASIC command which will read the directory sector. This sample program uses DSKI$ to search the directory for filenames with the extension "DAT":

```
10 FOR X=3 TO 11
20 DSKI$ 0,17,X,A$,B$
30 C$ = A$ + LEFT$(B$,127)
40 NAM$(0) = LEFT$(C$,8)
50 EXT$(0) = MID$(C$,9,3)
60 FOR N=1 TO 7
70 NAM$(N) = MID$(C$,N*32+1,8)
80 EXT$(N) = MID$(C$,9+N*32,3)
90 NEXT N
100 FOR N=0 TO 7
110 IF EXT$(N) = "DAT" AND
    LEFT$(NAM$(N),1)<>CHR$(0) THEN PRINT
    NAM$(N)
120 NEXT N
130 NEXT X
```

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

1. not to output data over the directory sectors unless you no longer plan to use the directory.

2. not to output data over other data you presently have stored on the disk.

The format of DSKO$ is:

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

String1 will go in the first 128 bytes of the sector. *String2* will go 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. "FIRST STRING" will go in the first 128 bytes of this sector. "SECOND STRING" will go in the second 128 bytes.

APPENDIXES

PROGRAMMING EXERCISE ANSWERS

PROGRAMMING EXERCISE #5-1

```
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: CLS
80 PRINT "INPUT ANIMALS YOU WANT TO ADD"
90 PRINT "PRESS <ENTER> WHEN FINISHED"
100 INPUT A$
110 IF A$ = "" THEN 140
120 WRITE #2, A$
130 GOTO 80
140 CLOSE #2
150 KILL "ANIMALS/DAT"
160 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
40 INPUT #1, A$
50 PRINT: PRINT A$
60 INPUT "DO YOU WISH TO DELETE THIS"; 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 --"
20 PRINT "(1) STORE A NEW FILE"
30 PRINT "(2) SEE THE FILE"
40 PRINT "(3) END"
50 INPUT 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 : INPUT "DO YOU WANT TO CHANGE THE
    FILE"; Q2$
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$
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$
310 GOTO 160
320 PRINT: INPUT "DO YOU WISH TO ADD A NEW
    MEMBER"; Q3$
330 IF Q3$ = "NO" THEN 380
340 GOSUB 430
350 IF N$ = "" THEN 380
360 WRITE #2, N$, A$, P$
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

This 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 180
60 NEXT X
70 INPUT "WHICH RECORD(1-10)"; X
80 IF X > 10 THEN 170
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$,,A$,,C$,,S$,,Z$
140 INPUT "DO YOU WANT TO CHANGE THIS"; R$
150 IF R$ = "YES" THEN GOSUB 180
160 GOTO 70
170 CLOSE #1: END
180 INPUT "NAME"; N$
190 INPUT "ADDRESS "; A$
200 INPUT "CITY :"; C$
210 INPUT "STATE:"; S$
220 INPUT "ZIP :"; Z$
230 WRITE #1, N$, A$, C$, S$, Z$
240 PUT #1, X
250 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
```

CHAPTER CHECKPOINT ANSWERS

CHAPTER 2

1. Unless the disk has been 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 an individual block of information stored on the disk, under a filename.

4. Information stored in memory will only be there temporarily. It will be destroyed if the Computer is turned OFF or if you execute a NEW, LOAD, DISKINI, BACKUP, or COPY command. (We'll discuss BACKUP and COPY in the next chapters). Information stored on disk will be there permanently. It won't be destroyed if the Computer's turned off or if memory is cleared. (Don't leave a disk in the drive when you turn the Computer off. We'll explain why in the next chapter.)

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

CHAPTER 3

1. Turning the Computer ON or OFF while the disk is in its drive may damage the disk.

2. Only felt tip pens may be used to write on the disk's label. Hard point pens and pencils may 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 will ask you to insert the destination disk and press (ENTER). This procedure is repeated until the Computer prints OK. On a multi-drive system, type the BACKUP command specifying the drive number with 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. A file can be renamed with the RENAME command. For example, RENAME "OLDFILE/NAM" TO "NEWFILE/NAM" renames OLDFILE/NAM to NEWFILE/NAM. You must specify the extension for both filenames so the Computer can find them.

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

3. Unless otherwise specified, the Computer always uses Drive 0. This can be changed by typing DRIVE 1, which enables you to access Drive 1 without having to specify the number in your command. (i.e., now DIR and DIR1 would both get you 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 must be CLOSEd so that the information still in the buffer will end up where it's supposed to and so that the file can be reopened. 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. When you OPEN a "sequential access" file, you can only OPEN it for "I" or "O"—not both. You can't output to a file opened for "I," nor can you input from a file opened for "O."

2. No. The file must first be closed and then reopened for input.

CHAPTER 7

1. Records are equal-sized divisions in your disk file where you can put your data. Since each is the same size, the Computer can use them to access your data directly.

2. Fields are subdivisions 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, it can determine where each individual record is (by the size of the records).

4. Since each record of the file has a known location, the Computer can access it without going through the preceding parts of the file, as it would if the file was sequential.

CHAPTER 8

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

2. The Computer first WRITEs the number's sign (a minus sign if it's negative or a blank space if it's positive). Then it WRITEs the number itself. Immediately following the number, it WRITEs on trailing blank space.

3. A string is written with quotation marks around it.

4. INPUT inputs only the data items listed, while 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. A string is printed simply as the string itself. It is not enclosed in quotation marks.

