A SPECTRUM BOOK

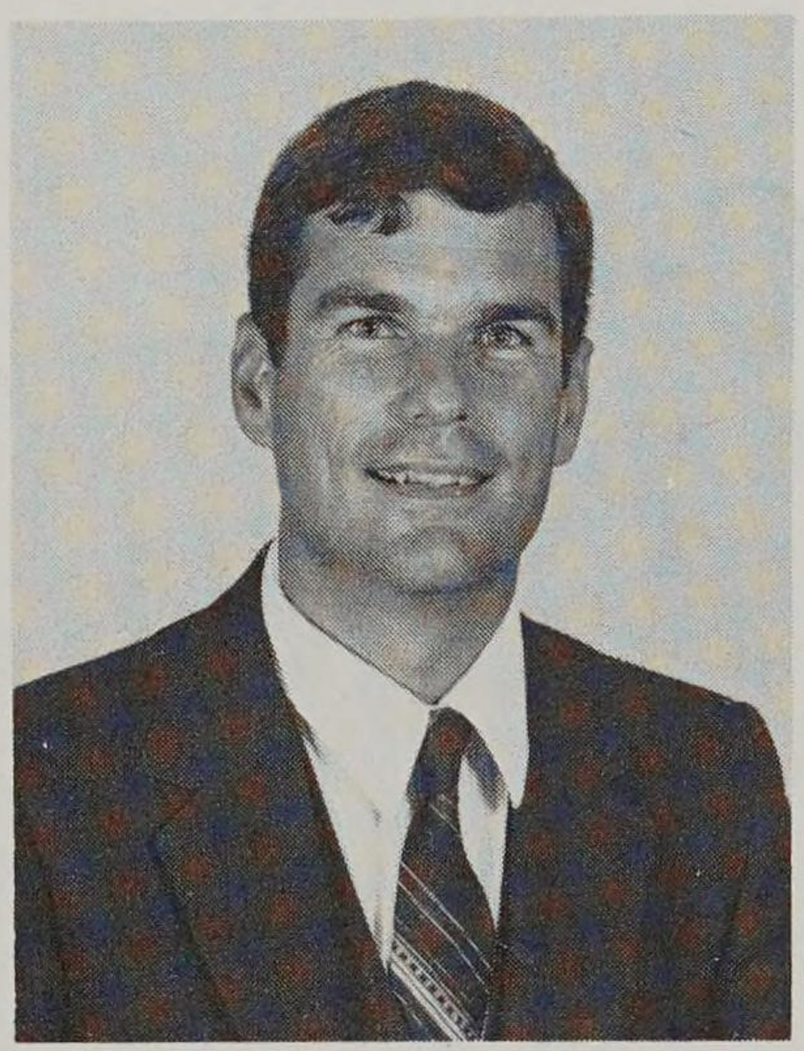
A MICROTREND BOOK

MS-DOS™ & PC-DOS ON THE IBM® PC

Charles Jackson

MS™-DOS & PC-DOS on the IBM® PC

23358



Charles Jackson grew up in San Diego, except for a three year interlude in Istanbul, Turkey in his teens. He graduated from UCLA in 1972 with a B.A. in Near Eastern Studies. After four years as an officer in the Marine Corps, he returned to graduate school at San Diego State and received a Master's degree in Linguistics in 1978. From 1979 to 1982 he taught French while he pursued a Ph.D. in Linguistics at UCSD.

In 1982, the success of his computer training business, Micro Management Resources, compelled him to leave the pleasant surroundings of academia and turn his full attention to providing on-site, hands-on training sessions for businesses in Southern California.

He derives great satisfication from explaining computer usage to others. He has turned more and more to writing books, articles, and a column in the *Computer News of San Diego*.

His company has grown rapidly and is now venturing into software development. He likes the fact that he does not have to worry about finding someone qualified to write the documentation.

MS™-DOS & PC-DOS on the IBM® PC

by

Charles Jackson

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Jackson, Charles H.
MS-DOS and PC-DOS on the IBM PC.

"A Spectrum Book."
"A Microtrend book."
Includes index.
1. IBM Personal Computer – Programming. 2. MS-DOS (Computer operating system) 3. PC-DOS (Computer operating system) I. Title.
QA76.8.I2594J33 1984 001.64'2 84-11602
ISBN 0-13-604281-3 (pbk.)

This book is available at a special discount when ordered in bulk quantities. Contact Prentice-Hall, Inc., General Publishing Division, Special Sales, Englewood Cliffs, N.J. 07632.

©1984 by Microtrend, Inc.
All rights reserved. No part of this book may be reproduced in any form or by any means without permission in writing from the publisher. A Spectrum Book. Printed in the United States of America.

1 2 3 4 5 6 7 8 9 10

ISBN 0-13-604281-3

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall of Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*
Editora Prentice-Hall do Brasil Ltda., *Rio de Janeiro*

Contents

Trademarks vi
Acknowledgments vii
Preface viii
Introduction 1

CHAPTER 1 Getting Started 3
CHAPTER 2 Preparing Disks 9
CHAPTER 3 Directories, Filenames and Wildcards 18
CHAPTER 4 Copying Files 32
CHAPTER 5 Fundamental DOS Commands 40
CHAPTER 6 Subdirectories 52
CHAPTER 7 Where Is It? Running Programs 70
CHAPTER 8 Batch Files and Automatic Start-ups 87
CHAPTER 9 Using the DOS Editor: EDLIN 100
CHAPTER 10 Advanced Batch File Usage and DOS Menus 112
CHAPTER 11 Making Backups 130
CHAPTER 12 Directing Data to Devices 151
CHAPTER 13 Filters, Pipes and More 166

APPENDIX A Summary of DOS Commands Covered in the Book 182
APPENDIX B Summary of DOS Commands Not Covered in the Book 187
APPENDIX C Summary of Internal Versus External DOS Commands 188
APPENDIX D ASCII Code Chart 189
APPENDIX E Summary of DOS Editing Keys and EDLIN Commands 191
APPENDIX F About the Keyboard 192

Index 194

Trademarks

Compaq is a registered trademark of Compaq Computer Corporation.

CP/M-86 is a registered trademark of Digital Research.

dBASE II is a registered trademark of Ashton-Tate.

IBM PC, IBM XT, IBM PC*jr,* and IBM BASIC are registered trademarks of International Business Machines Corporation.

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

MS™-DOS is a registered trademark of Microsoft.

MultiMate is a registered trademark of Softword Systems, Inc.

PC-Talk III is a registered trademark of Freeware.

SixPakPlus is a registered trademark of AST Research.

UCSD p-System is a registered trademark of the Regents of the University of California.

Volkswriter is a registered trademark of Lifetree.

WordStar is a registered trademark of MicroPro International Corp.

Limitations of Warranty and Liability

The author, producer, and publisher will not be held liable for any direct, indirect, or incidental damages which may result from any error or omission in this publication, or in any disk which may be related to this publication.

Acknowledgments

Writing this book was fun, enervating, engrossing, and arduous, all at once. I am indebted to some very special people:

– Elizabeth Burdett and Linnea Dayton, the editors, for their perseverance and attention to detail.

– Jim Reesman of Computerland of San Diego, for his help in all the technical areas that I usually try so hard to avoid.

– Sandy Erickson, my friend and colleague, for bearing more than her share of the workload in our business while I completed this book.

– Steve Dimse, M.D., who sleeps even less than I do, for answering those nitpicking questions, always at midnight.

– Chris Larson of Microsoft, for clarifications on MS-DOS versus PC-DOS.

– Lee Tydlaska, for helping me to understand data communications, baud rates, parity, and all that other mumbo jumbo.

– Pam Bell, for being such a patient sounding board for so many random thoughts. Chaos then usually evolved into coherency.

– And especially my wife, Hallie, for tolerating the long hours, and my son, Charlie, for graciously terminating games of "Burger Time" or "Frogger" whenever I needed the Compaq, even if an all-time high score was at stake.

Preface

I wish that I could have written a witty, entertaining novel about DOS, so that you all could have lots of fun reading this book. Unfortunately, the subject matter does not lend itself to a lot of humor, and for most people, getting over the hurdle of learning DOS is serious business.

So, the book is pretty dry, full of explanations and examples, not jokes and stories. It will probably be great reading for one of those nights when you're having trouble getting to sleep. Even better than a warm glass of milk. I am confident, though, that it will be a useful tool, helping you to master one of the more difficult aspects of microcomputers, the operating system.

I write for everyday users, for all of you who have little or no interest in what goes on inside a computer, but who do want to know how to operate it, how to get some effective use out of it. There are no explanations of interrupt vectors or DOS calls (nitty-gritty programming stuff), nor are there comparisons to CP/M and Unix, two other operating systems. Someone who is into Unix and interrupt vectors does *not* need a book like this. Someone who wants to know about subdirectories, paths, and copying *does* need it, because the IBM documentation leaves a lot to be desired.

So jump right in. The early chapters will get you started, and the later ones will make you a sophisticated user. When you've finished this book, you'll know what you're doing!

Introduction

What Is PC-DOS?

IBM PC-DOS is the operating system that most people use with IBM PCs and XTs and is the only one that works with the IBM PCjr. Although most of it was written for IBM by Microsoft, several of the programs were added by IBM, including COMP, DISKCOMP, DISKCOPY, FDISK, GRAPHICS, MODE, and TREE.

Microsoft retained the right to license the same operating system to other computer manufacturers. When it is licensed, it is called MS-DOS. MS-DOS and PC-DOS are essentially the same. Most manufacturers of IBM PC-compatible computers try to come up with equivalents to those programs that IBM added to PC-DOS.

DOS (it rhymes with "boss") stands for Disk Operating System. The term DOS as used throughout the book really refers to IBM PC-DOS. The vast majority of descriptions and explanations hold for MS-DOS as well, but I make no guarantees. This is one area where people with compatibles may be in for a surprise about just how compatible their systems really are.

This book covers DOS Versions 2.0 and 2.1. The only difference between these two is that only 2.1 can be used on an IBM PCjr. Version 2.1 has no new commands and no changes in the commands. From a user's perspective, the two versions are the same.

DOS 2.0/2.1 is very different from the earlier versions, 1.0 and 1.1. Space is allocated differently on the disk, and many new, powerful commands have been added to 2.0/2.1 that are not found in the older versions. Because the disks are set up in a different way, if someone starts up the system with an old

version, a 2.0/2.1 disk cannot then be used. Fortunately, if the computer is started with DOS 2.0/2.1, disks created under 1.0 or 1.1 *can* be used, so users can access or copy files to other disks.

How to Use This Book

Before starting in on Chapter 1, anyone who is not familiar with the keyboard should read Appendix F, "About the Keyboard."

It is not absolutely necessary to read the chapters of this book in order, although it is better to do so, because new terminology is explained as soon as it is introduced. After that, the new terms are used freely.

Each chapter has a succinct summary at the end. For someone who has a little bit of experience with computers and has an urgent need for the information, it is possible that the summary may provide enough of an explanation to get by for the moment. Very probably, though, it will not.

After you read the book carefully, Appendix A, a summary of commands covered, can be used as a reference, and it shouldn't be necessary to spend a lot of time rereading the text.

CHAPTER 1

Getting Started

A new computer user is often overwhelmed by all the different facets of a microcomputer. It's almost as though it's a big jigsaw puzzle. Eventually the pieces start coming together one by one, just as they do with a puzzle. One of the most important, and usually most difficult, aspects of the computer to master is the operating system.

Many people try to ignore the operating system and just use their application programs for tasks such as word processing and data base management. In the long run, though, it cannot be ignored. It must be mastered so that backup disks can be made, files copied from one disk to another, and so on.

1.1 What Is an Operating System?

Understanding an operating system helps bring together many of the pieces of the microcomputer jigsaw puzzle. Essentially, an operating system is a buffer between the user and the nitty-gritty level of the zeroes and ones that tell a microprocessor chip what to do. A translation process must take place for us to be able to type in a command like DIR and get a directory listing on the screen of the files on a disk,. The sequence of characters D I R must be translated into lots of zeroes and ones.

But it is not enough to translate the letters D I R. A particular process must be linked to those letters. The process includes reading the directory of the disk to find all the filenames, placing the filenames on the screen in a certain order, and so on. So when we type in DIR and hit the ENTER key, the operating system takes in this input from the keyboard, and says, "Aha, this person wants a directory listing. Call up routine 26 and show 'em what's on the disk."

But what is "routine 26"? Well, it's a sequence of programming commands that is part of the operating system – lots of zeroes and ones telling the computer every single step it must go through to place a directory listing on the screen.

So we see that the operating system is a program itself. Actually it is a conglomeration of lots of programs. But what's a program, anyway? It's nothing more than a list of instructions to the computer's microprocessor chip. The instructions are executed, one after another, in a very carefully specified order.

If we had to issue commands to computers in sets of zeroes and ones, it would render computers useless to most of us. So operating systems were devised to act as go-betweens. A command such as ERASE or COPY can be typed in, and the operating system will convert it to the zeroes and ones that the microprocessor chip needs.

What types of things does an operating system normally handle? A lot of it can be called "housekeeping" – things such as getting a directory listing of what is recorded on a disk, saving (recording) a file onto a disk, erasing a file from a disk, or making a copy of a file from one disk to another.

The operating system is also very important to people writing programs for the commercial market, because without operating systems they would be able to produce their programs for only one brand of computer. Let's look at a hypothetical example that will show why this is.

Brand XYZ computer sends data out a serial port via memory location 12. A serial port is a connector device, usually on the back of the computer, that can have a printer hooked to it, or a modem. Modems are the devices that allow computer signals to be sent over phone lines; they translate the signal into the type of signal that the telephone lines can handle.

Brand ABC computer is different. It sends data to its serial port via memory location 26. Which location does the commercial program writer refer to in order to route data to the port? To the rescue come the operating systems. Remember, an operating system is a buffer. It has a specific code in it for routing data to the port. Rather than refer to the hardware memory location 12 or 26, our programmer talks to the operating system instead, telling it to take care of directing output to the serial port. This is done by referring to the code for SERIAL PORT, whatever that might be.

Now, before brand XYZ computer is released on the marketplace, the manufacturer must make small changes to the operating system so that the code that refers to the SERIAL PORT location gets tied to memory location 12. The ABC computer people do the same, but tie the code for the SERIAL PORT to memory location 26. For anyone interested in some of the internal details, this is usually referred to as the BIOS portion of the operating system (for Basic Input/Output System). The advantage of having an operating system is that the programmer does not have to worry about each different computer's specific hardware, but can instead just tell the operating system to send to the SERIAL PORT. This way, the program runs on both computers, with no modifications required.

In a nutshell then, the operating system acts as a buffer between the machine language level of the computer and both users and programmers. For programmers, it is an interface to the hardware that saves them from worrying about some very complicated details, such as how to get to a serial port or a printer, or how to get data recorded on the disk bit by bit, track by track. From the user's point of view, it is a bunch of commands that are available to take care of lots of housekeeping details. From a more technically correct view, it is programming that instructs the microprocessor chip how to react to the commands that we type in directly, like DIR, or the commands a programmer has placed in an application program.

1.2 Booting Up

One of the most important things to understand about the DOS is that it does NOT reside permanently in the chips of the computer. In every computer there are always some programming instructions stored permanently in what is called the ROM, the Read Only Memory. In the jargon, we say that these instructions are "burned in," or "hardwired in." Everything else goes in the RAM, the Random Access Memory, often called the "working memory." But programming and data only stay in the working memory as long as the computer is on. When it is turned off, *everything* in there disappears!

For the IBM PC, XT, PC*jr*, and compatibles, this includes the DOS. So when we power up, the computer needs to be able to find the DOS recorded on a disk. Then the DOS can be brought into the computer's memory. Doing this is called **booting up**. This term comes from the early days of microcomputers, when starting up a micro was difficult and required going through a lot of initial programming each time. This type of programming was thought of as "pulling the computer up by its bootstraps."

Then operating systems came along to take care of all the tedious detail of starting up. One small program stored in the ROM, the permanent memory, tells the computer how to go find the bulk of the operating system on the disk. Programmers often refer to this little program as the "bootstrap" program, or the "bootstrap loader." Thus the term "booting up."

Starting up by turning on the power and loading DOS is called a cold boot. If the computer is already on and we want to restart for some reason, we can use the Control, Alternate, and Delete keys to reboot. This is called a warm boot. To do this, hold down the Control key and the Alternate key, and then push the Delete key one time. The standard notation for this is Ctrl-Alt-Del.

This warm boot sequence will work for the IBM PC and all the look-alikes that have identical keyboards. Some brands of computers that use MS-DOS do not use this sequence to reboot. Their sequence might be something like Ctrl-F20. The documentation accompanying the computer will tell how to do it. As you read the documentation, look for the term "reboot," "restart," or "reset."

When we boot up, the computer goes to the disk, finds the DOS, and brings it into the memory. The programming that makes up the DOS then takes

control of the computer's operations. This may also be referred to as "booting DOS into the computer."

Notice that this description has been vague about which disk has the DOS recorded on it. That's because it depends on the type of computer. If it has one or more floppy disk drives, then it will always boot up on drive A. That means it will always go looking in drive A for a disk that has the DOS on it. If the computer has a hard disk, though, other possibilities exist.