CHAPTER 9

1. The Computer will set the record length at 256 bytes.

2. The data must have a field with a specific length for it to be LSET. This length is assigned in the FIELD line.

3. MKN$ converts a number into a 5-byte coded string.

CHAPTER 10

1. You must type an A at the end of your SAVE command if you plan to ever MERGE it with a program in memory.

2. The line number of the program saved on disk prevails.

3. The Computer reserves two buffers when you power-up.

4. The Computer reserves a total of 256 bytes of buffer space when you power-up.

5. FILES 3, 3000 get the Computer to reserve 3 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 which contains all your checks and deposits for the entire year. You can print them out by the month or the year. If you want to use your printer, change the appropriate PRINT lines to PRINT #−2. (See Chapter 21 in *Getting Started with Color BASIC*).

```
10 '  Checkbook Program
20 '
30 '  This program provides a
record of your checks,
40 '  deposits, and balances.
The checks can be labeled
50 '  with an account number to
show to what expense
60 '  they were paid, such as
medical,rent,food,etc.
70 '  The program uses direct
addressing,each file record
80 '  being 40 bytes long, and
formatted as follows:  8
90 '  bytes for the date, 4 bytes
for the check or deposit
100 '  slip number, 20 bytes for
the recipient of the check
110 '  3 bytes for the account
number, and 5 bytes for the
120 '  amount of the check or
deposit.
130 '
140 CLEAR 1000
150 DIM CHK$(50)
160 CLS
170 PRINT @ 107,"SELECTIONS:"
180 PRINT @ 162,"1) ADD CHECKS
TO YOUR FILE"
190 PRINT @ 194,"2) LIST YOUR
CHECKS,DEPOSITS,"
200 PRINT @ 229,"AND BALANCES"
230 PRINT @ 322,"3) END JOB?"
240 PRINT @ 394, "(1,2, OR 3)"
250 AN$=INKEY$
260 IF AN$="" THEN 250
270 ON VAL(AN$) GOSUB 310,700,
1080,1560
280 GOTO 160
290 '
300 '
310 '  This subroutine inputs
the data.
320 '
330 OPEN "D",#1,"CHECKS/DAT",40
340 FIELD #1,8 AS DATE$,4 AS CHNO$,
20 AS PDTO$,3 AS ACNO$,5 AS AMT$
350 REC = LOF(1)
360 REC = REC + 1
370 CLS
380 PRINT @ 64, "CHECK OR
DEPOSIT(C/D)"
390 AN$ = INKEY$
400 IF AN$ = "D" THEN 430
410 IF AN$ = "C" THEN 490
420 GOTO 390
430 INPUT "DEPOSIT DATE
(MM/DD/YY)";D$
440 INPUT "DEPOSIT SLIP NUMBER
(NNNN)";C$
450 P$ = ""
460 INPUT "ACCOUNT NUMBER(NNN)";A$
470 INPUT "AMOUNT OF DEPOSIT";AMT
480 GOTO 550
490 INPUT "CHECK DATE(MM/DD/YY)";D$
500 INPUT "CHECK NUMBER(NNNN)";C$
510 INPUT "PAID TO";P$
520 INPUT "ACCOUNT NUMBER(NNN)";A$
530 INPUT "AMOUNT OF CHECK";AMT
540 AMT = -AMT
550 LSET DATE$ = D$
560 LSET CHNO$ = C$
570 LSET PDTO$ = P$
580 LSET ACNO$ = A$
590 LSET AMT$ = MKN$(AMT)
600 PUT #1,REC
610 PRINT @ 320, "MORE INPUT(Y/N)"
620 AN$ = INKEY$
630 IF AN$ = "N" THEN 660
640 IF AN$ = "Y" THEN 360
650 GOTO 620
660 CLOSE #1
670 RETURN
680 '
690 '
700 '  This subroutine balances the
checkbook and outputs the results.
710 '
720 OPEN "D",#1,"CHECKS/DAT",40
730 FIELD #1,8 AS DATE$,4 AS CHNO$,
20 AS PDTO$,3 AS ACNO$,5 AS AMT$
740 CLS
750 PRINT @ 160,"DO YOU WANT A
LISTING FOR A MONTH OR FOR THE"
760 PRINT @ 192,"WHOLE YEAR? (Y/M)"
770 INPUT A$
780 IF A$ = "M" THEN PRINT @ 254,
```

```
"WHAT MONTH(MM)":INPUT MN$
790 BAL = 0
800 FOR REC = 1 TO LOF(1)
810 GET #1,REC
820 BAL = BAL + CVN(AMT$)
830 IF A$ = "M" AND LEFT$(DATE$,2)
<> MN$ THEN 1030
840 CLS
850 IF PDTO$ = "
"                                    THEN 930
860 PRINT @ 64, "DATE OF CHECK:":
PRINT @ 84,DATE$
870 PRINT "CHECK NUMBER:":PRINT
@ 116,CHNO$
880 PRINT "PAID TO:":PRINT @
148,PDTO$
890 PRINT @ 160,"ACCOUNT NUMBER:":
PRINT @ 180,ACNO$
900 PRINT "AMOUNT OF CHECK:":PRINT
@ 211,USING "$$###.##";-CVN(AMT$)
910 PRINT "BALANCE:":PRINT @ 243,
USING "$$###.##";BAL
920 GOTO 980
930 PRINT:PRINT:PRINT "DATE OF
DEPOSIT:":PRINT @ 85,DATE$
940 PRINT "DEPOSIT SLIP NUMBER:":
PRINT @ 117,CHNO$
950 PRINT "ACCOUNT NUMBER:":PRINT
@ 149,ACNO$
960 PRINT "AMOUNT OF DEPOSIT:":
PRINT @ 180,USING "$$###.##";
CVN(AMT$)
970 PRINT "BALANCE:":PRINT @
212,USING "$$###.##";BAL
980 PRINT @ 256, "PRESS <ENTER>
FOR NEXT RECORD OR <R> TO RETURN
TO 'SELECTIONS'"
990 AN$ = INKEY$
1000 IF AN$ = CHR$(13) THEN 1030
1010 IF AN$ = "R" THEN 1040
1020 GOTO 990
1030 NEXT REC
1040 CLOSE #1
1050 RETURN
1540 '
1550 '
1560 '  This subroutine terminates
the program.
1570 '
1580 END
```

SAMPLE PROGRAM #2
SORTING YOUR CHECKS.