IBM XTs normally have one floppy drive, which is drive A, and the hard disk, which is drive C. If there is no disk in drive A when the system is booted, then the XT will look for DOS on the hard disk, drive C. This assumes that the hard disk has been installed properly, and that the DOS has been recorded onto it. But if there is a disk in drive A, the XT will try to find the DOS on that disk.

With other hard-disk systems, the startup procedure varies. Davong hard disks require that a special boot disk be placed in the number one floppy disk drive which then causes the system to switch over to the hard disk to find DOS. Most users of hard disks will want to configure their systems so that the computer will boot up from the hard disk.

If the disk being searched does not have DOS recorded on it, what happens next depends on the brand of the computer. IBMs go into the programming language called BASIC. A message appears on the screen that says BASIC is up, and the prompt for BASIC is displayed:

```
Ok
```

Compaqs come up with a message that says:

```
Non-System disk or disk error
Replace and strike any key when ready
```

The next chapter explains how DOS is put onto new disks.

When DOS is ready to go, the first thing it does is ask for the date. The screen displays:

```
Current date is Tues 1-01-1980
Enter new date: __
```

The date is entered in the form "month-day-year." For example, to enter January 18th, 1984, we would type in 1-18-84 and hit the ENTER key.

Next the DOS asks for the time. It displays on the screen something like this:

```
Current time is 0:00:44:31
Enter new time:____
```

The time is entered in the form "hour:minute:second." It is not necessary to put in the seconds, so to put in 8:30 a.m. we would type in 8:30 and hit the ENTER key. The system uses a twenty-four clock, so 3:34 p.m. would be entered 15:34.

It is possible to skip over putting in the date and time by hitting the ENTER key alone, but it is not a good idea. Everyone should definitely get in the habit of putting in at least the date. This date is used by DOS to keep track of when files are created.

After the time has been entered, a message from DOS to us appears on the screen. The form of this message is:

```
A>
```

1.3. Disk Drive Considerations

In the A> prompt, the letter A tells us that the system currently is on disk drive A, and the right-angle bracket (>) tells us that DOS is ready for a command. The > "prompts" us that DOS is ready, and so it is called the **system prompt**. This combination is usually referred to as "A-Prompt."

A computer system may have one disk drive, or it may have several. Each disk drive has a name, so that we can refer to each separately. The first disk drive, the one that normally boots up when we turn on the computer, is named:

```
A:
```

The colon *is* part of the name. A second disk drive is named B:, a third, C:, and so on. Using these names, we can tell DOS which disk drive to access.

We can shift over to another disk drive simply by typing in its name. For example:

```
B:      (and the ENTER key)
```

This changes us over to drive B. This is shown on the screen by the fact that the system prompt is accompanied by the letter B.

```
B>
```

To switch back to drive A, we just type in A: and hit ENTER.

So what is the significance of being signed on to a particular drive? If we don't specify otherwise, DOS commands will refer to this disk. An example is the DIR command, which produces a directory listing of what is on a disk. If

A-Prompt (A>) is on the screen (telling us we are currently on drive A), and we type in DIR and then hit ENTER, we get a directory of what is on the disk in drive A. If B-Prompt (B>) is on the screen, and we type in DIR and hit ENTER, we get a directory listing of what is on the disk in B.

Throughout the book, the disk drive that is being used will be referred to as the "current" drive. In some documentation it is referred to as the "default" drive – if we don't tell the computer otherwise, commands will apply to this drive "by default." The notion of default is a very important one in dealing with computers, and one that new users often have some trouble with. In the case of disk drives, every time we use the terminology "current drive," the words "default drive" could be substituted.

Switching the current drive from one drive to the other is one thing; actually swapping disks in the disk drive is another. Whenever there is a DOS prompt on the screen, a disk in a drive can be removed and replaced with a different disk. This is a nice feature of DOS. Some operating systems are very finicky about when disks can be swapped and what must be done to tell the system that a disk has been changed.

Switching disks is okay when the DOS prompt is there, but we must be careful about doing it at any other point. For example, when WordStar, a leading word processing program, is used, if disks are changed during the editing of a document, it can wreak havoc on the directory listing of the disk. Similarly, with dBASE II, a leading data base management program, if a disk is removed while the program still has a data file open and the file has been updated, that file can be ruined. (For now, don't worry about the technical details of what it means for a file to be open.) To use the hip jargon here, we say "it will trash the file."

1.4 Summary

The operating system is a buffer between us and the inner workings of the computer. It takes care of a lot of housekeeping details for us. All we have to do is learn the various commands that it makes available, and it will carry out these commands.

Starting up the system is called booting up. When we start up, the computer goes looking for the DOS on a disk so that it can bring it into the memory. DOS first asks for the date and time to be entered, then puts its prompt on the screen. The prompt (A> or C>) is a message to us that DOS is ready to accept a valid command.

We can change from one drive to another by typing in the name of the drive, which will include a colon; for example, A: or B:. When we change from drive A to B, the DOS prompt changes accordingly (from A> to B>), telling us that B is the current drive.

CHAPTER 2

Preparing Disks

2.1 What Is Formatting?

Disks usually come from the manufacturer totally blank. They are not prefabricated to work in any particular computer. Each brand of computer has its own pattern for placing data on disks. Until the disk has had this kind of pattern imprinted on it, it cannot be used with any computer.

The process of setting up a disk for a particular computer is normally called **formatting**, although sometimes it is referred to as **initializing**. This process places on the disk the pattern or "format" required by that computer. Once this has been done, the disk can store data.

It is very important to realize that the first step that takes place in formatting is to erase the entire disk. If the disk is brand new, this really has no effect. But, if it is a disk that has been used before and still has data on it, the data will be erased. Formatting is very similar to taking a cassette tape, running it back to the beginning, and recording something new. The old material is erased in the process. This is one way to clear off an old disk and reuse it.

Once the disk has been wiped clean by the formatting program, the tracks and sectors are placed on it. A track goes all the way around the disk; there is no spiraling effect as there is on a phonograph. There are 40 tracks on each side of a disk prepared by DOS 2.0 or 2.1. The tracks are numbered from 0 to 39. Disks can be set up to be used on just one side, or both, depending on the type of disk drives that the computer has. Some can use only one side of a disk.

On floppy disks, each track is subdivided into 9 sectors. On hard disks, each track is divided into 17 sectors. Each sector can hold 512 bytes, or 512 characters. So, when both sides of a floppy disk are used, as they normally are,

there are 80 tracks with 9 sectors each, giving 720 sectors. This number multiplied by the 512 bytes in each sector gives a total of 368,640 bytes of storage capacity on a disk.

The formatting process also includes a check of the disk for flawed areas. If any are found, they are partitioned off from the usable space on the disk. It's a bit like a fence being put around the bad area. If this happens, it is reported in the message that appears when the formatting is done. This message tells how many bytes of space on the disk are defective.

Once the pattern of tracks and sectors has been implanted on the disk, the formatting program, which is in control, checks to see if we specified that the DOS should also be recorded on the disk. If so, the DOS is put on the disk. If not, the formatting process is finished.

2.2 Formatting Data Disks, No DOS Included: FORMAT

A program must be used to format a disk. This program has to be available on one of the disks being used by the computer at the time. In other words, the programming that takes care of the formatting process is not always in the memory of the computer and thus is not automatically available to us. It must be located on a disk first and brought into the memory. Once this is done, the program can take over and do the formatting.

This process of getting a program from disk into the computer's memory and executing it is normally referred to as "running a program." The name of the formatting program is FORMAT. To run it, we simply type its name. Here is the basic procedure. The DOS disk is placed in drive A. The computer is turned on, the DATE and TIME questions are answered, and the A-Prompt (A>) comes up on the screen. The name FORMAT is typed in, followed by a space and the name of the disk drive that will contain the disk to be formatted, and the ENTER key is pushed.

FORMAT B: (and then <ENTER>)

It does not matter whether all of this is typed in uppercase or lowercase. The following would work in exactly the same way:

---

format b: (and then <ENTER>)

If the program FORMAT.COM is not on the disk, an error message appears:

Bad command or file name

If the FORMAT program is there, it is brought into the computer's

memory from the disk in drive A, and it then begins the formatting process. The first thing that happens is that a message appears on the screen:

```
Insert new diskette for drive B:
and strike any key when ready
```

Assuming at this point that the computer has two floppy disk drives, the new disk is placed in drive B. Hitting any key (typically the space bar) starts the formatting.

If the computer has only one disk drive, or if it is an IBM XT, which only has one floppy disk drive (plus the hard disk), then the command would be:

```
FORMAT A:
```

When the message to insert the new diskette in drive A: appears, the DOS disk is removed from the disk drive if it is there, and replaced with the new disk. Striking any key starts the formatting.

When the formatting is finished, the screen shows:

```
Formatting...Format complete

    362496 bytes total disk space
    362496 bytes available on disk

Format another (Y/N)?
```

Simply pushing the n key (for No) ends the process, and the A-Prompt reappears on the screen. If y (for Yes) is pushed, the message to insert the new diskette appears again. A new disk can be placed in the correct disk drive and the process started again.

The procedure described above does nothing but place the pattern used by DOS 2.0 or 2.1 on the disk. All of the disk's storage capacity is available. The disk does *not* have DOS itself recorded on it. Disks formatted in this way cannot be used to boot up from drive A. They are intended to be used for recording data. For this reason, they are called data disks.

2.3 Formatting Disks, DOS Included

In order to create a bootable disk, something must be added to the command that initiates the formatting process. An optional formatting "parameter" can be specified that says, "After placing the format on the disk, also record the DOS on it." Disks formatted this way can then be used to boot

up. Since they have the operating system (the DOS) on them, they are referred to as **system disks**.

To tell the computer to include the operating "system" on the disk, /S is added after the disk drive name. This says, "Include the system." Here's an example:

```
FORMAT B:/S      (and then <ENTER>)
```

The message to insert the new diskette and strike any key appears. When the formatting is done, something like this appears on the screen:

```
Formatting...Format complete
System transferred

362496 bytes total disk space
40960 bytes used by system
321536 bytes available on disk

    Format another (Y/N)?
```

Pushing the n key once ends the formatting process. Pushing the y key causes it to start up again.

We can see from the "System transferred" message that more than the formatting gets placed on system disks. A lot of programming gets recorded on them as well. Three different files are put on the disk. In a directory listing of one of these disks (obtained by typing in DIR B: and pressing the ENTER key), only the one named COMMAND.COM shows up. The other two are "hidden" files. Their names do not show up on a directory listing. These three files contain all the DOS programming that goes into the computer's memory when we boot up.

2.4 Some Formatting Options

The message displayed when formatting is finished says that there are 362,496 bytes available. This is slightly less than the total space on a disk that was calculated earlier in the chapter. That's because what is being reported here is how much space is available to the user. Some space gets automatically allocated to the directory of the disk. Let's discuss in more detail this issue of disk space.

Essentially a byte represents one character. Disk storage capacity is usually measured in thousands of bytes, and the symbol "K" is used to stand for 1000 (similar to "10K run" being used to mean a 10 kilometer run). We say that a DOS 2.0 or 2.1 floppy disk has 362K of storage.

This is the case for computers that have double-sided disk drives, as most do these days. This means that the disk drive unit can use both sides of the disk, because the drive has read/write heads on both sides. It also tells us that the disk drive is using nine sectors per track.

The first IBM PC disk drives were one-sided. A one-sided disk drive unit cannot use disks that are formatted for double-sided use. Furthermore, the previous version of DOS, called DOS 1.1, used only eight sectors per track. Thus, disks created on single-sided drives had a storage capacity of 160K.

To format a disk for one-sided use, the option /1 must be added when formatting. It tells the FORMAT program to just use one side. The command would be:

```
FORMAT B:/1 (and then <ENTER>)
```

When formatting is complete, the message on the screen about disk space says that the disk has 179,712 bytes, or as we normally say, 179K.

If the disk must also have DOS on it, the /S option must be included as well. The command would be:

```
FORMAT B:/S/1
```

(Reminder: It does not matter if this is typed in uppercase or lowercase.)

Notice that even though only one side is being used, there is still more space (179K) available than in the old days with DOS Version 1.1 and one side. That is because nine sectors per track are being used, versus eight sectors per track in DOS 1.1, so the same physical space can pack more information.

Because of this difference, anyone using DOS 1.1 will not be able to get at the data on a DOS 2.0 or 2.1 formatted disk. Under DOS 1.1, the computer expects to find eight sectors per track. When there are nine, things don't mesh. Fortunately, the reverse is not true. When using DOS 2.0 or 2.1, a disk created under 1.1 with only eight sectors per track can be used.

If it is really necessary to create a disk compatible with DOS 1.1, there is an option for this. Including /8 in the command tells the format program to use only eight sectors per track. If nothing else is specified, then the disk will be formatted as double-sided and will end up with a total capacity of 320K. If the disk must be set up to be one-sided (maybe it is to be mailed to someone who has a computer with one-sided disk drives), then the /1 option must also be included. As always, if the disk needs to have the DOS placed on it as well, then the /S option must be included. Depending on what is needed, the appropriate command would be one of these three:

```
FORMAT B:/S/1/8
FORMAT B:/1/8
FORMAT B:/8
```

A hard disk must be formatted just like a floppy disk, but there may be more to consider than what has been covered here. The documentation that comes with the hard disk will give the details.

Hard-disk storage capacity is measured in megabytes. One megabyte is 1000K. One double-sided floppy disk can hold 362K, while a 10 megabyte hard disk (usually abbreviated 10 MB) can hold 10,000K, or 10 million characters.

2.5 Putting DOS on Later: SYS

It is possible to put DOS onto a disk after formatting, but only under the right circumstances. One of the following circumstances must exist:

1. Nothing has been recorded on the disk yet.
2. Space was allocated on the disk for the DOS during the original formatting.

The way to set aside space on the disk during the formatting is to include the /B option. Why would anyone do this? Everyday users probably never would, but companies marketing software do. They do not own the rights to DOS. They would have to pay to include DOS on the disks they are selling to the public. Since they know we all have DOS anyway, they can reserve the space on the disk and let us put it on ourselves, using the SYS program. The SYS program must be on one of the disks in the computer before we can use it.

SYS might also be used if we forget to include the /S and realize it right after doing the formatting. Using SYS to put DOS on the disk is faster than going through the entire process of formatting again, including the /S.

If the command SYS B: is used, and the target disk in drive B already has something recorded on it, the system responds with the message:

No room for system on destination disk

This happens because the three DOS files must be on the very first part of the disk, and if something is already there, the SYS command cannot work properly.

So what do we do if we have data files and program files on a disk that does not have DOS on it and realize to our chagrin that it needs to be there? Since there is no way to get DOS onto that disk, we must format a new disk with DOS on it (include the /S option), and then copy everything from the old disk to the new, using the COPY command (not DISKCOPY!). Copying is covered in Chapter 4.

2.6 Naming the Disk Volume

Each disk, including a hard disk, can be given a name. This name appears

at the top of a directory listing every time the DIR command is used. The name is placed on the disk by including the /V option (V for Volume). When this option is included, the formatting takes place first, then the following message appears:

```
Formatting...Format complete

Volume label (11 characters, ENTER for none)?
```

The name is entered at this point. It can be no more than 11 characters long. Any character that can be used in a filename can be used in a volume name, but in addition, volume names can have spaces (each space counts as one of the 11 characters).

If the name DOS BOOK were entered, then the directory would show:

```
 Volume in drive B is DOS BOOK
```

The volume name can be entered in uppercase or lowercase. It always gets converted to uppercase, as do all filenames. If a disk is formatted without using the /V option, it is not possible to go back later and put in a volume name.

2.7 Protecting Data

Since formatting wipes a disk clean first, there is always the danger of formatting the wrong disk and wiping out programming or data that is very valuable. It happens to a lot of people. There are various steps everyone can take to protect against this.

All 5¼-inch floppy disks have a notch cut out on the right side, as you look at the disk with the label up. This notch must be open for data to be recorded (saved) onto the disk. If it is covered, then a device inside the disk drive senses this and will not allow anything to be recorded on that disk.

In computer jargon, recording on a disk is referred to as "writing to" the disk. If the **write-protect notch** is covered, then the disk is said to be **write-protected**, and nothing can be recorded on it.

If the disk is one that contains programming, like a word processing program or a spreadsheet program, and it will never have data recorded on it, it is a good idea to put a write-protect tab over the notch. Tabs usually come with disks when they are purchased. They are normally small silver rectangles. Transparent tape will work too.

If there is a write-protect tab on the disk, it cannot be formatted, since the formatting process "writes" (records) to the disk. If the disk is going to be used to store data, such as a document created with a word processor, it cannot have a write-protect tab on it.

So how can we protect against loss of data, from formatting or any other calamity, with these disks that cannot be write-protected? The answer is *make backups*! As long as the information is stored on at least two different disks, if something happens to one of them, the information will still be on the other. Chapter 11 covers backing up in detail.

Another protection against formatting the wrong disk (one that has data we want to save) is to double-check which disk is in the drive before allowing the formatting process to actually start. Here's the procedure: Place the DOS disk in drive A and a new disk in B. Enter the command FORMAT B:.The message "Insert the new disk and hit any key to start" comes up on the screen. Before hitting a key to start the formatting, open drive B, slip the disk part of the way out, and check to see that it is indeed a new disk, not a data disk that was left in there from a previous work session. Check this even if the new disk was placed in drive B only moments earlier. The reason: It should become a habit. The one time it saves you from wiping out an important disk, you'll be very glad you made it a habit!