This subroutine will be especially helpful at tax time. It takes the checks file which you created in "Sample Program #1" and sorts all the checks by account. Want to know exactly how much you spent on medical bills (or business expenses, contributions, etc.)? This program will let you know right away.

```
210 PRINT @ 258,"3) SORT YOUR
CHECKS BY"
220 PRINT @ 293,"ACCOUNT NUMBER?"
230 PRINT @ 322,"4) END JOB?"
240 PRINT @ 394, "(1,2,3,OR 4)"
270 ON VAL(AN$) GOSUB 310,700,
1080,1560
1060 '
1070 '
1080 '  This subroutine sorts the
checks from  those with the
1090 '  smallest account numbers
to the largest account numbers
1100 '  using a "bubble sort".
Each check is handled as one
1110 '  data string to make the
swaps easier.
1120 '
1130 OPEN "D",#1,"CHECKS/DAT",40
1140 FIELD #1,40 AS INFO$
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)
1370 PRINT @ 130,"CHECK NUMBER:"
:PRINT @ 149,MID$(CHK$(I),9,4)
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$ = CHR$(13) THEN 1510
1490 IF A2$ = "R" THEN 1520
1500 GOTO 1470
1510 NEXT I
1520 CLOSE #1
1530 RETURN
```

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 will store their addresses and phone numbers also.

```
10 '  Create list and alphabetize.
20 '
30 '  The object of this program
is to create a file of
40 '  alphabetically arranged names
and phone numbers.  The
50 '  names and numbers are first
input into an array,ARRAY$(I),
60 '  then put into alphabetical
order, and finally put into
70 '  a disk file called "NAMES/NOS".
The file is 35 bytes
80 '  long, all of it allotted to
one variable,INFO$.  The
90 '  file can be added to anytime
after its creation and will
100 ' automatically be alphabetized.
The program can be used
110 ' in conjunction with the "Search
a list" program (sample
120 ' program #4).
130 '
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 any
200 '  records currently on it.
210 '
220 IF LOF(1) = 0 THEN I=1:GO TO 310
230 FOR I=1 TO LOF(1)
240 GET #1,I
250 ARRAY$(I) = INFO$
260 NEXT I
270 '
280 '  The new names and numbers
are input and then concatenated
290 '  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 '  Then, ARRAY$(I) is put into
alphabetical order.
460 '
470 FOR J=1 TO I
480 FOR K=J TO I
490 IF ARRAY$(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 '  Finally, the list is trans-
ferred 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 called "NAMES/NOS
40 '  exists--see "Create list and
alphabetize--Sample program #3)
50 '  This program searches a disk
file which holds names and
60 '  phone numbers in alphabetical
order. The file is a direct
70 '  access file called "NAMES/NOS",
is 35 bytes long, and
80 '  is formatted as follows:  24
bytes for the name; 3 bytes
90 '  for the area code; 8 bytes
for the phone number.  The
100 '  program uses iterative
searching.
110 '
120 OPEN "D",#1,"NAMES/NOS",35
130 FIELD #1,24 AS NAME$,3 AS
AREA$,8 AS PHONE$
140 CLS
150 PRINT @ 99 ,"ENTER NAME(LAST,
FIRST MI)
160 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 the last
record first because it won't
290 '  be checked in the regular
search
300 '
310 GET #1,LAST
320 IF NAME$ = NM$ THEN 450
330 '
340 '  Program keeps comparing NM$
with NAME$ from record MID
350 '  until NM$ is found or enough
records have been seen
360 '  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 <Q>
TO QUIT"
500 AN$ = INKEY$
510 IF AN$ = "Q" THEN CLOSE: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 '  Subprograms for modifying
NM$ to a 20 byte string
```

```
790 '
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 in a hurry with this program.

```
10 '  Edit your names file
20 '
30 '  The object of this program
is to edit the "NAMES/NOS" file
40 '  from "Create list and alpha-
betize" (Sample program #3). The
50 '  program can either retain a
record, change one of the variables
60 '  in that record, or delete the
record entirely from the file.
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,"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$ :
GO TO 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 GO TO 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 RETURN
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
```

SAMPLE PROGRAM #6
GRADING TESTS

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

```
10 '  Test program
20 '
30 '  The object of this program
is to input several
40 '  files--a names file and several
test files.
50 '  The files can then be accessed
as desired and the
60 '  test scores processed to find
averages and standard
70 '  deviations.  The files are all
sequential access files.
80 '
90 '
100 '  Main module of program
110 '
120 DIM NAME$(30),GRADE(6,30)
130 CLS
140 PRINT @ 107,"SELECTIONS"
160 PRINT @ 164,"1) CREATE A
'NAMES' FILE"
170 PRINT @ 196,"2) ADD A NEW
TEST FILE"
180 PRINT @ 228,"3) PROCESS SCORES"
190 PRINT @ 260,"4) END"
210 PRINT @ 331, "1,2,3 OR 4"
220 AN$=INKEY$
230 IF AN$="" THEN 220
240 ON VAL(AN$) GOSUB 290,430,
640,1430
250 GOTO 130
260 '
270 '  This subroutine builds a
"NAMES" file.
280 '
290 OPEN "O",#1,"NAME/FIL"
300 CLS
310 PRINT @ 96,"ENTER NAME OF
STUDENT:"
320 LINE INPUT NAME$
330 WRITE #1,NAME$
340 PRINT @ 196,"(PRESS <ENTER> TO
ENTER":PRINT @ 228,"ANOTHER NAME,
PRESS <Q>":PRINT @ 260,"TO QUIT)"
350 AN$=INKEY$
360 IF AN$="" THEN 350
370 IF AN$<>"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 the
600 '  desired test files and then
processes them either
610 '  on a class basis or an
individual basis, and
620 '  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 portion processes the
scores by the student.
890 '
900 FOR Y=1 TO YEND
910 CLS
920 PRINT @ 105,NAME$(Y)
930 PRINT @ 137,"SCORES:"
940 STUTOT = 0
950 FOR X=1 TO N
960 PRINT TAB(10) GRADE(X,Y)
970 STUTOT = STUTOT + GRADE(X,Y)
980 NEXT X
990 AVE(Y) = STUTOT / N
1000 NUM = 0
1010 FOR X=1 TO N
1020 NUM = (AVE(Y) - GRADE(X,Y))↑2
+ NUM
1030 NEXT X
1040 SD = SQR(NUM / N)
1050 PRINT USING "%       %##.##";
"AVERAGE:";AVE(Y)
1060 PRINT USING "%                    %
##.##";"STANDARD DEVIATION:";SD
1070 PRINT "PRESS <ENTER> TO SEE
NEXT NAME"
1080 AN$ = INKEY$
1090 IF AN$ = CHR$(13) THEN 1100
ELSE 1080
1100 NEXT Y
1110 CLS
1120 PRINT @ 105,"NO MORE NAMES"
1130 GOTO 1350
1140 '
1150 '  This portion processes the
scores by the test number
1160 '  for the whole class.
1170 '
1180 INPUT "  WHICH TEST NUMBER";X
1190 CLS
1200 PRINT @ 4,"DATA FOR TEST
NUMBER ";X
1210 PRINT "  NAME";TAB(25)"SCORE"
1220 TTOT = 0
1230 FOR Y=1 TO YEND
1240 TTOT = TTOT + GRADE(X,Y)
1250 PRINT TAB(1) NAME$(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))
1330 PRINT:PRINT USING
"%                    %#%%##.##";
"AVERAGE FOR TEST #";X;":";AVE
1340 PRINT USING "%                    %
##.##";"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