Owners of hard disk systems must be especially careful to avoid formatting the hard disk, because formatting wipes the disk clean. *Everything* on the hard disk is erased. It has happened all too many times. What happens typically is that people will switch to the hard disk (make the hard disk the "current" drive), then type in the command FORMAT with no disk drive name after it, just FORMAT. When no drive is specified, the program just assumes that this means to format the current, or default, disk drive, which is the hard disk. Hit any key to start the process and whoosh! Bye-bye to everything on the hard disk.

It really is incredible that the makers of DOS allow this to happen so easily. They should have built more protection against such a possibility into the system. There are two ways to avert total disaster. First, delete the FORMAT program from the hard disk. If it is not on the hard disk, a floppy disk that has it will have to be used, and there is much less chance that anyone will type in FORMAT followed by the hard disk drive name (A, B, or whatever). Second, back up the data on the hard disk to floppy disks. This is imperative, because formatting is just *one* way to lose data.

2.8 Summary

Formatting wipes a disk clean and then imprints on the disk the pattern for recording data that is used by that brand of computer. The FORMAT program prepares disks on computers that use DOS. To execute this program, we type in its name, followed by the disk drive which contains the disk to be formatted:

```
FORMAT B:
```

The user must decide before issuing the FORMAT command whether to

place the DOS on the disk at the same time. Including /S at the end of the command tells the FORMAT program to go ahead and record the DOS on that disk.

Various other options are possible with FORMAT. Including /1 tells the program to format only one side of the disk so that it can be used in computers with single-sided disk drives. The /8 option is tacked on to make the disk compatible with an older version of DOS, the 1.1 version, when necessary. That version used only eight sectors per track instead of the current nine sectors per track. The /V option allows us to give each disk a "volume" name. The name can be up to 11 characters long and, unlike filenames, it may include spaces.

The SYS command puts DOS onto a disk after it has been formatted, but only if the disk was specially prepared for this situation, or if it has no data files or extra programs recorded on it yet.

Since the FORMAT command can wipe out data on a disk, we must use it very carefully. A write-protect tab can be placed over the open notch on the side of a floppy disk to protect against erasing by accident, but data disks cannot have write-protect tabs on them. Instead, use backup disks with the data reduplicated on them.

CHAPTER 3

Directories, Filenames and Wildcards

This chapter covers the DIR command, filename constraints, and the DOS wildcard characters. The DIR command is used to get directory listings of the files on a disk. Files may contain programs that do word processing or some other application, or they may simply contain the data generated by an application program. Either way, files have names that are recorded on the disk.

Filename constraints are important because they come into play no matter what type of program is being used. When a document is created using word processing, the name of that document must conform to the DOS constraints. A filename has different parts to it. There is always the main part of the name. The other parts – the extension, the drive specification, and the path – may or may not be needed.

Wildcards are used for things like telling DOS we want to erase all files that begin with the letters MEMO (for example MEMO1, MEMO2, MEMONEW, MEMORIAL, etc.), or for copying all files ending in .COM from one disk to another.

3.1 Getting a Directory Listing: DIR

Typing in the command DIR and then hitting the ENTER key produces a listing of the files on the disk in the current drive. (Remember, if the screen shows A>, then the default or current drive is A. If it shows B>, then it is B; C> indicates C.) This directory also shows how much space each file takes up on the disk, as well as the date and time that were in the system when the file was created.

The DIR command can be used whenever the DOS prompt is on the screen. This is different from when the FORMAT command can be used. To

do formatting, the FORMAT program must be on a disk that is in the computer at the time. Typing in FORMAT and then hitting the ENTER key causes the computer to go looking for that program on the current disk. Any DOS command that *must* be available on disk in order to be used is called an **external** DOS command.

DIR is an **internal** command. When we boot up, much of what we call DOS goes into the computer's memory. The ability to recognize the command DIR is included in what is always in the memory, as long as the computer remains turned on. For this reason it is called an internal command. External commands are really programs that must be retrieved from disk because the programming to take care of their particular function is not in the computer's memory at all times.

If a disk in drive B was not given a name during the formatting process, its directory listing might look like this:

```
 Volume in drive B has no label
 Directory of B:\

COMMAND      COM        17792    10-20-83   12:00p
AUTOEXEC     BAT           18     1-13-84    4:51p
D            COM         1536    10-05-82   12:00p
SAMPLES      BAS         3067    10-20-83   12:00p
ART          BAS         1920    10-20-83   12:00p
MUSIC        BAS        10291    10-20-83   12:00p
BASICA       COM        26112    10-20-83   12:00p
MEMO                     8665     1-25-84    1:02a
RESUME       OLD         2560     2-12-84    9:19p
RESUME       NEW         2667     2-12-84    9:32p
LOTUS                <DIR>        2-15-84   10:09a
CHECKS       DBF         4688     2-22-84    9:12p
BALANCE      PRG         4736     2-23-84   11:15p
SUMDEP       PRG         3584     2-25-84    7:53p
SUMTAX       PRG         1921     2-26-84    8:34p
POSTDEP      PRG         1288     2-26-84   11:45p
POSTCHK      BAK         1792     2-29-84    1:12p
POSTCHK      PRG         1855     2-29-84    2:22p
WS                   <DIR>        2-29-84    7:27p
TAXES        NDX         3584     2-29-84    7:49p
TAXES        FRM          256     2-29-84    8:23p

      21 File(s) 190464 bytes free
```

Going across a line, we see the full name of the file, then the amount of disk space (in bytes) that the file takes up, then the date and time that was in effect when the file was recorded on disk. If no date or time was entered at the

start of the session during which the file was created, then the date will probably be 1-01-1980. The time will have started at 0:00, so times like 0:22 and 1:32 show up on the files.

The two listings that do not show any number for bytes used are subdirectories. This is indicated by the <DIR> that appears next to the name. Disks can be subdivided. Files can be placed in the different areas, which are called subdirectories. For floppy disks, which can hold a maximum of 362K, this may not be important. For hard disks that hold 10 MB (megabytes, or 10,000K) or more, it is critical. A hard disk with one big directory would become unmanageable very quickly. Putting certain types of files in one subdirectory area and other types in other subdirectory areas is really a necessity.

The number of bytes a file takes up on the disk is useful information. Usually, though, we don't deal in numbers of individual bytes. Instead, it is commonplace to deal in thousands of bytes. In Chapter 2, we saw that 1,000 bytes is normally abbreviated 1K. Thus the file TAXES.NDX, which has 3,584 bytes, takes up 3K of space. To be a little more precise, we could say it takes up 3.5K. BASICA.COM takes up 26K. It is fairly easy to get in the habit of ignoring the last three digits on the screen display of space available and just adding the term K to the amount.

At the bottom of the listing, the total number of files is shown, as well as how much disk space remains. This last is very important. A floppy disk formatted in DOS 2.0 or 2.1 (with no special options) can hold 362,496 bytes, or 362K. In the above listing, the disk has 190,464 bytes (190K) of space remaining.

This number is not the result of adding up all the bytes of space occupied by the files actually shown in this listing and then subtracting that from 362K. The files that show space taken (there are 19 of them) take up a total of 98,332 bytes. Take this away from 362,496 and we get 264,164. Yet the disk only has 190,464 bytes of space left. This tells us that the two subdirectories, LOTUS and WS, have files in them that take up a total of 73,700 bytes. The 98,332 bytes of the 19 files combined with 73,700 bytes in the subdirectories produces a total of 172,032 bytes of space used up. This number, subtracted from the total space of 362,496, gives us 190,464 bytes, or 190K of space remaining.

3.2 Parameters: Add-ons to Commands

When the command DIR is entered, the listing produced is for the disk in the current drive. There are two ways to get a directory listing of another disk. One is to change over to that disk drive. This is done by simply entering the drive letter, followed by a colon. To move from drive A to B, we type in:

---

b: (and then <ENTER>)

B-Prompt (B>) will show on the screen. Then a DIR command gets a directory of the disk in drive B. But, if it would then be necessary to type in A: to move right back to drive A, there is an easier way.

A directory listing of a disk other than the current drive can be obtained by typing in DIR, then a space, then the other drive letter, followed by a colon. From drive A (A> on the screen), the command to get a directory of B is:

```
dir b: (and then <ENTER>)
```

The space after the DIR absolutely must be there!

The DIR command, like almost all DOS commands, can have "parameters" tacked onto it. **Parameters** are additional items that can be included with a command. They may be optional items, or they may be mandatory. In Chapter 2, we saw various options, or parameters, that go with the FORMAT program, like /S, /V and /8.

With commands such as DIR and FORMAT, it matters a lot whether or not a disk drive is specified along with the command. If none is specified, action is taken on the current drive. With DIR, a directory of the disk in use is the most commonly desired result. On the other hand, with FORMAT, typing in FORMAT (plus the ENTER key, of course) and then hitting any key to start the process could be disastrous. This would cause the computer to try to format the very same disk that contains the FORMAT program, which is probably not the desired effect. Thus, FORMAT will usually be followed by a disk drive specification, which is just one of several possible parameters.

3.3 Directory Options

Directory listings go zooming up the screen so fast even a super speed-reader could not read them. Moving up the screen like this is called **scrolling**, as if a scroll contained the text and was being rolled up, causing the text to go up the screen.

It is possible to stop this scrolling temporarily by using a combination of the Control (Ctrl) and Number Lock (NumLock) keys. The Control key works the same way a shift key does. It is held down as the Number Lock key is pressed once. When this sequence, normally written as Ctrl-NumLock, is used while the scrolling is taking place, the listing halts immediately. Hitting any key makes it continue.

Two options available with the DIR command provide easier ways to view the whole directory. They are /W and /P. The W stands for Wide and the P for Pause.

The /W option lists the files in five columns across the screen. When this is done, the disk space used and the dates and times of creation are not included in the listing. The command to get a directory of the disk in drive B, while located on drive A, with the Wide option, is:

```
dir b:/w
```

Using the /W option on the same disk whose directory listing appears earlier in this section would produce a directory that looks like this:

```
Volume in drive B has no label
Directory of B:\

COMMAND  COM     MUSIC    BAS     RESUME    NEW   SUMDEP   PRG   POSTCHK  PRG
AUTOEXEC BAT     BASICA   COM     LOTUS           SUMTAX   PRG   WS
D        COM     MEMO             CHECKS    DBF   POSTDEP  PRG   TAXES    NDX
SAMPLES  BAS     RESUME   OLD     BALANCE   PRG   POSTCHK  BAK   TAXES    FRM
ART      BAS

      21 File(s) 190464 bytes free
```

The obvious benefit of using this option is that the names of all the files can be seen at one time. The drawback is that the listing does not show the disk space occupied by each file or its date and time of creation. The /P option can be used to see this extra information without having to worry about when to start and stop the scrolling.

When the /P option is included, a directory listing goes scrolling up the screen as usual, but it stops when the first file in the listing reaches the top of the screen. At the bottom of the screen this message appears:

```
Strike a key when ready . . .
```

Hitting any key causes the listing to continue. The listing continues up the screen until a new screenful is displayed, then stops again, showing the same message. This goes on until all files have been shown.

The Pause and Wide options are not mutually exclusive. If there are so many files on a disk (or in a subdirectory area) that there is more than one screenful even when file names are displayed five columns across with the /W option, then the /P option as well. It does not matter which comes first, the /P or the /W, but each must be preceded by the /. Examples:

```
dir b:/w/p
dir b:/p/w
```

3.4 Sending Directories (or Anything Else) to the Printer

When there is a printer hooked up to the computer, it is a simple matter to send a directory listing to that printer as well as to the screen. Entering a

Control-P tells the DOS to open up the "port" to the printer, so that whatever is being sent to the screen will also be sent to the printer. "Control-P" means "hold the Control key down and push the P key once." This is normally abbreviated Ctrl-P. Sometimes it is abbreviated ^P.

The term **port** is computer jargon for a hook-up device on a computer, normally on the back. A parallel port is the connection used to hook up a parallel-type printer. A serial port is used to hook up serial-type printers or modems. See Chapter 12 for more on devices that connect to a computer.

To shut off the flow to the printer, enter another Ctrl-P. When the same command turns some function on and off, it is called a toggle. This comes from the analogy to turning a toggle switch, like a light switch, on and off. So, Ctrl-P is a toggle.

The Ctrl-P can be entered either before or after typing DIR. If it is entered first, then the letters DIR will appear on the paper along with the rest of the directory listing. If it is entered after the letters DIR but before hitting the ENTER key, the letters DIR do not appear on the paper.

Another command that does exactly the same thing as Ctrl-P is Ctrl-PrtSc (say Control-Print-Screen). This means hold down the Control key and push the key marked PrtSc once. "PrtSc" stands for Print Screen. A Ctrl-PrtSc sends the same command as Ctrl-P to DOS; that is, "Open the port to the printer."

The tricky thing about using the Ctrl-P or Ctrl-PrtSc command is that nothing appears on the screen when it is entered. Because of this, a common problem is that people tend to hold down the P or the PrtSc key too long, and the repeating capability of the keyboard is activated. If the key is held down just long enough to send two Ctrl-Ps, then nothing will happen, because the command is a toggle. The command will have been turned on, then immediately turned off again.

There is one other way to send something to the printer. At times, whatever is showing on the screen may be worth preserving on paper for record-keeping purposes. To get the contents of the screen printed out, a Shift-PrtSc is used (pronounced Shift-Print-Screen). This means "hold down one of the shift keys and push the PrtSc key once." This command causes what is called a screen dump. It "dumps" the contents of the screen to the printer. As long as the screen has characters on it, not graphics of some sort, then any printer will be able to handle the printing of those characters. Only certain dot matrix printers can print out graphics when the Shift-PrtSc command is used. See Chapter 13 for details on the GRAPHICS command.

The Shift-PrtSc command can be very handy. For example, when a particular sequence of DOS commands has been used and something important has been learned, rather than taking notes on a piece of paper, you can perform a screen dump. There it is, a history of the sequence on paper. Or, if someone wants to show a friend exactly what the screen looks like at a certain stage of using some program, the Shift-PrtSc can be used to put that on paper.

3.5 What are Control Characters?

Control characters generally do not show up on the screen, but they are taken in by the computer from the keyboard just as other characters are. The general term "control character" refers to any character that is entered by holding down the Control key and then pushing that character. Let's delve into what goes into the computer when a key is pushed.

When we push just the key that has both a 3 and a # on it, a certain code number goes into the computer. This code number is linked to the symbol that we call "three." When we hold down the shift key and push the same key, the code number associated with the # symbol goes in. When we hold the Control key down and push the same key again, a third code number goes in. Holding down the Alt key (short for Alternate) and pushing that same 3 key produces a fourth code number.

The code number that is associated with each character is standardized in the microcomputer industry. Every brand of microcomputer uses the same code number for a particular character. What is used is the American Standard Code for Information Interchange. It is always referred to by the acronym ASCII (which is pronounced like passkey without the p). The code number (in a decimal-based counting system) for the ASCII character 3 happens to be 51. For the # symbol it is 35. Appendix D contains a chart of all the ASCII characters and their numbers.

Thus, the reason we normally see nothing on the screen when we enter a control character is that there is no symbol, like 3 or #, associated with that ASCII code number. If something does appear on the screen, then the particular program in use must be causing this display. Control characters are sometimes displayed – in word processing programs, for example.

3.6 Filenames

There can be several parts to a filename. Besides the main part of the name, there are **extensions**, **drive specifications**, and **paths**. The DOS documentation does not have a special way of referring to the central part of the name, but we will call it the "corename."

The corename consists of one to eight characters, and there are many characters that are not allowed. Here are some that are banned:

? * . = < > ; : [ ] + " ~ (Space)

All letters and numbers are allowed, as well as these characters:

$ & # @ ! % ( ) - _ { }

Probably the easiest way to stay out of trouble is to remember that letters of the alphabet, numbers, the dash (-) and the underline (_) are allowed. Above all, remember that a space cannot occur in a filename, *anywhere*!

There will always be a corename, but an extension may be optional. An extension is separated from the corename by a period and can have up to three characters. The three are restricted to the same characters allowed in a corename. In addition, certain extensions have a special meaning to DOS. Here are some examples of valid filenames:

memo	resume
memo.old	resume.new
fy83	fy_84
ltr.1	ltr.2
12-83	12-83.bak

One thing that tends to confuse many people is that the period does not show in directory listings. Despite this visual incongruity, the period is indeed part of the filename, and must be included when a filename is being used with commands like COPY or ERASE.

Here are some examples of invalid filenames:

Name	*Error*
memotojoe	Nine characters
fy83 new	Illegal character: space
memo.old2	Four characters in extension
old+new.23	Illegal character: +

Certain extensions are reserved for special uses:

Name	*Use*	
.COM	A command file	(see Chapter 7)
.EXE	An exec file	(see Chapter 7)
.BAS	A program in BASIC	(see Chapter 7)
.BAT	A batch file	(see Chapter 8)

These do have special meaning to DOS and should not be used in regular filenames created with word processing or other similar programs.

So far we have seen two parts to the filename, the *corename* and the *extension*. At times, the disk drive specification will have to be included in the name. "Disk drive specification" means the name includes information about

which disk drive the file is found on. For example, if a file named WORDS is on drive A, it may be necessary to refer to:

```
A:WORDS
```

Examples of when this may be needed will be shown with the COPY and DEL/ERASE commands, as well as in the next section on partial directories. The disk drive specification always comes before the corename. Remember, there are *never* any spaces in a filename, no matter what that filename may include.

Filenames may also include their path. Paths are explained in detail in Chapter 6. Here, all we'll say is that a path indicates which subdirectory a file is in. The path comes after the drive specification, but before the corename. Some examples of full-blown filenames that include paths are:

```
a:\lotus\budgets\fy84.wks
b:\dbfiles\roster.dbf
```

Although the notions of path and subdirectory have not yet been explained, we can see now that "filename" means more than just a corename of up to eight characters. Instead, the filename may also have an extension, and it may specify the drive and path too.

3.7 Wildcard Characters in Filenames

When using the DIR, COPY, or DEL/ERASE commands, it is often useful to be able to refer to a group of filenames that have something in common. For example, we might want to copy from one disk to another all files that have an extension of .BAS (that would be all the files that are programs written in the language called BASIC). We might want to erase (delete) from a disk all files that start with LTR and then have any other characters in them. This kind of thing can be done by using **wildcards** in filenames.

There are two wildcards: the Single Character wildcard, which is the question mark (?), and the Anything wildcard, which is the asterisk (*). When the Single Character wildcard is used, it stands in for any character. Thus the filename:

```
LTR?
```

will match up to filenames that have the characters L T R and then any other character (or no character). All of the following would match:

```
LTR1        LTR2        LTR3
LTRA        LTRB        LTR
```

So, if all of these files happened to be on one disk, the command:

```
ERASE LTR?
```

would cause all of them to be erased from the disk. The following files would not be matched and so would not be erased:

```
LTR12       LTRXX       LTR1A
LTR.OLD     LTR.NEW     LTR1.22
```

The ? can be used to stand in for one character only. If we enter:

```
ERASE LTR??
```

then the following would be erased:

```
LTR1        LTR2        LTR3
LTRA        LTRB        LTR
LTR12       LTRXX       LTR1A
```

but:

```
LTR.OLD     LTR.NEW     LTR1.22
```

would not be erased. A Single Character wildcard can match up to a void, as the second ? does for the name LTR2. But if no extension is specified, then it means that the filename matched must have no extension. To include filenames that do have extensions, some extension must be entered. For example:

```
ERASE LTR??.???
```

This command will erase from the disk all of the names used above. Including three question marks as an extension makes the above command say: "Erase all files that start with the letters L T R, and then have nothing, one more character, or two more characters, followed by no extension, or any extension, whatever it might be."

We can, in fact, have a sequence of up to eight question marks, like this:

```
ERASE ????????.BAK
```

This would cause all files ending in .BAK to be erased, but there is an easier way to do this! The Anything wildcard is designed for just these kinds of situations.

Using the Anything wildcard says: "Match up to any sequence of characters, including no characters." So, the above command is more easily written:

```
ERASE *.BAK
```

The command:

```
ERASE LTR??.???
```

can be entered like this:

```
ERASE LTR*.*
```

This command can be verbalized as: "Erase all the files from the disk that start with L T R and then have any sequence of characters, including none, up to the period, and then have any extension, including none."

It is important to understand that the scope of the Anything wildcard (the *) does not cross the boundary between the corename and an extension. The command:

```
ERASE *
```

would only erase files that have no extension. To match up to all files, the Anything wildcard must be used on both sides of the period:

```
*.*
```

This is often called "star dot star," simply because that is easier to say than "asterisk period asterisk." The term "wildcard" is used because of the similarity with playing poker, with certain cards declared "wild." For example, if one-eyed jacks are wild, then either of the one-eyed jack cards can be used to match up with any set of cards. If the player has two kings and a one-eyed jack, he or she is considered to have three kings, because the one-eyed jack can be used as anything.

3.8 Partial Directories: Using Wildcards

We can combine the use of the DIR command with our knowledge of what the wildcard characters do in order to produce directory listings that show only specific files.

First, the DIR command can be followed by an explicit filename. This tells the DOS to check to see if that filename is on the disk. Here's an example:

```
A> dir format.com
```

If the file is indeed on the disk, then here is what we see:

```
 Volume in drive A has no label
 Directory of A:\

 FORMAT      COM       6912     10-20-83    12:00
 1 File(s)   60416 bytes free
```

If the file is not on the disk, then we see this:

```
 Volume in drive A has no label
 Directory of A:\

 File not found
```

This type of check is not restricted to the current disk. If the current drive is A, and we need to see whether a particular file is on drive B, we have two options. One is to switch over to drive B, by entering B: followed by ENTER, and then use the DIR command. But, alternatively, we can stay on drive A and issue the following command;

```
dir b:format.com
```

This checks to see if the FORMAT.COM file is on drive B. If no drive specification were included after the DIR, the current drive would be checked. By including the B: in the name of the file here, we tell DOS: "Check and see if this file exists on the disk in drive B."

If, for example, we are on drive A and need to check for a file called BUDGET.WKS, and we must check a subdirectory on drive B called LOTUS, the command is:

```
dir b:\lotus\budget.wks
```

Wildcards can be used to get what might be called "partial" directories. A partial directory is one that lists only files with certain characteristics, such as those ending in .BAS, or beginning with LTR.

Rather than putting an exact filename after DIR, we enter a filename that includes one or two wildcards. For example, to get a listing of all the files on the current drive that are programs in BASIC, this command can be entered:

```
dir *.bas
```

If we are working on drive A, to get a listing of all files located on drive B that end in .WKS (Lotus 1-2-3 worksheets all have this ending), the command is:

```
dir b:*.wks
```

The wildcards are really most useful with the COPY and DEL/ERASE commands. More examples of how they can be used are given in the sections where these commands are explained.

3.9 Summary

The DIR command is used to get directory listings of disks. The listings include the names of all files on the disk, how much space they take, and the date and time that they were created. A listing of a disk other than the one in the current drive can be obtained if DIR is followed by the drive name. Example:

```
DIR B:
```

There are two options that can be used with DIR. They are /W and /P. Including /W (for Wide) at the end causes the listing to be displayed in five columns, allowing more names to be on the screen at one time. The /P (for Pause) option displays one screenful, then halts and waits for the user to hit any key. One more screenful is then displayed, and so on until all files have been listed.

The Ctrl-P toggle can be used to direct that the output of any command be sent to the printer as well as the screen. Ctrl-PrtSc does the same thing. Shift-PrtSc is used to "dump" the contents of the screen to the printer.

Filenames are made up of a corename and some other elements that may or may not be used, depending on the situation. These other parts of the name are the extension, the drive specification, and the path. The extension is separated from the core by a period. Unfortunately, this period does not show in a displayed or printed directory listing, even though it must be used to refer to a file when a command such as COPY is used. If all parts of a filename are used, their order is:

```
DRIVE-PATH-CORE.EXTENSION
```

The core of a filename can have a maximum of eight characters. There are many nonalphabetic characters that are not allowed in filenames. Most notably, spaces are not allowed. The extension can have one to three characters. The restriction on characters that applies to the corename is true for the extension as well. Certain extensions have special meaning to DOS and should not be used by us. They include:

```
.COM        .EXE
.BAS        .BAT
```

Wildcard characters are used with commands like DIR, COPY, and DEL/ERASE to match up to more than one filename. The Single Character wildcard is the question mark. It matches up to any one character. An example is:

```
DIR LTR?
```

The Anything wildcard matches up to any sequence of characters in either the core or the extension of the filename, but not both. It does not cross the boundary that separates the core and the extension. An example is:

```
DIR LTR.*
```

In order to refer to *all* files, *.* can be used. This is often called "star dot star" and has important applications with the COPY and DEL/ERASE commands.

CHAPTER 4

Copying Files

The COPY command is one of the most important of all the DOS commands. No matter what someone uses a computer for, it will be necessary to copy files from one disk to another. For example, this may be done to put all of the programs just purchased onto a working disk, or to make backup copies of data files, spreadsheet files, document files, and so on.

The basic syntax of the COPY command is straightforward. **Syntax** refers to what elements are contained in a command, and the exact order that these elements must be in. A generalized form of the COPY command can be stated like this:

COPY FILENAME DESTINATION

This command structure is deceptively simple. It is also not totally correct, but it will serve our needs 99 percent of the time. (Section 4.5 shows why it is not technically correct.)

In the previous chapter, we learned what makes a filename. It is not simply a few characters making up a name. Instead it is made up of the corename and other possible additions — the extension, the drive specification, and the path. Just as a filename is more than the corename alone, the notion here of DESTINATION is not restricted to just the name of a disk drive to copy to. In fact, the file being copied may not be going to a different disk drive at all but rather to a different subdirectory on the same disk.

Since this is such an important command, we will be looking at it in all its glorious detail.

4.1 Copying from the Current Disk to Another

Copying from the current, or default, disk drive to another is the simplest and most straightforward way to copy. If drive A is the current drive (A> shows on the screen), then to copy the file BUDGET.84 from the disk in A to the disk in B, the command is:

```
COPY BUDGET.84 B:
```

Since we are located on drive A, specifying BUDGET.84 as the filename is adequate. Since we did not specify a drive, the DOS will look for the file on the default drive, which is A. It would be perfectly correct, but redundant, to put in A:BUDGET.84. This notion of default location will permeate much of what we do in DOS.

Notice that we do not have to concern ourselves with locating COPY as a program on one of the disks, as we did with FORMAT. That's because, just like DIR, it is an internal command. When we boot up, and the DOS goes into the memory of the computer, the programming to recognize and carry out this command is part of what goes in. The COPY command can be used anytime the DOS prompt is on the screen.

Here are a few more examples (the current drive is indicated by the DOS prompt shown):

```
A> COPY CHKDSK.COM C:
B> COPY LTR12 A:
C> COPY FY84.WKS A:
```

4.2 Copying from Another Disk to the Current Disk

Once the whole concept of "which is the current drive" is really clear, it becomes apparent that it is simpler and actually takes fewer keystrokes to enter the command to bring a file to the current disk from some other disk.

What is useful here is the notion of what must be specified and what can be left unsaid because it is a default. Let's look at some examples. We are working on drive B and want to copy a file from A to B. The name of the file to copy is CHKDSK.COM. The following command would not work:

```
B> COPY CHKDSK.COM B:
```

Because only the core of the name is given, and no drive specification, it is understood that the file (CHKDSK.COM) is to be found on the current drive, drive B. DOS will look for the CHKDSK.COM file on drive B and will not be able to find it there. The disk drive specification must be included in the name of the file:

```
B> COPY A:CHKDSK.COM B:
```

This command will work, but it has some extra baggage on it. The destination is entered as B:, which is correct. But what happens if no drive is specified? The command looks like this:

```
B> COPY A:CHKDSK.COM
```

(Of course, all of these commands are followed by hitting the ENTER key.) Since no destination is specified, DOS will use the current disk drive as the destination. The current drive is B, and so the command works just the way we want it to. The command prior to this one, which includes the B: as destination, is perfectly correct, but redundant.

So, to copy a file from another disk to the current disk, it is only necessary to include COPY, and then the filename to be copied. But that filename must include its disk drive specification (it may also include the path that says what subdirectory the file is in – this will be covered in Chapter 6).

4.3 Copying Selected Files Using Wildcards

We said earlier, of the generalized form of the COPY command:

```
COPY FILENAME DESTINATION
```

this was deceptively simple. FILENAME alone says a lot. This name might need to include its drive specification or its path so that DOS knows where to find it. In the previous chapter we also saw that a filename may include wildcard characters and will then match up to more than one file. This fact can be used to copy selected files.

The key to coming up with the right wildcard specifications is to think, "What is the same about the names that we want to match to?" Whatever the names share, whatever characters they have that are identical, must be specified, and the wildcards must be used to match up to what is not the same from one name to the next.

Let's look at a sample situation. The following is a list of files that are on a disk in drive A.

ltr1	ltr15	memo	ball.bas
ltr2	ltr16	memo.new	art.bas
ltrold	ltrnew	memo.old	music.bas
ltr.83	ltr.84	memo5	pie.bas

Drive A is the current drive. In order to copy all the letters (abbreviated LTR), the Anything wildcard is used. It is not enough, though, to specify just LTR*, because two of them have extensions (LTR.83 and LTR.84). The command is:

```
A> COPY LTR*.* B:
```

What the files all have in common is that they start with LTR. After that, there is nothing the same. By putting in LTR*, we match all files whose corenames start with LTR, and then have any sequence of characters after LTR. By putting in .*, we include any file that starts with LTR in the corename and then has any sequence of characters in its extension (LTR.83 and LTR.84), or no extension at all (all the rest).

To copy all the memos, the command is similar. The files all start with MEMO. We can take a little bit of a shortcut here if we want to, by using the following command:

```
A> COPY ME*.* B:
```

It is not necessary here to include all of MEMO, because no other files begin with ME. If there were files such as METRIC or MEASURE, we would not use the ME* in the command. We cannot shorten the command to:

```
A> COPY M*.* B:
```

because this would also include MUSIC.BAS.

The files that end in .BAS are programs in BASIC. If we want to copy all of them, we note that they have one thing in common – the fact that they end with .BAS. The rest of the filename for each must be encompassed by a wildcard. The command is:

```
A> COPY *.BAS
```

The asterisk before the period says, "Match all files that have any sequence of characters in their corename."

4.4 Copying All Files

Copying all files from one disk to another is actually a simple matter. By using the sequence *.* ("star dot star"), we match all files:

```
A> COPY *.* B:
```

To copy all files from A to B while we are on drive B the command is:

```
B> COPY A:*.*
```

It is necessary to include the drive specification (A:) in the filename so that DOS will look for the files on A. Otherwise, with no specification DOS would look on the current drive, B. It is not necessary to include B: as the destination specification, although it would not be wrong to do so. It would just be redundant. No drive is specified as the destination, so DOS will send the files to whatever the current drive is, which in this case is B.

4.5 Copying a File and Changing Its Name

All the examples so far in this chapter include one important assumption – that we want the files being copied to keep the same name. However, it is possible to COPY a file and change its name at the same time. To do this, the new name must be included in the DESTINATION part. For example, to copy LTR.NEW from A to B and have it be named LTR.OLD on drive B, the command would be:

```
A> COPY LTR.NEW B:LTR.OLD
```

It is because of this capability that we must note that the generalized form of the COPY command given early in this chapter is not technically complete:

```
COPY FILENAME DESTINATION
```

To be more precise, the DESTINATION needs to convey that if a different name is included, this will cause the name of the file to be changed. We can express this as:

```
COPY FILENAME DESTINATION(-NEWFILENAME)
```

This ability to specify a new name leads to the next feature, reduplicating a file on the same disk.

4.6 Duplicating a File on the Same Disk

The COPY command can create a duplicate of a file that is recorded on

the same disk. To do this, the duplicate must be given a different name. The procedure is to tell DOS to COPY the file, changing its name in the process and making the destination the same disk that the file is already on. Working on drive A, to reduplicate FY84.WKS and name the duplicate FY85.WKS, the command is:

```
A> COPY FY84.WKS FY85.WKS
```

Because no drive specification is included in the destination part, the file will go to the current drive, which is A.

It is a common error to type in a COPY command and forget to put a destination when it really should be there. For example, we want to COPY the file FY84.WKS from A to B, and we forget the destination:

```
A> COPY FY84.WKS
```

No harm done; DOS will just come back with an error message that says:

```
File cannot be copied onto itself
        0 File(s) copied
A>
```

All that has to be done then is to reenter the command correctly.

4.7 Ensuring Accurate Transfers

Errors in this copying process are very rare. Nevertheless, there are provisions in DOS to safeguard against them. There are two basic ways to avoid problems. One is tell DOS to double-check everything as it is copied. The other is to compare what has been copied to the original.

There are two ways to tell DOS to double-check. This double-checking is called VERIFYing. The COPY command includes the /V option (V for VERIFY), which tells DOS to verify that each unit recorded, called a sector, has been recorded correctly. The option is tacked onto the end of the command line. For example:

```
COPY LTR.OLD B:/V
COPY B:*.*/V
```

When the VERIFY option is in effect, the copying will take longer (because more checking takes place).

The /V option remains in effect only for the copying specified. The second way to initiate double-checking is more lasting. Included in the list of DOS commands is VERIFY, which is used to turn the verification process on or off. The command VERIFY is followed by ON, OFF, or nothing. By typing in VERIFY all by itself we ask DOS to tell us what the current status is. The response appears on the line below. Example:

```
A> VERIFY          (and <ENTER>)
VERIFY is off
```

VERIFY is an internal command and can be used any time a DOS prompt is on the screen. Once verifying is turned on with this command, it remains in effect until turned off with a VERIFY OFF command, or until the computer is turned off.

Turning verification on affects more than just the COPY command. When it is on, any time DOS records something on the disk, whether initiated by a COPY command, a RENAME command, or a save in a word processing program, then the data is verified.