These four programs will display 3 scenes — a house and two rooms. Each scene is stored on disk as a program file.

```
10 '  "DISPLAY/BAS"
20 '
30 '  The object of this program
is to show you how you can
40 '  access another program from
your main program.  It uses
50 '  a main program called
"DISPLAY/BAS" and three graphics
60 ' programs called "HOUSE/BAS",
"FOYER/BAS", and "STAIRS/BAS".
70 '  (Naturally they must be on
disk before you can run this
```

```
80 '  program.)
90 '
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;
BM50,156;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
170 PAINT (67,70),4,4
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 PCLS
40 PMODE 3,1
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;R8;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)-(255,0),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;BM48,31;U18;
BM40,22;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;R2;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 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 ' The object of this program
is to build three direct access
40 ' files, one a listing of a
balanced budget, another, a listing
50 ' of transactions, and the
third, a listing of the updated
60 ' budget.  The program allows
for carryover from the previous
70 ' period's budget.  A journal
can be printed out giving a list
80 ' of the budget,expenses, and
balances.  (NOTE:  As written,
90 ' this program requires a printer
for outputting the journal.
100 ' However, with slight modifi-
cation, it could be used without
110 ' a printer.)
120 '
130 '
140 ' Main module of 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 360,950,
1450,1970
260 GOTO 160
270 '
280 ' This subroutine builds the
budget file (called
290 ' BUDGET.ORG), and builds or
updates the file BUDGET.UPD
300 ' It allows you to input the
start date of the budget
310 ' and the total amount you
have to divide up to accounts.
320 ' Tentative amounts are entered
for each account and a
330 ' running balance is kept to
advise you of the amount
340 ' left in your total budget.
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 1810
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 @ 162,"PROJECTED INCOME
FROM:"
500 PRINT @ 196,"SALARY:"
510 PRINT @ 228,"OTHER:"
520 PRINT @ 96
530 INPUT "  DATE(MM/DD/YY):";DATE$
540 PRINT @ 162, "PROJECTED INCOME
FROM:"
550 INPUT "     SALARY:";SAL
560 INPUT "     OTHER:";OTHER
570 BTOT = SAL + OTHER
580 CLS
600 PRINT @ 9,"CURRENT BUDGET"
610 PRINT "ACCT#  DESCRIPTION
BALANCE"
620 SUMBUD = 0
630 FOR I=1 TO 9
640 PRINT USING "####% %
%                %#####.##-";ACNO(I);
SPACE$;DESC$(I);AMT(I)
650 SUMBUD = SUMBUD + AMT(I)
660 NEXT I
670 PRINT @ 86,USING "$#####.##-"
;AMT(1)
680 PRINT @ 419,USING
"%                %$$####.##";
"REMAINING MONEY:";BTOT - (SUMBUD
- PTOT)
690 PRINT @ 451, "ENTER ACCT# OF
ITEM TO BE"
700 INPUT "   CHANGED (000 TO
QUIT)";AN
710 IF AN = 0 THEN 790
720 CLS
730 N = AN / 100
740 PRINT @ 105,ACNO(N)
750 PRINT @ 138,DESC$(N)
760 PRINT @ 170,"$";AMT(N)
770 PRINT:INPUT "       NEW
AMOUNT";AMT(N)
780 GO TO 580
790 DATE = VAL(LEFT$(DATE$,2) +
MID$(DATE$,4,2) + RIGHT$(DATE$,2))
800 LSET OAMT$ = MKN$(DATE)
810 PUT #1,1
820 FOR I=1 TO 9
830 LSET OAMT$ = MKN$(AMT(I))
840 LSET UPDAMT$ = MKN$(AMT(I))