Because this always slows down the system, and because copying and other types of recording on the disk are really quite reliable, it is not necessary to use verification often. On occasion, with *very* important data, it may be desirable to use it.

The alternative to verifying is to compare files after they have been recorded. This is done with the COMP command. COMP is short for "compare." A general form of the command can be written like this:

```
COMP FILENAME1 FILENAME2
```

If the files are on the same disk (or in the same subdirectory), they will obviously have to have different names. A more likely situation is that we need to compare a copied file, which has the same name as the original. In this case, the names will be the same and still must be specified in full. In addition, at least one of them will have to include the drive (and possibly the path) specification. For example, we have just copied FY84.WKS from A to B. A is the current drive. To compare, the command is:

```
A> COMP FY84.WKS B:FY84.WKS
```

Wildcards may be used in the filenames in order to specify that a set of files, or even all files, be compared. Here are some examples:

```
A> COMP *.WKS B:*.WKS
A> COMP *.* B:*.*
```

COMP is an external command. It is not in the computer's memory at all times, so it must be available on disk when used. In the example above, it would have to be recorded on the disk in drive A. An easier but less reliable way to compare two files is to look at their entries in the appropriate directory listings and see if the files contain exactly the same number of bytes.

4.8 Summary

The COPY command is an internal one, so it can be used anytime there is a DOS prompt (>) on the screen. A generalized form of the command can be written:

```
COPY FILENAME DESTINATION
```

The notion of "filename" here encompasses the corename and extensions, drive specifications, and paths as well. To copy a file named FY84.WKS from A to B, with A the current drive, the command is:

```
COPY FY84.WKS B:
```

To copy from B to A, while A is the current drive, the name of the file to be copied must include its drive specification (so that it can be found) but the destination can be left off, since it is redundant:

```
COPY B:FY84.WKS
```

Wildcards can be used to have several files or even all files copied with only one command. For example:

```
COPY LTR*.* B:
COPY *.* B:
```

The first example copies to drive B all files whose names begin with LTR, while the second copies all files to B. Both copy from the current drive.

CHAPTER 5

Fundamental DOS Commands

This chapter covers basic DOS commands that every user needs to know. They are not of a large enough scope to require an entire chapter for each, but they are important to everyday use of the computer.

5.1 Copying Entire Disks: DISKCOPY

The COPY command reduplicates selected files to another disk or subdirectory area. Sometimes, though, we need to make an exact copy of an entire disk. Rather than format a new disk and then copy all files from some disk to that new one, it can be quicker to use the DISKCOPY command.

DISKCOPY copies the entire contents of one disk onto another. The command includes the formatting process, so the disk being copied to can be a brand-new, totally blank disk. Since the DISKCOPY process includes formatting, anything previously recorded on the disk will be erased.

DISKCOPY makes a mirror-image copy of an entire disk. It *cannot* be used to combine two disks. COPY must be used to do that. It is a fairly common error for people to format a disk, using the /S option to place DOS on the disk, then use DISKCOPY to make a working copy of the program disk they just purchased. Thinking then that DOS has been properly recorded on the disk, they cannot understand why the disk will not boot. The reason is that normally the manufacturers of commercial software do not include DOS on their disks because they would have to pay royalties to do so. When DISKCOPY is used, the target disk is first wiped clean, then formatted, and then everything on the source disk is copied to it. If the source disk includes DOS, then so will the target. If it does not, then the target will not either.

The structure of the DISKCOPY command is straightforward:

```
DISKCOPY SOURCE-DRIVE TARGET-DRIVE
```

The typical way to enter this command is:

```
A>DISKCOPY A: B:
```

This says, "Run the DISKCOPY program and copy the contents of the disk in A onto the disk in B."

DISKCOPY is an external command, which means that to be used it must be on a disk in the computer. Normally the working copy of the original DOS disk is placed in A, and the command is entered as shown above.

Since we need to be able to copy disks other than the DOS disk, which is already in drive A, the program puts a message on the screen, telling us what to do next:

```
Insert source diskette in drive A:
Insert target diskette in drive B:
Strike any key when ready
```

At this point, the working DOS disk is removed, and the disk we want to copy is placed in drive A. If the target disk is not already in drive B, it must be put in at this time. When everything is ready, hitting any key starts the copying process.

On the screen, a message appears:

```
Copying 9 sectors per track, 2 side(s)
```

If the target disk is not already formatted, then the message includes this line:

```
Formatting while copying
```

When the copy is done, the program asks if we want to copy any more disks:

```
Copy another (Y/N)?
```

To make a copy of another disk, we just push the y key for Yes, and the process starts again with the program telling us to place the source in A and the target in B. Answering with the n key for No causes the DOS prompt to reappear, and we are ready to go on to the next task.

If no disk drive specifications are entered along with the DISKCOPY command, then the program assumes that there is only one disk drive and will do a "one-drive" copy. This is the way to do it if the computer only has one disk drive (drive A, of course). After we type in DISKCOPY and hit the ENTER key, the program responds with:

Insert source diskette in drive A:

Strike any key when ready

After any key is pushed, the message about copying 9 sectors, 2 sides comes up. After the program has taken a portion of what is on the source disk, it stops and tells us to place the target in drive A and strike any key to continue. After we swap the disks and then hit any key, the data is recorded on the new disk. If the disk is brand-new, it will be formatted first. This swapping takes place a couple of times, and when everything has been recorded from one disk to the other, the program says:

Copy complete

Copy another (Y/N)?

Pushing the n key finishes the copying.

The DISKCOPY program has one option, which is /1. Including this at the end of the command tells the program to copy only the first side of the disk. This would be required if the computer being used has disk drives that can read only one side of the disk, or "single-sided drives," as they are called.

Once a copy of a disk has been made, if we want to be absolutely sure that the copy is exactly like the original, the DISKCOMP program can be used to compare the two. Like DISKCOPY, the DISKCOMP program is an external one. In order for it to be run, it must be on one of the disks in use.

Let's say we just made a copy of a very important data disk. The current drive is A. We put the working copy of the DOS disk in A and enter:

DISKCOMP A: B:

The program then tells us to place the first disk in drive A and the second disk in drive B, and to strike any key when ready. After we hit a key to start the

process, it tells us that it is comparing 9 sectors per track, 2 sides. If there are any differences, a message like this appears:

```
Compare error(s) on
Track 39, Side 1
```

If the disks are identical, then the message is:

```
Diskettes compare ok
```

When the comparing is done, we can compare more disks by answering y for Yes to the "Compare more?" question or finish by answering n for No.

For computers that have only one disk drive, we just type in DISKCOMP and hit ENTER. The program is set up to understand that this means that there is only one drive, and the disks will have to be swapped during the comparing process. The program says when to put in the first one and when to put in the second.

Like DISKCOPY, the DISKCOMP command can take the /1 option, which tells it to compare only the first side of the two disks. This is used when the computer has single-sided disk drives.

DISKCOMP should not be used when the COPY command has been used to reduplicate a disk. DISKCOMP compares the disks track by track. However, when files are originally recorded on a disk, the data in them may be scattered around the disk on different tracks; DOS records the data wherever it can find free space and keeps track of where everything is located. When COPY is used, the file is copied onto the target disk on consecutive tracks, which may be different from the way it is recorded on the source. In that case, COMP should be used to compare individual files (see section 4.7).

5.2 Checking Status and Remaining Space: CHKDSK

The CHKDSK program (short for CHECKDISK, because filenames are restricted to eight characters) reports on how much space remains on the disk, how many files there are, whether the files have been recorded using sectors on the disk that come one after the other (the most efficient way), and how much of the computer's working memory (RAM) is still free. It is an external program and therefore must be available on a disk in use.

To run this program, we just type in CHKDSK and hit the ENTER key. This assumes that the CHKDSK program can be found on the current drive and that it is the current drive that we want to check. The results of running CHKDSK will look something like this:

```
362496 bytes total disk space
22528 bytes in 2 hidden files
279552 bytes in 44 user files
60416 bytes available on disk

327680 bytes total memory
303104 bytes free
```

Ignore the last three digits and read a number such as 362496 as 362K. The two hidden files that take up 22K of space are part of DOS. They get put on a disk if the /S option is used when formatting is done. Those two, plus the COMMAND.COM file, are needed if a disk is to be bootable. Together, they take up 40K of space. Using the CHKDSK program is a way to look and see if the disk has DOS recorded on it.

CHKDSK was very important in previous versions of DOS (before 2.0) because the ending of a directory listing did not include the number telling how much disk space was left on the disk. This feature was added to version 2.0 and subsequent releases of DOS. So previously, CHKDSK was the only way to see how much space remained on a disk.

To have the CHKDSK program check the status of a disk other than the current one, the command to run it is followed by the drive specification. For example:

```
A> chkdsk b:
```

Since the DIR command now tells us how much disk space remains, the primary uses for the CHKDSK program are to check and see whether files are recorded in contiguous sectors on the disk (one after the other), and to see if the table that keeps track of all these sectors is correct.

Let's look at why these items may be of interest. The minimal unit of space on a disk is a sector, which contains 512 bytes (basically 512 characters). When the DOS must record a file on the disk, it looks for the first free sector, puts as much of the file as possible in that sector, then looks for the next free sector in which to place more of the file. Usually this will be the next sector, but not always. If the next sector already has data in it, then it is skipped.

As each sector is filled, it gets listed in the File Allocation Table, more commonly known as the FAT. This table contains a list of all the sectors used to record a file. The DOS then refers to this table. If both sides of a disk are being used, DOS actually deals in chunks of two sectors. These two sectors are called a **cluster**. Since the chunk may be one sector (for single-sided disks) or two sectors (for double-sided disks), the general term **block** is used to refer to this chunk, whatever size it is.

Tacking on a filename (which may include wildcards) tells the CHKDSK program to check to see if the files specified are recorded in contiguous sectors; that is, blocks that are one after the other on the disk. For example, with the DOS disk in A, and a disk containing Lotus 1-2-3 files in B, we could enter:

```
A> chkdsk b:*.wks
```

First, the usual details about total bytes, bytes available, and so on appear, and then, if all the files specified are recorded on contiguous blocks, the message is:

```
All specified file(s) are contiguous.
```

If any of the files use blocks that are not contiguous, these filenames are listed, along with a count of how many blocks are not contiguous. The message would look something like this, after the usual details on disk and memory space:

```
B:\PAYABLE.WKS
     Contains 2 non-contiguous blocks.
B:\SALESOFF.WKS
     Contains 5 non-contiguous blocks.
B:\FY84.WKS
     Contains 3 non-contiguous blocks.
```

When the blocks of a file are scattered all over the disk, the disk drive's read/write head must be moved from one location to another, rather than just continuing to read subsequent blocks. This becomes rather inefficient. If there are many files like this, it is a good idea to format a new disk, then use COPY to transfer all the files to the new disk. When this is done, the files are copied one after the other to contiguous blocks.

The second use for CHKDSK is to check the File Allocation Table. This table contains a list of all sectors used to record a file. On rare occasions, this table may lose track of some sectors. If there are sectors on the disk that have something recorded in them but are not listed in the table, they are considered "lost." If the CHKDSK program finds lost clusters, it asks if we want to fix the FAT. The message will look something like this:

```
006 lost clusters found in 003 chains
Convert lost chains to files (Y/N)?
```

Even though it asks, and goes through routines that look as though corrections will be made, CHKDSK will actually make changes *only* if the /F option was included in the CHKDSK command that was typed in.

If the /F option was included in the command, and we answer yes to the query, the CHKDSK program converts the lost clusters into files. The files will be named FILExxxx.CHK, where xxxx is a number between 0000 and 9999. The files can then be checked to see if there is anything worth keeping in them. If not, they can be erased from the disk.

It is very rare that problems arise with the FAT. It would not be unusual to operate a computer for years and never see any problems with lost clusters.

5.3 Deleting a File: DEL/ERASE

There are two commands to take a file off a disk. They are DEL and ERASE. The IBM DOS manual says that DEL is a "valid abbreviation for ERASE." This may be stretching the notion of "abbreviation" a little bit, but DEL is certainly easier to type than ERASE. For this reason most people prefer it. The general form of the command is:

```
DEL filename
ERASE filename
```

The two commands do exactly the same thing. These commands are internal ones. We never have to worry about finding them on a disk. They can be used anytime the DOS prompt is on the screen.

As we've seen with the COPY command, the notion of "filename" encompasses several things, not just the corename. The drive specification may need to be included. For example, if the current drive is A, and the file named FY84.WKS is to be deleted from B, the command is:

```
del b:fy84.wks
```

Wildcards may be used to delete groups of files, or even all files. For example:

del *.bak	Delete all files that have .BAK as an extension.
del b:fy*.*	Delete all files on drive B that start with the letters FY.
del *.*	Delete all files.

If the *.* (star dot star) wildcards are used, the system asks for verification:

```
Are you sure (Y/N)?
```

Pushing the y key for YES allows the deleting to take place. If there are no subdirectories involved, this will delete all the files on the disk. If there are subdirectories on the disk, all the files in the current subdirectory are erased, but files that are in other subdirectories are left intact. See Chapter 6 for more information.

It is worth mentioning that when a file is deleted with either DEL or ERASE, the contents of the file are not actually wiped off the disk. All that happens is that the filename is removed from the directory. It is possible to recover this data, but only with some kind of "un-erase" program. A program of this sort is called a utility program. Buying an un-erase program might be a worthwhile investment.

5.4 Renaming a File: RENAME

The RENAME command is used to change the name of a file. It is an internal command, so it will work all the time, no matter what disks are in the drives. It can be abbreviated REN. The general form of the command is:

```
RENAME oldname newname
```

This assumes that the current drive is the one with the file on it. Here are some examples:

```
rename sample.bak sample.1
ren memo.new memo.old
ren budget.old budget2
```

5.5 Listing a Text File on the Screen: TYPE

There are times when it is very handy to be able to display the contents of a file on the screen. If a file is a standard ASCII text file it can be listed on the screen with the TYPE command. If the file is a program file, what appears on the screen cannot be predicted but will usually be total garbage. Let's examine in detail here what a text file is. To do so, we must first understand what ASCII code is.

When we push the b key, what happens? We normally see the letter b on the screen. But how does it get there? A close look reveals that it is made up of a pattern of dots. Why did the computer use that particular pattern? Well,

each time we push a key, a code number is sent into the computer. The code number is converted into a particular pattern of dots associated with that number. The operating system then takes care of putting that pattern on the screen at the correct location.

The code numbers associated with the symbols on the keyboard are always the same, no matter which brand of microcomputer is used. That's because there is a recognized standard, the American Standard Code for Information Interchange, better known by its abbreviation, ASCII (pronounced like passkey without the p).

These code numbers range from 0 to 127, if you are using a decimal counting system. We are used to dealing with numbers in a decimal system, which is based on ten digits, but numbers in a computer are in a binary counting system, which is based on only two digits, zero and one. Counting from one to eight in such a system looks like this: 0 1 10 11 100 101 110 111.

Just to complicate life a little more, there is another possibility. In some instances, a base-16 counting system is used, which is called the hexadecimal system.

The computer deals in binary code, while we humans prefer decimal code, and some types of programming require hexadecimal code. The result is that occasions arise where there is a need to convert from one to another. The easiest way to do this is by using a chart. On such a chart we can look up the ASCII symbol, for example a capital K, under the column labeled ASCII. Under the DECIMAL column we find that its number is 75. Under the HEXADECIMAL column, we find the value 4B, and under the BINARY column we find 1001011. Appendix D contains just such a chart.

Anyway, back to ASCII text files. A standard text file is a file that contains nothing but the code numbers for ASCII characters. The characters can be displayed on the screen with the TYPE command. Certain characters have special meanings to the computer. For instance, Carriage Return (13) and Line Feed (10) cause a video display or printer to go to a new line, while Ctrl-Z (26) is used to mark the end of a text file. Text files normally contain sequences of characters that we are familiar with, each line ending with a Carriage Return and Line Feed, and the entire file ending with a Ctrl-Z. So, a text file might be displayed like this:

April 15th, 1984

Dear Joel:

How are things with you in your new home? Hope you are enjoying it. Don't forget your old friends here in San Diego. We'll have to get together sometime soon for some basketball or over-the-line.

Your buddy,

Gary

What is actually on disk looks a bit more like what follows. <CR> is used to mean code 13, a Carriage Return, <LF> means 10, a Line Feed, and <^Z> means 26, a Ctrl-Z.

```
April 15th, 1984<CR><LF><CR><LF>Dear Joel:<CR>
<LF><CR><LF>How are things with you in your new
home? Hope you are<CR><LF>enjoying it. Don't forget
your old friends here in <CR><LF>San Diego. We'll have to
get together sometime soon<CR><LF>for some basketball
or over-the-line.<CR><LF><CR><LF>Your buddy,<CR>
<LF><CR><LF>Gary<^Z>
```

TYPE is an internal command, so it will always work when the DOS prompt is on the screen. The general form of the command is:

```
TYPE filename
```

The notion of "filename" here is the same as for the COPY command. The name must include the extension if there is one, and a drive specification if the file is not on the current drive. It may also include the path, as explained in Chapter 6. Wildcards are not allowed in the filename, since only one file can be listed to the screen at a time. Some examples are:

```
type memo.old
type autoexec.bat
type b:memo.new
```

The display of the file that appears on the screen can also be routed to the printer. At any point before the ENTER key is pushed, a Ctrl-P can be typed in, and this will cause the output to go the printer as well as the screen. To stop the output of subsequent DOS commands from going to the printer too, another Ctrl-P is entered when the TYPE command has finished listing the file.