850 PUT #1,I+1
860 PUT #2,I
870 NEXT I
880 CLOSE
890 RETURN
900 '
910 ' This subroutine builds a
transaction file called TFILE.DAT
920 ' which contains any updates
to the budget, and updates the
930 ' file BUDGET.UPD .
940 '
950 OPEN "D",#1,"BUDGET/UPD",5
960 OPEN "D",#2,"TFILE/DAT",36
970 FIELD #1,5 AS UPDAMT$
980 FIELD #2,3 AS ACNO$,8 AS DATE$,
20 AS DESC$,5 AS TAMT$
990 FOR I=1 TO 9

1000 GET #1,I
1010 AMT(I) = CVN(UPDAMT$)
1020 NEXT I
1030 GOSUB 1810
1040 CLS
1050 SUMBUD = 0
1060 PRINT @ 9,"CURRENT BUDGET"
1070 PRINT "ACCT#  DESCRIPTION
BALANCE"
1080 FOR I=1 TO 9
1090 PRINT USING "####% %
0                %#####.##-";ACNO(I);
SPACES$;DESC$(I);AMT(I)
1100 SUMBUD = SUMBUD + AMT(I)
1110 NEXT I
1120 PRINT @ 86,USING
"$#####.##-";AMT(1)
1130 PRINT @ 419,USING "%                %
$$####.##";"TOTAL BALANCE:";SUMBUD
1140 PRINT @ 451,"ENTER ACCT# OF
ITEM TO BE
1150 INPUT "   UPDATED (000 TO QUIT)";AN
1160 IF AN = 0 THEN 1350
1170 CLS
1180 N = AN / 100
1190 PRINT @ 95,AN
1200 PRINT DESC$(N)
1210 PRINT USING "%                %
$$###.##";"CURRENT BALANCE";AMT(N)
1220 PRINT:INPUT "DATE(MM/DD/YY)";DT$
1230 PRINT "DESCRIPTION OF
TRANSACTION:"
1240 INPUT DS$
1250 PRINT "AMOUNT OF TRANSACTION:"
1260 PRINT "(NEGATIVE NUMBER FOR A
CREDIT)"
1270 INPUT TRANS
1280 AMT(N) = AMT(N) - TRANS
1290 LSET ACNO$ = RIGHT$(STR$(AN),3)
```

```
1300 LSET DATE$ = DT$
1310 LSET DESC$ = DS$
1320 LSET TAMT$ = MKN$(TRANS)
1330 PUT #2,LOF(2)+1
1340 GOTO 1040
1350 FOR I=1 TO 9
1360 LSET UPDAMT$ = MKN$(AMT(I))
1370 PUT #1,I
1380 NEXT I
1390 CLOSE
1400 RETURN
1410 '
1420 ' This subroutine prints out
a journal listing the
1430 ' budget,transactions,and
balances.
1440 '
1450 OPEN "D",#1,"BUDGET/ORG",5
1460 FIELD #1,5 AS AMT$
1470 OPEN "D",#2,"TFILE/DAT",36
1480 FIELD  #2,3 AS TACNO$,8 AS
TDATE$,20 AS TRDESC$,5 AS TMT$
1490 GOSUB 1810
1500 CLS
1510 PRINT @ 172,"PRINTING"
1520 Get #1, I
1530 DATE$ = STR$(CVN(AMT$))
1540 IF LEN(DATE$) < 6 THEN DATE$
= " " + DATE$
1550 DATE$ = LEFT$(DATE$,2) +
"/" + MID$(DATE$,3,2) + "/" +
RIGHT$(DATE$,2)
1560 PRINT #-2,TAB(30)"BUDGET FOR
THE PERIOD"
1570 PRINT #-2,TAB(31)
"STARTING  ";DATE$
1580 PRINT #-2:PRINT #-2
1590 PRINT #-2,TAB(28)"ACCOUNT OR"
1600 PRINT #-2,TAB(10)
"ACCOUNT";TAB(27)"TRANSACTION"
1610 PRINT #-2,TAB(10)"NUMBER";
TAB(14)"DATE";TAB(27)"DESCRIPTION";
TAB(47)"TRANSACTION";TAB(61)
"BALANCE"
1620 FOR I=2 TO LOF(1)
1630 GET #1,I
1640 PRINT #-2
1650 PRINT #-2,TAB(12)ACNO(I-1);
TAB(17)DATE$;TAB(27)DESC$(I-1);
TAB(61)CVN(AMT$)
1660 BAL=CVN(AMT$)
1670 FOR J=1 TO LOF(2)
1680 GET #2,J
1690 IF ACNO(I-1) <> VAL(TACNO$)
THEN 1730
1700 BAL=BAL - CVN(TMT$)
1710 IF CVN(TMT$) <.0 THEN CR$="CR"
ELSE CR$=""
1720 PRINT #-2,TAB(17)TDATE$;TAB(27)
TRDESC$;TAB(47)ABS(CVN(TMT$));CR$
;TAB(61)BAL
1730 NEXT J
1740 NEXT I
1750 CLOSE
1760 RETURN
1770 '
1780 ' This subroutine sets the
values of the account numbers,
1790 ' ACNO(I), and account
descriptions,DESC$(I).
1800 '
1810 FOR I=1 TO 9
1820 ACNO(I) = I * 100
1830 NEXT I
1840 DESC$(1) = "FOOD"
1850 DESC$(2) = "RENT"
1860 DESC$(3) = "CAR"
1870 DESC$(4) = "UTILITIES"
1880 DESC$(5) = "INSURANCE"
1890 DESC$(6) = "TAXES"
1900 DESC$(7) = "CLOTHING"
1910 DESC$(8) = "ENTERTAINMENT"
1920 DESC$(9) = "MISCELLANEOUS"
1930 RETURN
1940 '
1950 ' This subroutine terminates
the program.
1960 '
1970 END
```

ASCII CHARACTER CODES

These are the ASCII codes for each of the characters on your keyboard. The first column is the character; the second is the code in decimal notation; and the third converts the code to a hexadecimal (16-based number).

Chapter 15 shows how to use these codes with CHR$ to produce a character.

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
↑*	94	5E
↓*	10	OA
←*	8	O8
→*	9	O9
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

**If shifted, 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).*

LOWER-CASE CODES

These are the ASCII codes for lower-case letters. You can produce these characters by pressing the (SHIFT) and (0) keys simultaneously to get into an upper/lower case mode. The lower case letters will appear on your screen in reversed colors (green with a black background).

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

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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

MEMORY MAP

DECIMAL	HEX	CONTENTS	DESCRIPTION
0-255	0000-00FF	System Direct Page RAM	See Section IV of *Getting Started with COLOR BASIC* for detailed information.
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* *top of RAM* is 16383 for 16K systems; 32767 for 32K systems	0989-*top of RAM* *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. 256 bytes are reserved for this on start-up. This value can be reset by the FILES statement.
		2. File Control Blocks (FCBs)	Control data on each user buffer. 843 bytes are reserved for this on start-up. This value can be reset by the FILES statement: (number of buffers set by FILES + 1) x 281 bytes.
		3. Graphics Video Memory	Space reserved for graphics video pages. 6144 bytes or 4 pages are reserved for this on start-up. This value can be reset by the PCLEAR statement: number of pages reserved by PCLEAR X 1,536 bytes per page. (Note: All pages must start at a 256-byte page boundary—i.e., a memory location divisible by 256.)
		4. BASIC Program Storage 5. BASIC Variable Storage 6. Stack	Space reserved for BASIC Programs and Variables. 6455* bytes (16K systems) or 22,839* bytes (32K systems) are reserved for this on start-up. This value can be reset by different settings of Random File Buffers, FCBs, Graphics Video Memory, String Space or User Memory.
		7. String Space	Total space for string data. On start-up, 200 bytes are reserved, but this can be reset by the CLEAR statement.
		8. User Memory	Total space for user machine-language routines. No space is reserved for this on start-up, but this can be reset by the CLEAR statement.
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	

**If you execute a PRINT MEM command, on start-up, you will get a number a little lower than this because of the overhead necessary to execute this command.*

SPECIFICATIONS

Type of disks	5¼" mini-diskettes Radio Shack Catalog Number 26-305 26-405 (package of three) or 26-406 (package of 10)
Disk Organization **(Formatted disk)**	Single-sided Double-density 35 tracks 18 sectors per track 256 data bytes per sector Directory on track 17
Operating Temperature	18 to 113 degrees Fahrenheit 10 to 45 degrees Centigrade
Memory Capacity	
Unformatted	218.8 kilobytes per disk 6.2 kilobytes per track
Soft sector **(I/O sector/track)**	179.1 kilobytes per disk 5.1 kilobytes per track
Data transmission speed	250 kilobits per second
Access Time	
Track to track	5 m sec.
Average	100 m sec.
Settling time	15 m sec.
Number of indexes	1
Weight of Disk Drive	3.8 kg.
Disk Controller	WD1793

ERROR MESSAGES

/0 Division by zero. The Computer was asked to divide a number by 0, which is impossible. You could also get this error message if you do not enclose a filename in quotation marks.

AE File Already Exists. You are trying to RENAME or COPY a file to a filename which Already Exists.

AO Attempt to open a data file which is Already Open.

BR Bad Record Number. You have used an impossible record number in your PUT or GET line. Either it is 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 which dimensions array A for 12 or more elements, you will get this error.

CN Can't continue. If you use the command CONT and you are at the END of the program, you will get this error.

DD Attempt to redimension an array. An array can only be dimensioned once. For example, you cannot have DIM A(12) and DIM A(50) in the same program.

DF Disk Full. The Disk you are trying to store your file on is Full. Use another disk.

DN This is either a Drive Number or Device Number error.

Drive Number Error. You are using a drive number higher than 3. You will 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 (DSKINI0 or BACKUP 0).

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 nonexisting 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 with no line numbers.

ER Write or Input past End of Record (direct access only). You are attempting to PUT more data in the record than it can hold or INPUT more data than it contains.

FC Illegal Function Call. This happens when you use a parameter (number) with a BASIC word that is out of range. For example SOUND (260,260) or CLS(10) will cause this error. Also RIGHT$(S$,20), when there are only 10 characters in S$, would cause it. Other examples are a negative subscript, such as A(–1), or a USR call before the address has been POKEd in.

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, 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 what you are attempting to do. For example, you are attempting to GET a record from a file OPENed for "I" (use "D") or WRITE data to a file OPENed for "I" (use "O").

FN Bad File Name. You used an unacceptable format to name your file.

FO Field Overflow. The field length is longer than the record length.

FS 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 IO for instructions on what to do.

ID Illegal Direct statement. You can only use INPUT 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've reached the end of the file. When you have, CLOSE it.

IO Input/Output error. The Computer is having trouble inputting or outputting information to the disk.

(1) Make sure there is a disk inserted properly in the indicated drive and 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 using a different one or reformatting it. (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 the Radio Shack Repair Center.

This error could also be caused by input/output problems with another device, such as the tape recorder.

LS String too Long. A string may only be 255 characters.

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 more than one disk drive, 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.

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 input or output data to a file until you have OPENed it.

OB Out of Buffer space. Use FILES to reserve more space.

OD Out of Data. A READ was executed with insufficient DATA for it to READ. A DATA statement may have been left out of the program.

OM Out of Memory. All available memory has been 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 is in your program with no matching GOSUB.

SE Set to non-fielded string. The field in which you are attempting to LSET or RSET data in has not yet been FIELDed. Check the FIELD line.

SN Syntax error. This could result from a misspelled command, incorrect punctuation, open parenthesis, or an illegal character. Type the program line or command over.

ST String formula too complex. A string operation was too complex to handle. Break up the operation into shorter steps.

TM Type Mismatch. This occurs when you try to assign numeric data to a string variable (A$=3) or string data to a numeric variable (A = "DATA"). This could also occur if you do not enclose a filename in quotes.

UL Undefined Line. You have a GOTO, GOSUB, or other branching line in the program asking the Computer to go to a nonexisting line number.

VF Verification. You will only get the 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 IO for instructions on what to do.

WP Write Protected. You are trying to store information on a disk which is Write Protected. Either take the label off the write protect notch or use a different disk. If your disk is not Write Protected, then there is an input/output problem. See IO for instructions on what to do about this.

DISK BASIC SUMMARY

This is a short summary on each new DISK BASIC "command." You may also use any of the EXTENDED COLOR BASIC commands. (See *Getting Started with Extended Color BASIC* or the *Color Computer Quick Reference Card* for a complete listing.)