Many word processing programs create text files. Sometimes it is necessary to take a quick look at what is in the files on a disk, as the names may not be enough of a reminder. Having to start up the word processing program, load in a file, leave it, load in the next, and so on may be slow and painful. Instead, the TYPE command can be used to show very quickly what is in a text file. This would be done before the word processing program is started.

The contents will scroll up the screen rapidly. Ctrl-NumLock can be used to stop this scrolling temporarily. Hitting any key allows it to continue. Nothing else can be done to the file. It cannot be edited, only viewed. The scrolling can only go one way, from start to end. The listing or scrolling can be

aborted by entering a Ctrl-Break (this means hold down the Control key and push the Break key once – the Break key is also labeled Scroll Lock).

TYPE is also quite useful for checking the contents of batch files, as they are standard text files. Batch files are described in Chapter 8.

5.6 Miscellaneous: DATE, TIME, CLS

If nothing special has been set up on a disk (see Chapter 8 on AUTOEXEC batch files), when the computer is started up we are asked to enter the date, then the time. If we later discover that they have been put in incorrectly, they can be changed.

To change the date, it is only necessary to type in the command DATE and then hit the ENTER key. A message comes back saying what the date currently in memory is and prompting us to enter the new date. If a date was not entered during the initial startup, the message looks like this:

```
Current date is Tue 1-01-1980
Enter new date:
```

The correct date is then entered, and the DOS prompt reappears. This date is sometimes referred to as the system date.

The same thing can be done for the time. The command used to change the time is TIME. When this is entered, the current time is shown, as well as a prompt to enter a new time. After the new time is entered, the DOS prompt reappears.

The CLS command clears the screen of everything on it. When CLS is used, the screen is wiped clean, and the DOS prompt appears in the upper left corner.

Notice that in the descriptions of how to use these three commands, there has been no discussion about finding a program on disk. All three are internal commands, available anytime the DOS prompt is on the screen.

5.7 Summary

This chapter covered various fundamental DOS commands that are used quite frequently. DISKCOPY is the program used to make mirror image copies of entire disks. Its most used form is:

```
DISKCOPY A: B:
```

DISKCOPY has one option, /1, which causes the copying to take place on only one side of the disk. DISKCOPY should not be used the first time a commercial disk is copied unless the DOS is already on it.

The DISKCOMP program is used to compare two disks to verify that they are identical. The command is similar to DISKCOPY:

```
DISKCOMP A: B:
```

The CHKDSK program is used to check the status of a disk. It returns a message that contains information on how much total space the disk has, how much is used, and how much remains. If the filename or filenames (wildcards can be used) are included as parameters after its name, it can also tell us if files are recorded in contiguous blocks on the disk. For example:

```
CHKDSK B:*.WKS
```

The DEL and ERASE commands are used to get rid of files on the disk that are no longer wanted. To use them, we just type in the command followed by the name of the file to be deleted.

The TYPE command displays the contents of a text file on the screen. It cannot display the contents of a program file (one that has .COM or .EXE as part of its name).

The DATE and TIME commands allow us to change the system date and time. After entering these commands, we are shown the date or time currently in memory, and we are prompted to enter a new date or time. The CLS command simply clears off the screen and places the cursor in the upper left corner.

CHAPTER 6

Subdirectories

DOS Versions 2.0 and 2.1 have several important features that earlier versions did not. The most important is subdirectories. Because many microcomputers were being purchased with hard disks, it became necessary for the DOS to have a way to segment a hard disk into manageable areas. A floppy disk can hold only a relatively small number of files, so a way to subdivide the disk is not normally needed. A directory listing might take several full screens, but this is acceptable.

A hard disk, however, can hold thousands of files. Keeping all of them in the same area would slow down the DOS considerably, as it must search through all of them to find a particular file. Also, keeping track of what is there could quickly get out of hand. Subdirectories give us a way to find and keep track of disks more efficiently.

In many ways the subdirectories function like a bunch of separate little disk drives. They have their own names, and we can move from one to the other by using these names. We can copy files from one to another. When we are in a subdirectory, the DIR command gives a directory listing of only the files in that subdirectory.

6.1 Tree Structure and the Root Directory

Subdirectories are organized into a hierarchical tree structure. This means that they are not simply separate entities on the disk, but are related in a structure that can be diagrammed as a branching tree. The first level is referred to as the root directory. All subdirectories branch from this root directory.

The root level automatically exists on a disk once the disk has been formatted. Subdirectories must be created by the user.

Figure 6-1 shows three subdirectories one level down from the root level. Each subdirectory is to hold different types of files. DB is for data base files, WP is for word processing files, and SS is for spreadsheet files.

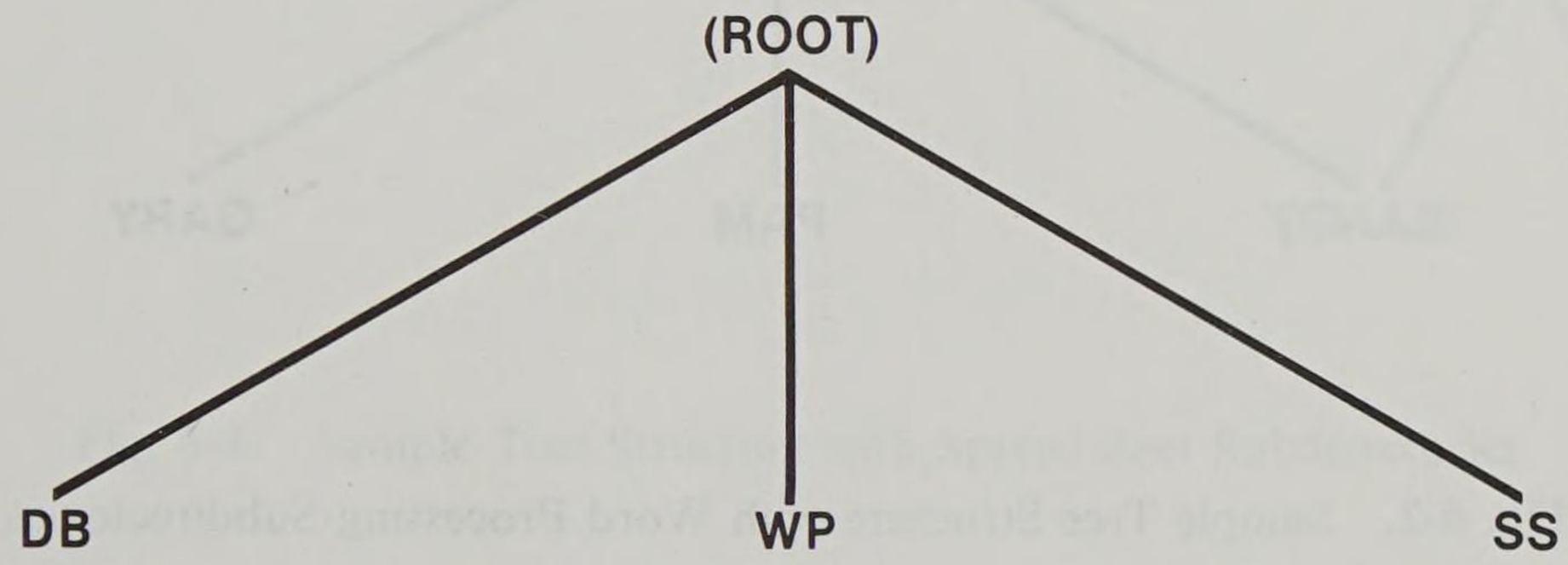


Fig. 6-1. A Simple Hierarchical Tree Structure

Lower level subdirectories can branch from each of these subdirectories. For example, if we need to have separate work areas for three different people under word processing, we can set up three subdirectories under WP, as shown in Figure 6-2.

Now Sandy, Pam, and Gary can have individual work areas for word processing. If only Sandy and Pam will be working with the data base applications, then we can set up a subdirectory for each of them under DB, as shown in Figure 6-3.

Only two people are going to use the spreadsheet application, so we give each of them a subdirectory, as shown in Figure 6-4.

Any subdirectory can have more subdirectories under it, and a subdirectory can have files as well as lower level subdirectories. The complete tree structure that has been designed here is shown in Figure 6-5.

6.2 The Path to a File

Pam is the first one to create some documents using word processing. The files are in her subdirectory, as shown in Figure 6-6. We can now identify a path to those files. In creating a path, we always start at the root directory and work along the various branches to the files. In this case, from the root level

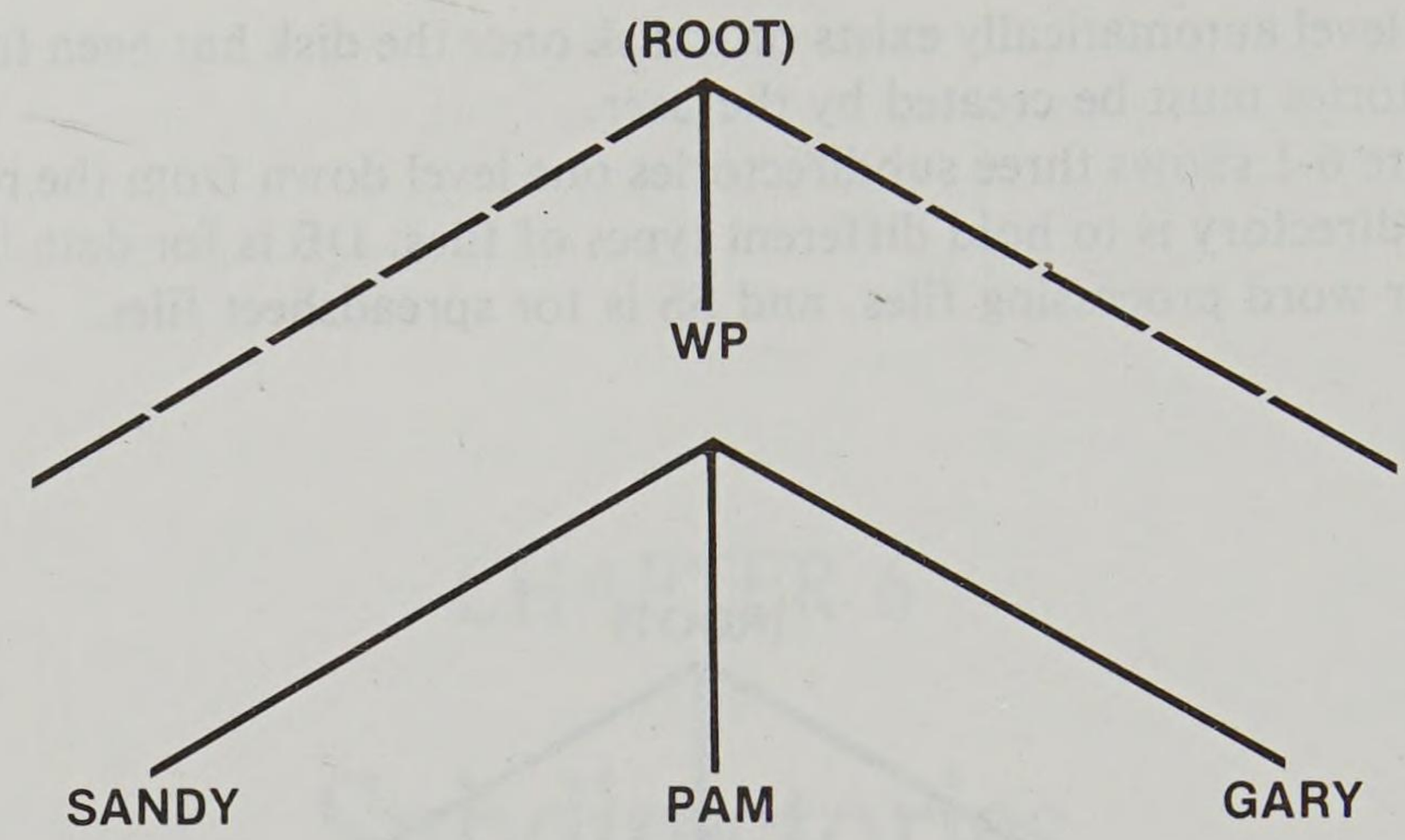


Fig. 6-2. Sample Tree Structure with Word Processing Subdirectories

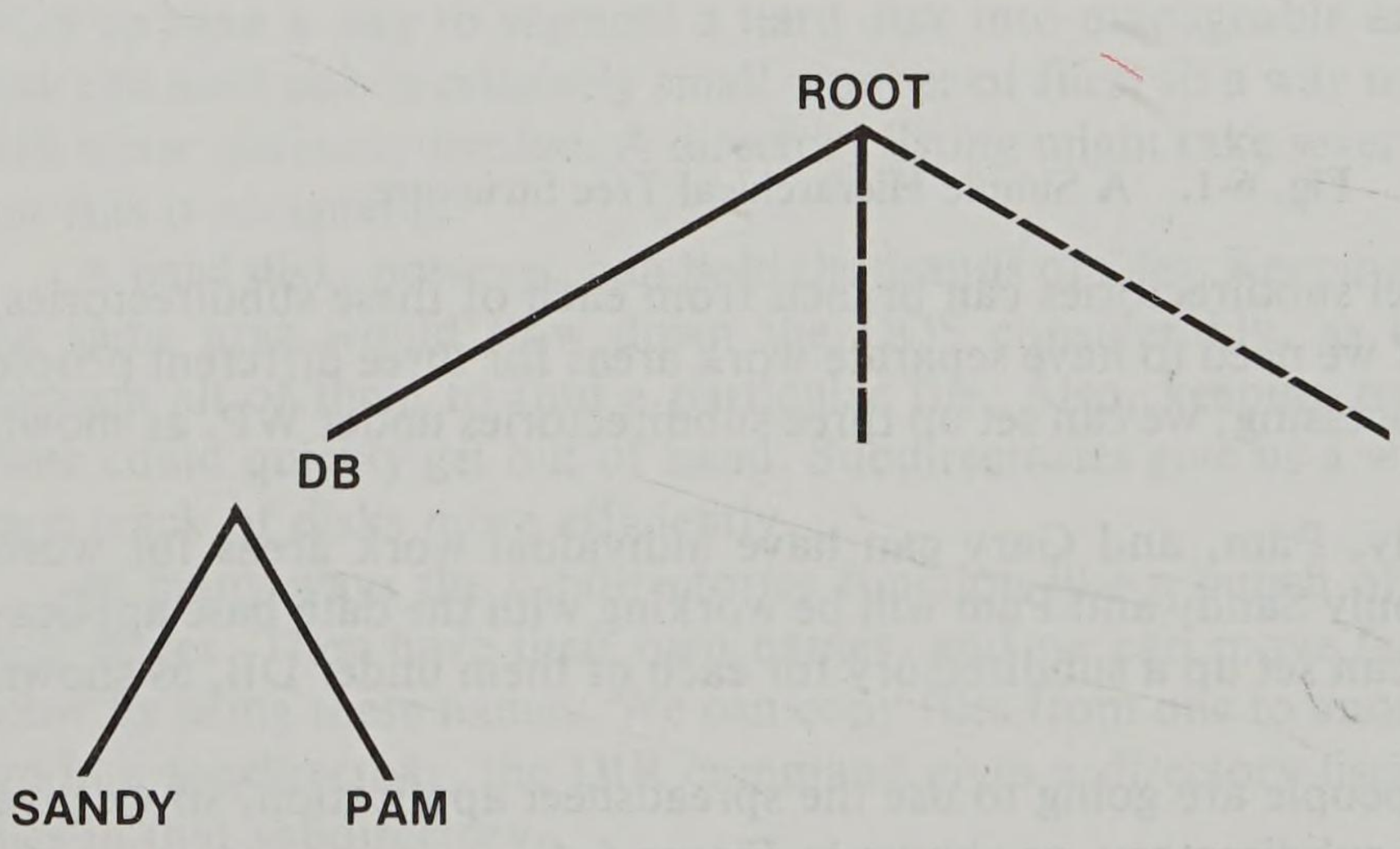


Fig. 6-3. Sample Tree Structure with Data Base Subdirectories

we go down to the WP subdirectory, then down one more level to the subdirectory named PAM. There we find the files in question. Traveling from branch to branch through the tree structure defines the path.

We need to have some way to describe the path, to write it down, so there will be an answer if someone asks, "What is the path to those files?" This is

Fig. 6-4. Sample Tree Structure with Spreadsheet Subdirectories

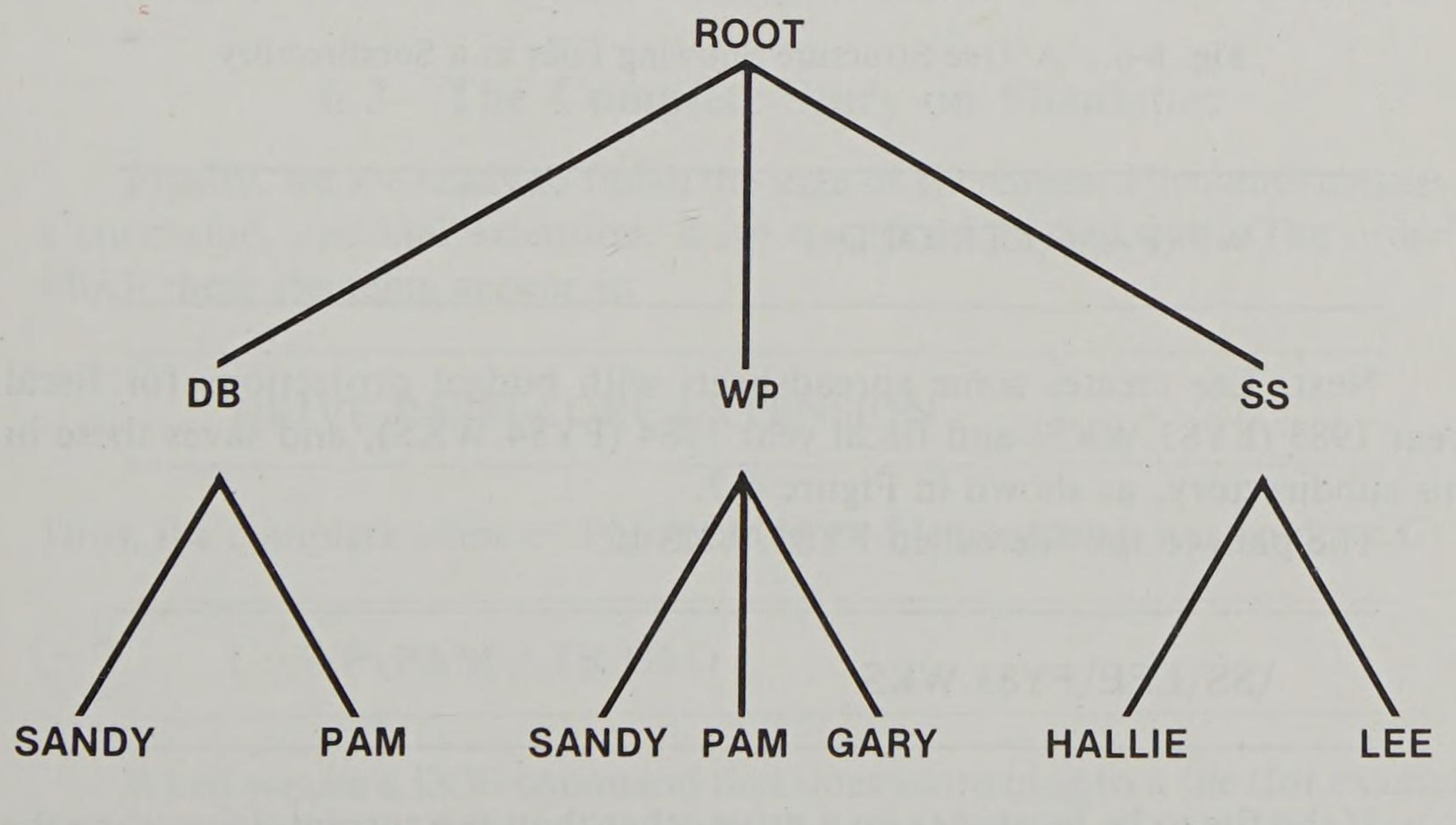


Fig. 6-5. A Completed Tree Structure

done by stringing together the names of the branches involved. Since all paths start at the root level, it is very important to know that the name for the root directory is the backslash symbol (\). That is the full name of the root directory. Subsequent subdirectory names are separated by other backslashes. No spaces are allowed in this path specification. Thus, the path to the file named LTR.OLD, including the filename itself, is:

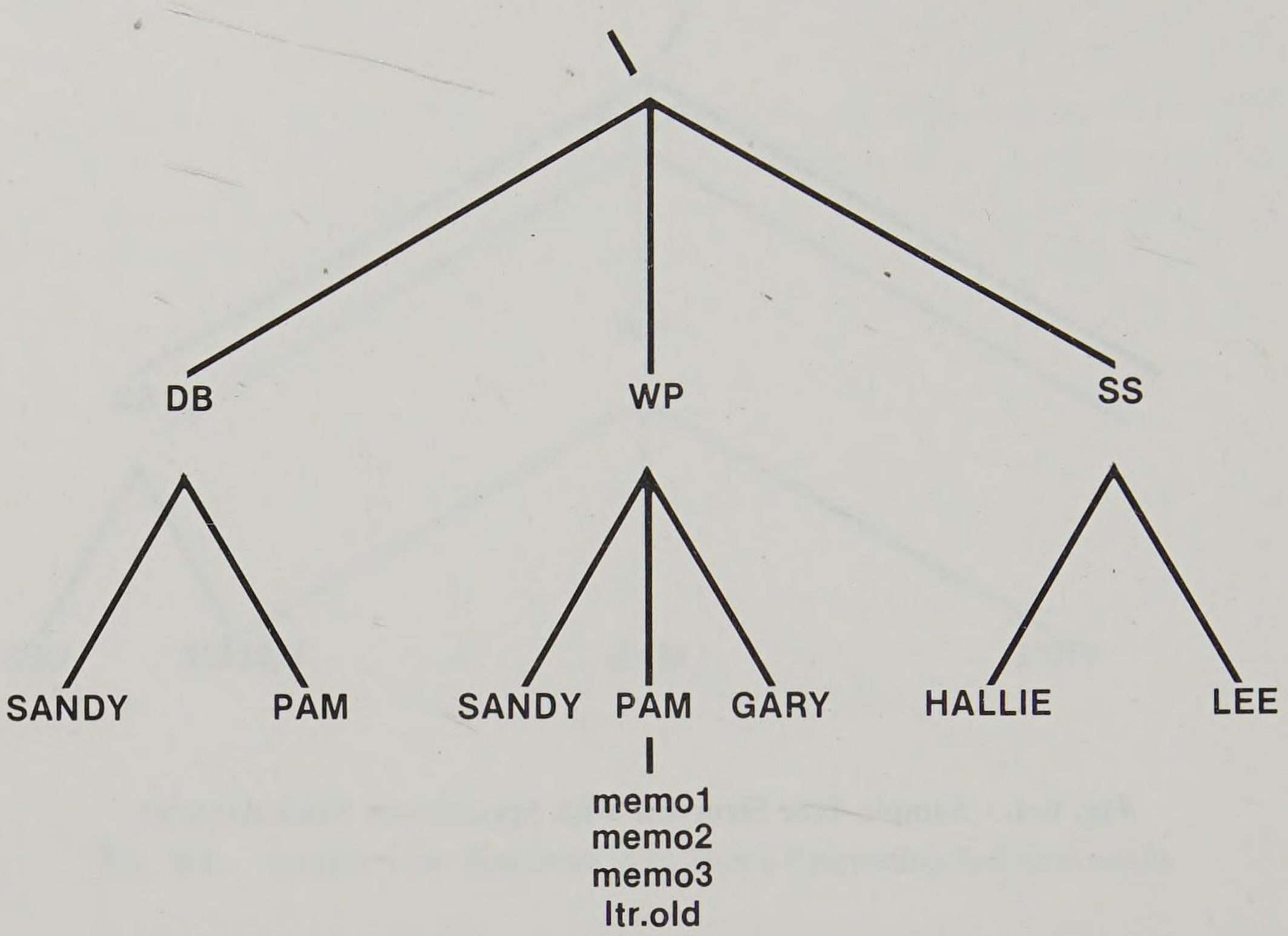


Fig. 6-6. A Tree Structure Showing Files in a Subdirectory

\WP\PAM\LTR.OLD

Next, Lee creates some spreadsheets with budget projections for fiscal year 1983 (FY83.WKS) and fiscal year 1984 (FY84.WKS), and saves these in his subdirectory, as shown in Figure 6-7.

The path to the file called FY83.WKS is:

\SS\LEE\FY83.WKS

If the file to be located is on a drive other than the current drive, then the disk drive specification must be included as part of the path. For example, if drive A is the current drive, and the tree stucture we have built here is on drive C, a hard disk, then the path to Lee's FY 83 worksheet is:

C:\SS\LEE\FY83.WKS

Notice that there are *no* spaces between any characters in these paths. Just as in filenames, no spaces are allowed. This is not coincidental; a "path" may in fact be part of a filename.

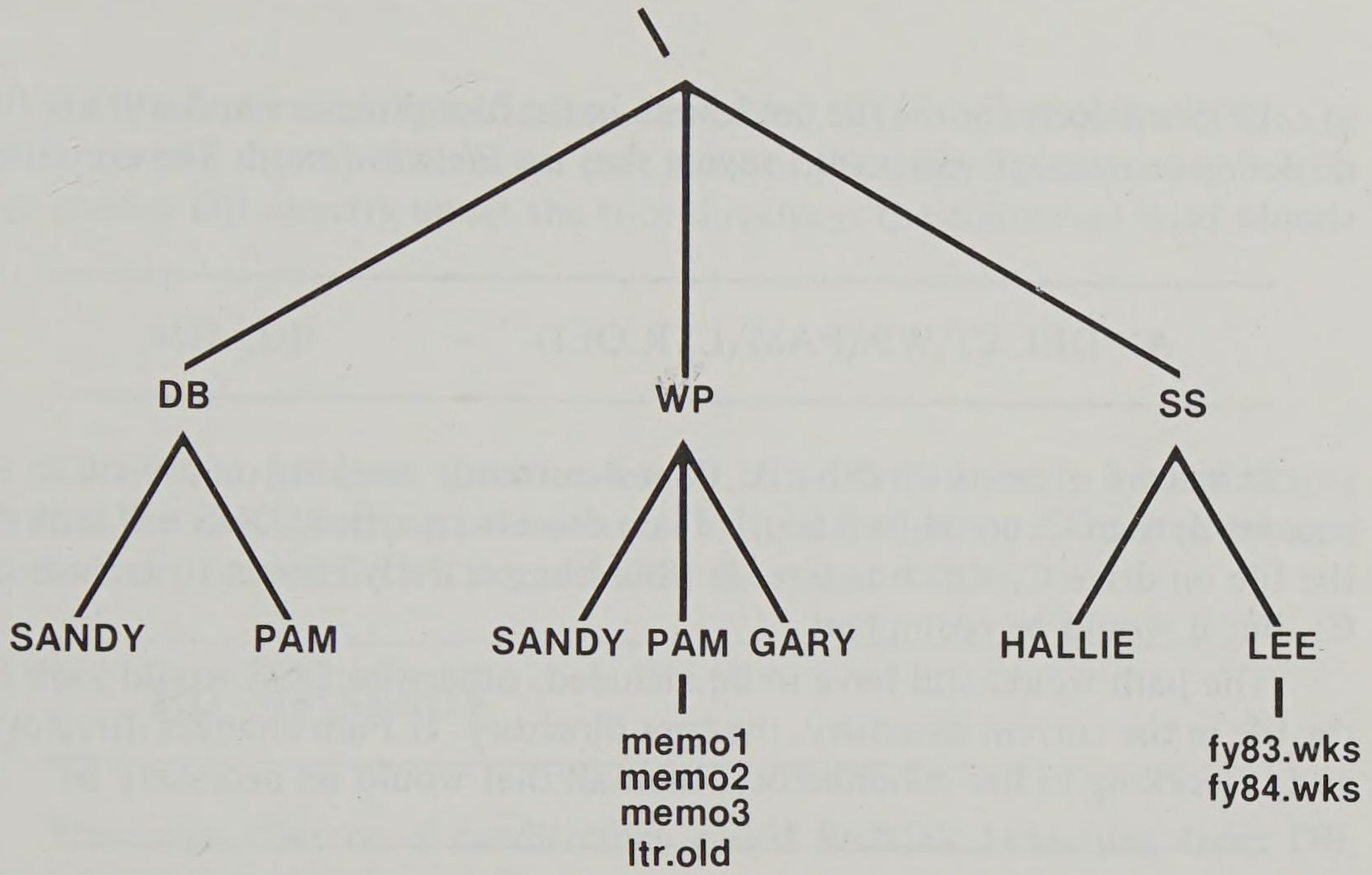


Fig. 6-7. A Tree Structure Showing Spreadsheet Files in a Subdirectory

6.3 The Complete Story on Filenames

Finally, we are ready to finish the saga of filenames. Filenames consist of a corename, optional extension, drive specification, and path. The order in which these elements appear is:

DRIVE-PATH-CORE.EXTENSION

Thus, the complete name of Pam's old letter file (assuming it is on drive C) is:

C:\WP\PAM\LTR.OLD

When we use a DOS command that does something to a file (for example, ERASE or COPY), DOS must be able to find that file. If it is in a subdirectory, then the path will have to be included in the name. If it is not on the current drive, then the drive specification will have to be included as well.

If no drive or path is included in a filename, then DOS assumes that the file is to be found in the current subdirectory of the current disk drive. For example, Pam wants to delete the file LTR.OLD, which is located in her subdirectory on drive C. The current drive is A. If she types in the following command, the file will not be found:

A> DEL LTR.OLD

DOS will look for the file on drive A in the root directory and will not find it. An error message will result, saying that *no file* was found. The command should be:

```
A> DEL C:\WP\PAM\LTR.OLD
```

If instead of being on drive A, Pam is currently working on drive C at the root level, then C: could be left off. If no drive is specified, DOS will look for the file on drive C, which is fine. It would be perfectly correct to include the C:, but it would be redundant.

The path would still have to be included, otherwise DOS would look for the file in the current directory, the root directory. If Pam changes directories and is working in her subdirectory, then all that would be necessary is:

```
DEL LTR.OLD
```

That's because the file will be found in the current subdirectory (\WP\PAM), on the current disk drive (C:).

Notice that the name for Pam's subdirectory was written:

```
\WP\PAM
```

That is the full name for this subdirectory. It turns out that technically, subdirectories are actually just files on the disk. Files that are supposedly "contained" in a subdirectory really just have their names tied somehow to the subdirectory name. Just as a filename has its path, so does a subdirectory name. Because they are files, there is no inherent limit to how many entries they may have (except the limitation imposed by disk space). The tree structure is constrained by disk space and the limitation that a full filename, including the path, cannot have more than 63 characters.

6.4 Creating a Subdirectory: MKDIR or MD

Only the root directory is established when a disk is formatted. All subdirectories must be created using the MKDIR command. This command can be abbreviated MD. It is an internal command, so it can be used anytime the DOS prompt is on the screen. The general form of the command is:

```
MD drive-path
```

As usual, the drive specification can be left off if the subdirectory is to be on the current drive, which is the case most of the time. To create a subdirectory named DB directly under the root directory, the command is:

```
MD \DB
```

This is the full path specification. It tells DOS to create a subdirectory that has the root directory (whose name is \) as its "parent." To then create a subdirectory named SANDY under DB, the command is:

```
MD \DB\SANDY
```

This says, "Set up a subdirectory called SANDY branching from DB, which itself branches from \."

Subdirectory names have the same constraints that filenames have. The maximum number of characters is eight in the first part (the core) and three in an extension. Spaces and many other characters are not allowed. Any corename that would be valid in a filename is valid as a subdirectory name.

It can make a difference which subdirectory is the current one. If a subdirectory is to be created one level down from the current one, then the full path specification is not mandatory. For example, if we move into the DB subdirectory, making it current, then the following command would be valid:

```
MD SANDY
```

If there is no backslash, this tells DOS to start at the current directory and create a subdirectory one level down. This command creates a subdirectory whose full name is \DB\SANDY.

While located in the DB subdirectory, Sandy can create new branches in the tree structure that are independent of, and at the same level as DB, simply by using the full path specification. Thus, no matter which subdirectory happens to be the current one, the following command is valid:

```
MD \SS
```

This creates a subdirectory named SS with the root directory as its parent. In order to create a subdirectory that has another subdirectory as its parent, the parent must already exist. For example, if the subdirectory \SS already exists, then the following command would be valid, no matter what subdirectory is the current one:

```
MD \SS\LEE
```

But if \SS did not exist, then this command would result in the error message:

```
Unable to create directory
```

To summarize, the command MD followed by a path specification is used to create a subdirectory. Only one subdirectory can be created at a time. All subdirectories in the path except the last (which is the one being created) must already exist. If a subdirectory is being created one level down from the current subdirectory, all that is required after the MD is the new name; the full path is not required.

6.5 Moving into a Subdirectory: CHDIR or CD

Once various subdirectories have been created, we need to be able to move into them, in the same way that we shift from one disk drive to another. To switch from drive A to B, we simply type in B: and hit the ENTER key. To move into a subdirectory, the command CHDIR is used, which may be abbreviated CD. CD is an internal command. It can be used no matter what disks are in the drives. The general form of the command is:

```
CD drive-path
```

The disk drive specification can be dropped when the change does not involve going from one disk drive to another. For example, if we are located at the root level of the drive that has all the subdirectories shown in Figure 6-7, and we want to move into Sandy's subdirectory for word processing files, then the command is:

```
CD \WP\SANDY
```

We end up at the last subdirectory in the path. In this case, the entire path must be specified so that DOS knows how to find the subdirectory we want. To shift over to Lee's subdirectory, the command is:

```
CD \SS\LEE
```

It is possible to use the CD command without specifying the full path. If the move is down one level, then only the name of the subdirectory need be entered. For example, if we are located in the subdirectory called DB, then to move into Sandy's subdirectory all we have to enter is:

```
CD SANDY
```

Actually, there are other times when we don't have to specify the full path. If a backslash is not the first character in the path, then DOS starts at the current directory, not at the root level. For example, if we have a subdirectory four levels down whose path is \DB\PAM\FILES\84 and we are located in the subdirectory named PAM, we can move into the subdirectory named 84 with this command:

```
CD FILES\84
```

By not having a leading backslash, we tell DOS that the path we want to move down does not start at the root level, but rather it starts at the current subdirectory.