The first line gives the format to use in typing the command. The italicized words represent "parameters"—values which you can specify with the command.

This is the meaning of some of the parameters you may specify:

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 must have 1 to 8 characters.
The *extension* is optional. It can have 1 to 3 characters.
The *drive number* is optional. If you do not use it when opening a disk file, the Computer will use drive 0 (or the drive specified in the DRIVE command).

number
This may be a number (1, 5.3), a numeric variable (A, BL), a numeric function (ABS(3)), or a numeric operation (5+3, A−7).

string
This may be characters ("B," "STRING"), a string variable (A$, BL$), a string function (LEFT$(S$, 5)), or a string operation ("M"+A$).

data
This may be *number* or *string*.

BASIC WORD	PAGES DISCUSSED
BACKUP *source drive* TO *destination drive* Duplicates the contents of the *source drive* to the *destination drive*. If you only have one drive, specify it as the *source drive*. The Computer will prompt you to switch disks as it makes the backup copy. Executing this command will erase memory. `BACKUP 0 TO 1  BACKUP 0`	13-15
CLOSE # *buffer, ...* Closes communication to the *buffers* specified. (See OPEN for *buffer* numbers). If you omit the *buffer*, the Computer will close all open files. `CLOSE #1  CLOSE #1, #2`	26-27

COPY *filename1* TO *filename2*
Copies the contents of *filename1* to *filename2*. Each *filename* must include an *extension*. (See format for *filenames* above.) Executing this command will erase memory.
`COPY "FILE/BAS" TO "NEWFILE/BAS"`
`COPY "ORG/DAT:0" TO "ORG/DAT:1"`
	21

BASIC WORD	PAGES DISCUSSED

CVN(*string variable*) — 50
Converts a 5-byte coded *string* (created by MKN$) back to the number it represents.
```
X=CVN(A$)
```

DIR*drive number* — 11
Displays a directory of the disk in the *drive number* you specify. If you omit the *drive number,* the Computer will use drive 0. (Unless you use the DRIVE command to change this default.) This is a typical directory display:
```
MYPROG     BAS    0 B 3
YOURPROG   BAS    0 A 1
HERDATA    DATA   1 A 5
USRPROG    BIN    2 B 2
```
The first column is the name of the file. The second column is its extension. The third is the file type (0 = BASIC program, 1 = BASIC data file, 2 = machine language file, 3 = editor source file). The fourth column is the storage format (A = ASCII, B = Binary). The fifth column is the file length in granules.
```
DIR0     DIR
```

DRIVE *drive number* — 11-15
Changes the drive default to the *drive number* you specify. If you do not use the DRIVE command, the Computer will default to drive 0.
```
DRIVE 1
```

DSKINI*drive number* — 8
Formats a disk in the *drive number* you specify. Executing this command will erase memory.
```
DSKINI0     DSKINI1
```

DSKI$ *drive number, track, sector, string variable1, string variable2* — 61-62
Inputs data from a certain *sector* within a certain *track* on the disk in *drive number.* The first 128 bytes of data are input into *string variable1;* the second 128 bytes into *string variable2.*
```
DSKI$ 0, 12, 3, M$, N$
```

DSKO$ *drive number, track, sector, string1, string2* — 61-62
Outputs string data into the *sector, track,* and *drive number* you specify. *string1* is output into the first 128 bytes of the sector; *string2* is output into the second 128 bytes. Used improperly, this command could garble the contents of the disk.
```
DSKO$ 0, 2, 1, "FIRST DATA", "SECOND DATA"
```

EOF *(buffer)* — 27
Returns a 0 if there is more data to be read in the *buffer* and a −1 if there is no more data in it. (See OPEN for *buffer* numbers.)
```
IF EOF(1) = -1 THEN CLOSE #1
```

FIELD # *buffer, field size* AS *field name,...* — 48-49
Organizes the space within a direct access *buffer* into fields. (See OPEN for *buffer* numbers.) You specify the *size* and *name* of each *field.*
```
FIELD #1, 10 AS A$, 12 AS B$, 5 AS C$
```

BASIC WORD	PAGES DISCUSSED

FILES *buffer number, buffer size* — 54-55

Tells the Computer how many buffers to reserve in memory *(buffer number)*, and the total bytes to reserve for these buffers *(buffer size)*. If you do not use FILES, the Computer will reserve enough memory space for buffers 1 and 2, and will reserve a total of 256 bytes for those buffers.

```
FILES 1, 1000    FILES 5
```

FREE*(drive number)* — 20

Returns the number of free granules on the disk in the *drive number* you specify.

```
PRINT FREE(0)
```

GET # *buffer, record number* — 34-36

Gets the next record or the *record number* you specify, and puts it in the *buffer*. (See OPEN for *buffer* numbers).

```
GET #1, 5    GET #2, 3
```

INPUT # *buffer, variable name,...* — 26-28

Inputs data from the *buffer* you specify and assigns each data item in the *buffer* to the *variable name* you specify. (See OPEN for *buffer* numbers.)

```
INPUT #1, A$, B$
```

KILL *filename* — 20

Deletes the *filename* you specify from the disk directory. (See the format for *filenames* above.) You must include the *extension* with the *filename*.

```
KILL "FILE/BAS"    "KILL FILE/DAT:1"
```

LINE INPUT # *buffer, data* — 42-43

Inputs a line (all the *data* up to the ENTER character) from the *buffer* you specify. (See OPEN for *buffer* numbers).

```
LINE INPUT #1, X$
```

LOAD *filename*, R — 9

Loads the BASIC program file you specify from a disk into memory. By including R, the Computer will RUN the program immediately after loading it. If your *filename* does not have an *extension*, the Computer assumes it is BAS. (See the format for *filenames* above.) Executing this command will erase memory.

```
LOAD "PROGRAM", R    LOAD "ACCTS/BAS:1"
```

LOADM *filename, offset address* — 61

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 it is BIN. (See the format for *filenames* above.)

```
LOADM "PROG/BIN, 3522
```

LOC*(buffer)*

Returns the current record number of the buffer you specify. (See OPEN for *buffer* numbers.)

```
PRINT LOC(1)
```

BASIC WORD	PAGES DISCUSSED

LOF*(buffer)* — 37

Returns the highest numbered record of the buffer you specify. (See OPEN for *buffer* numbers.)

```
FOR R = 1 TO LOF(1)
```

LSET *field name = data* — 48-50

Left justifies the *data* within the *field name* you specify. If the *data* is larger than the field, the RIGHT characters will be truncated (chopped off).

```
LSET A$ = "BANANAS"     LSET B$ = T$
```

MERGE *filename,* R — 53-54

Loads a program *file* from disk and merges it with the existing program in memory. If you include R, the Computer will immediately run the program after merging it. (See the format for *filenames* above.) The disk program *file* cannot be MERGEd unless it was SAVEd with the A (ASCII) option.

```
MERGE "SUB/BAS"     MERGE "NEW", R
```

MKN$*(number)* — 50

Converts a *number* to a 5-byte coded string, for storage in a formatted disk file.

```
LSET B$ = MKN$(53678910)
```

OPEN *"mode," # buffer, filename, record length* — 26-28, 29-31, 33-38

Opens a place in memory called a *buffer* which will communicate data to and from a certain device. The *buffers* and the devices they communicate with are:

0—screen or printer (it is not necessary to open this buffer)
-1—tape recorder
-2—printer
1-15—disk drive

The communication *modes* you can use are:

I—inputting data from a sequential access file
O—Outputting data to a sequential access file
D—Inputting or outputting data to a direct access file

The *filename* you use should be in the format defined above. If you do not give *filename* an *extension,* the Computer will give it the *extension* DAT.

If you are opening communication to a direct access file, you can also specify the *record length.* If you don't, the *record length* will be 256 bytes.

```
OPEN "D", #1, "FILE", 15
OPEN "I", #2 "CHGE/DAT"
```

PRINT *# buffer, data list* — 27-28

PRINTs the *data* to the *buffer.* (See OPEN for *buffer* numbers.) You may use a comma or a semi-colon to separate each item in the *data list.*

```
PRINT #1, "DATA"
```

PRINT *# buffer,* USING *format; data list* — 45-46

Prints data to the *buffer* using the *format* you specify. The *format* is a *string* which can either specify a *numerical* or *string format.*

numerical formats may consist of any of the following:

#	sets the field of a number
.	formats a decimal point

BASIC WORD	PAGES DISCUSSED

,	formats a comma every third number
**	fills leading spaces with asterisks
$	places $ ahead of number
$$	floating dollar sign
+	in first position, causes sign to be printed before number; in last position causes sign to be printed after the number
^^^^	prints number in exponential notation
-	prints a minus sign after negative numbers

```
PRINT USING #1, "**.*"; 53.76
PRINT USING #2, "**$*.**-"; -3.678
```

string formats may consist of either:

% % fields the length of a string.

! ! prints the first character of the string

```
PRINT USING #1, "!"; "WHITE"
PRINT USING #1, "% %"; "YELLOW"
```

See *Going Ahead With Extended Color BASIC* for more information on the formats.

PUT # *buffer, record number* — 34

Assigns a *record number* to the data in the *buffer.* If you do not specify a *record number,* the Computer will assign it to the current record. (See OPEN for *buffer* numbers.)

```
PUT #2, 3    PUT #1, 4
```

RENAME *old filename* TO *new filename* — 19-20

Renames a file on disk to a *new filename.* You must specify the extension of both *filenames.*

```
RENAME "MFILE/DAT:1" TO "BFILE/DAT:1"
```

RSET *field name = data*

Right justifies the *data* within the field name you specify. If the *data* is larger than the field, the RIGHT characters will be truncated (the same as with LSET).

```
RSET M$ = "SOAP"
```

RUN *filename,* R — 9

Loads *filename* from disk and runs it. If R is included, all open files will remain open. (See the format for *filenames* above.)

```
RUN "FILE'    RUN "PROG/BAS", R
```

SAVE *filename,* A — 8

Saves *filename* on disk. If you do not give *filename* an *extension,* the Computer will give it the extension BAS. By using the A option, your program will be saved in ASCII format. (See the format for *filenames* above.)

```
SAVE "PROG/BAS"    SAVE "TEST:1", A
```

SAVEM *filename, first address, last address, execution address* — 61

Saves *filename* — a machine language program beginning at *first address* (in memory) and ending at *last address.* You also specify the *address* in which it will be *executed.* If you do not give *filename* an *extension,* the Computer will give it the extension BIN. (See the format for filenames above.)

```
SAVEM "FILE/BIN:1", &H5200, &H5800, &H5300
```

BASIC WORD	PAGES DISCUSSED
UNLOAD *drive number* Closes any open files in the *drive number* you specify. If you don't specify a *drive number* the Computer will use drive 0 (or the *drive number* you specified with DRIVE). `UNLOAD 0 UNLOAD`	13
VERIFY ON VERIFY OFF Turns the verify function ON or OFF. When VERIFY is ON, the Computer will verify all disk writes.	15
WRITE # *buffer, data list* Writes the data to the *buffer* you specify. (See OPEN for *buffer* numbers.) Use a comma to separate each item in the *data list.* `WRITE #1, A$, B$, C`	25-26, 34-35

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.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.
FORT WORTH, TEXAS 76102

CANADA
BARRIE, ONTARIO, L4M4W5

TANDY CORPORATION

AUSTRALIA
91 KURRAJONG ROAD
MOUNT DRUITT, N.S.W. 2770

BELGIUM
PARC INDUSTRIEL NANINNE
5140 NANINNE

UNITED KINGDOM
BILSTON ROAD, WEDNESBURY
WEST MIDLANDS WS10 7JN

8749470-BCo

PRINTED IN U.S.A.