There are numerous occasions when it is necessary to return to the root level. The root level's name is \. That is also its full path. No matter what the current directory is, to move to the root level, the command is always:

```
CD \
```

Things get a little trickier if we establish some subdirectory as the current directory, using the CD command, of course, and then shift to a different disk drive. For example, we move into the subdirectory \DB\PAM\FILES\84 on drive C, do some work, and then shift to drive A. DOS will place us at the root level of drive A. But, it remembers that the current directory on C is \DB\PAM\FILES\84, not the root directory.

If at this point we enter a command to delete a file from the disk in drive C, DOS will look for the file in the \DB\PAM\FILES\84 subdirectory, because that is the current directory on C. If we shift back over to drive C, we are placed in this subdirectory, not at the root level.

One problem all users have with all this moving around to different subdirectories is keeping track of which directory is the current one. Although DOS seems to have no trouble with this, it is a problem for us because there is no indication on the screen about the current directory. (We always know which disk drive is current, because we see the prompts, A>, or B>, etc.) A very easy way to find out the name of the current directory is to enter the CD command with nothing following it. When this is done, the current path is displayed on the

screen. If we are located in the subdirectory \DB\PAM\FILES\84 on drive C and type in CD and hit ENTER, the screen shows:

```
C>CD
C:\DB\PAM\FILES\84
```

Also, when the DIR command is used, the second line of the listing always gives the complete path for the current directory.

Finally, the PROMPT command, covered in Chapter 10, offers a way to have the DOS display the current directory as part of a prompt line.

6.6 Directory Listings and Subdirectories

If we ask for a directory listing while in the \DB\PAM\FILES\84 subdirectory, the top of the screen will show:

```
C>dir

 Volume in drive C has no label
 Directory of C:\DB\PAM\FILES\84

.            <DIR>      11-23-84   6:09p
..           <DIR>      11-23-84   6:09p
ROSTER  DBF      12376  11-30-84   11:33a

                - etc. -
```

The listing indicates what the current directory is and then shows the files. It shows only the files that are in this current subdirectory. The . and .. are special entries that are explained after the next listing. As was stated earlier, subdirectories can be thought of as lots of mini disk drives. They are separate entities. Not only do we have to concern ourselves with which disk drive is current, but also with which directory is current.

If we move back up to the DB subdirectory (by entering CD \DB) and then enter a DIR command, the directory listing will include only files and subdirectories contained in \DB. Assuming that a tree structure like that of Figure 6-7 exists, the listing would show something like this:

```
C> dir

 Volume in drive C has no label
 Directory of C:\DB

.            <DIR>      11-23-84   6:09p
..           <DIR>      11-23-84   6:09p
```

```
SANDY          <DIR>      11-23-84   6:17p
PAM            <DIR>      11-23-84   6:18p
DBASE     COM      32256   11-11-83   12:02a
DBASEOVR  COM      37376    1-01-80   3:31a

                - etc. -
```

If a name is followed by a <DIR> label, that tells us that it is a subdirectory. The . and .. are special entries that are automatically put into a subdirectory when it is created. The single period tells DOS that this is a subdirectory. The double period stands in for the name of the parent directory. It can sometimes be used in commands to avoid having to enter the entire path, starting at the root level.

For example, if we are in the \DB\PAM\FILES\84 subdirectory, and want to move back up one level to the FILES subdirectory, there are two ways to do it. One is to specify the entire path, while the other is to use only the double period, which refers to the parent subdirectory.

```
cd \db\pam\files
cd ..
```

Here's another example. We are still located in the 84 subdirectory, and we want to create another subdirectory under FILES, one called 85. Figure 6-8 shows what the tree structure should look like when we have finished. There are three ways we could go about it. One would be to move up to FILES (using the command shown above), and then enter MD 85. But if we do not want to leave 84, the other two possibilities are:

```
md \db\pam\files\85
md ..\85
```

The path in the second command does not start with a backslash, so it begins in the current directory. The double period refers to the parent directory, so DOS moves back up one level. The \85 gives the new subdirectory's name, which will branch from FILES. After entering this command, we are still located in \DB\PAM\FILES\84 and a new subdirectory named \DB\PAM\FILES\85 has been created.

When we use the DIR command, only subdirectories and files that have the current directory as their parent show on a listing. In order to get a directory listing of a subdirectory other than the current one, we must either move into it or include its path with the DIR command. For example, if we are still in \db\pam\files\84 and want a listing of what is in \db\pam\files\85, but want to remain where we are, either of these two commands can be used:

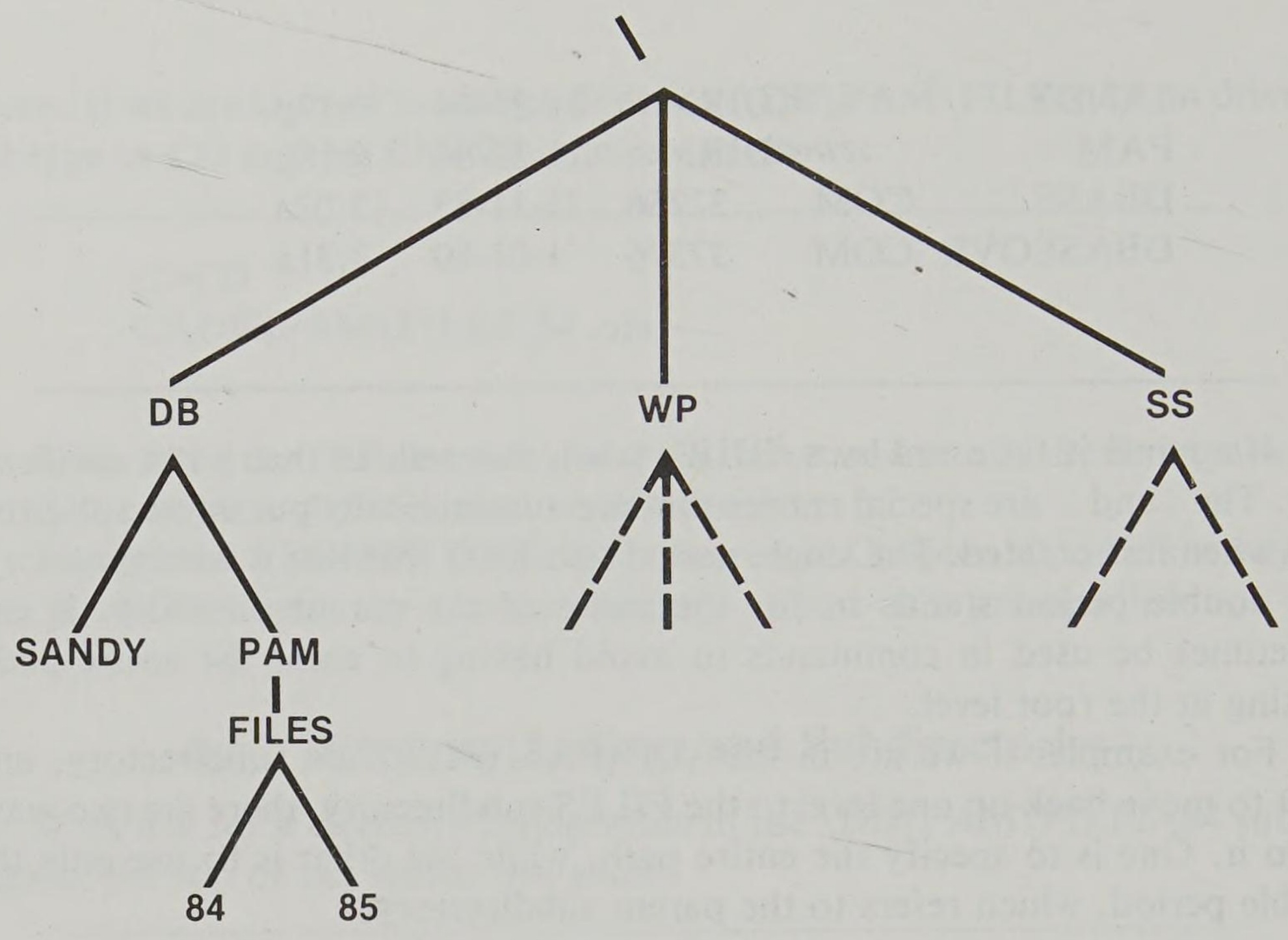


Fig. 6-8. A Tree Structure with Multiple Subdirectories

```
dir \db\pam\files\85
dir ..\85
```

If we were located on drive A, the path would have to include the drive specification:

```
dir c:\db\pam\files\85
```

To summarize here, the DIR command produces a listing of what is in the current directory on the current drive, if no parameters accompany it. If we want a listing of a directory other than the current one, we must specify its path along with the DIR command. It does not show any subdirectories or files that are farther up or down in the tree.

6.7 Copying Files to and from Subdirectories

Recall that the general form of the COPY command is:

```
COPY FILENAME DESTINATION
```

Now the full meaning of this command can be stated. The filename may

need to include its full path as well as a disk drive specification, so that the DOS will be able to find it. If the current directory is the one that contains the file to be copied, then nothing but its corename and extension are needed, because the rest is redundant. When no drive or path is specified, DOS looks in the current directory on the current drive.

The destination must include the drive specification if the file is being copied to another disk drive, and a path if it is going into a subdirectory. Let's look at some examples. We are located on drive A at the root level, where the file named ROSTER.DBF is located. We need to copy it to drive C, into a subdirectory named \DB\PAM\FILES\84. The command is:

```
COPY ROSTER.DBF C:\DB\PAM\FILES\84
```

If we need to copy all files ending in .DBF from where we are to this subdirectory, the command is:

```
COPY *.DBF C:\DB\PAM\FILES\84
```

If the reverse is the case, and we need to copy ROSTER.DBF from this subdirectory to the root directory on drive A (and we are presently located there), then the command is:

```
COPY C:\DB\PAM\FILES\84\ROSTER.DBF
```

It is not necessary to put in the destination, A:, because that is where we are currently located. The file will automatically be copied to A when no destination is specified, and it will be copied into the current directory, which in this case is the root directory.

If we are still located on drive A and need to copy the ROSTER.DBF file from C:\DB\PAM\FILES\84 to C:\DB\PAM\FILES\85, the command is:

```
COPY C:\DB\PAM\FILES\84\ROSTER.DBF
C:\DB\PAM\FILES\85
```

Nothing can be left off in this case, because both the original location and the destination are different from the current directory. The *full* filename and the *full* destination path must be specified.

Here are more examples:

Current drive is A; current directory is root:

COPY *.BAS B:\BASIC

Copies all files in the root directory of A that have an extension of .BAS to a subdirectory named BASIC on drive B.

COPY C:\WP\DARLENE\MEMO

Copies the file MEMO from Darlene's subdirectory on drive C to the root directory of A (which did not need to be included, because it is the current location).

Current drive is C; current subdirectory is \DB:

COPY A:*.COM

Copies all files ending in .COM from the current directory of A into the current directory, C:\DB.

COPY *.DBF \WP\PAM

Copies all files ending in .DBF from the current directory (C:\DB) to Pam's subdirectory (C:\WP\PAM).

It is possible, of course, to copy all the files from a directory to another disk or to a different subdirectory, using the *.* (star dot star) designation. For example, while located in C:\DB\PAM\FILES\84, if we want to copy all of its files to C:\DB\PAM\FILES\85, the command is:

```
COPY *.* \DB\PAM\FILES\85
```

In this case, using *.* is the easiest way to copy all files. Now, if we have to copy all the files from \DB\PAM\FILES\85 into our current subdirectory (also on drive C), we could use this command:

```
COPY \DB\PAM\FILES\85\*.*
```

However, to copy the entire contents of a subdirectory it is sufficient to use the subdirectory name. The following command would accomplish the same thing:

```
COPY \DB\PAM\FILES\85
```

What this says is, "Copy subdirectory 85 to the current directory." (Since no destination is specified, DOS will copy the files into the current directory.)

Care must be taken when using this method (referring to the subdirectory

name without including *.*), because if the subdirectory name is typed incorrectly, and there is no subdirectory that matches what is typed, then DOS will think that the incorrect name is to be interpreted as a new filename. Here's a sample command:

```
COPY ROSTER.DBF \DB\PAM\FILS
```

It's supposed to read \DB\PAM\FILES. Since there is no subdirectory \FILS, then FILS is taken to be a filename. When a filename is specified as part of the destination, that means that the file is to receive that new name at the destination. So, this command will copy ROSTER.DBF into the subdirectory \DB\PAM and rename it FILS at the same time. If a wildcard is used to designate more than one file to be copied, the copying takes place, but each file that is copied gets recorded under the same new (erroneous) name, overriding the previously copied file. When copying is completed, there is only one new file in the destination directory.

6.8 Removing a Subdirectory: RMDIR or RD

The RMDIR command, which can be abbreviated RD, is used to remove a subdirectory. It is an internal command, which means it is always available. The general form of the command is:

```
RD drive-path
```

The last subdirectory name in the path is the one that is removed. Only one subdirectory can be removed at a time. One condition must be met before a subdirectory can be deleted. The subdirectory must be empty. No files may be in it when the attempt to remove it takes place, with the exception of the files that are named . and .. (which are always there). They do not have any effect on the process. This is a precaution, so that we cannot remove the wrong subdirectory by mistake. If the wrong name is entered, or if the subdirectory still has files in it, this message appears:

```
Invalid path, not directory,
or directory not empty
```

When files are in the subdirectory that is to be removed, they must first be deleted with the DEL/ERASE command. All files in a subdirectory can be deleted using the *.* wildcards, or, just as all files can be copied from a subdirectory by simply referring to the subdirectory's name, they can all be deleted

by using its name. The *.* wildcards would be used when the subdirectory to be purged is the current one. The command then is simply:

```
DEL *.*
```

But if the current drive is A, and the subdirectory to be cleaned out is C:\DB\PAM\FILES\84, the command to use is:

```
DEL C:\DB\PAM\FILES\84
```

This has the same effect as using the command:

```
DEL C:\DB\PAM\FILES\84\*.*
```

Each time all files are about to be erased from a directory, the confirmation request "Are you sure (Y/N)?" appears. After seeing this message many times, people generally start to pay little attention to it. Never rush through this procedure! *Think carefully anytime all files are going to be erased.* Too many disasters have occurred because people were not careful enough about this procedure for deleting all files. A good precautionary method is to use the DIR command first, in place of DEL. DIR followed by the filename with wildcards (or just a subdirectory name) shows a listing of the files that will be deleted if DIR is replaced with DEL.

Once all the files have been deleted from a subdirectory, that subdirectory can be removed. If there are several subdirectories to be removed and every one has files in it, it will take two steps to get rid of each one: delete all the files, then remove the empty subdirectory. There is no shortcut.

6.9 Summary

Subdirectories are used to create separate working areas on a disk. They are used primarily on hard disk systems. They are organized into hierarchical tree structures, which means that each subdirectory is "attached" to a parent directory. The root level is the starting point. There is no inherent limitation on the number of subdirectories except disk space. The only practical limitation is that a subdirectory's full path specification cannot exceed 63 characters.

The path to a subdirectory is a list of all the subdirectory names between it and the root level. Names are separated by backslashes (\). For a tree structure such as that shown in Figure 6-8, the path to Pam's subdirectory that contains her 1984 files is:

```
\DB\PAM\FILES\84
```

The name of a file may need to include its full path – when the filename is used with the ERASE command, for example. A path that is included in a filename comes after the drive specification and before the corename. The generic form of a complete filename, along with an example, is:

```
DRIVE-PATH-CORE.EXTENSION
c:\db\pam\files\84\roster.dbf
```

The MKDIR (abbreviated MD) command is used to create subdirectories. It is followed by the path of the new subdirectory. CHDIR (abbreviated CD) is used to make a subdirectory the current one. This is similar to switching to a different disk. The CD command is also followed by the path. RMDIR (abbreviated RD) is used to remove a subdirectory. The subdirectory must be empty before it can be removed.

The DIR command gives a directory listing of only the current subdirectory. It can be followed by a path that tells it which subdirectory to list.

Copying files to and from subdirectories is just like going from disk to disk. The FILENAME portion of the command must include the path to locate the file or files unless the subdirectory in which they are located is the current one. The DESTINATION can be omitted if it is the current directory; otherwise, it will need to have its path specified.

CHAPTER 7

Where Is It? Running Programs

7.1 Internal Versus External DOS Commands

There are many tasks that DOS must perform for the user – formatting, copying, deleting, and renaming, for examples. DOS must, therefore, include the programming routines to take care of each of these tasks, and this programming must be available to the user.

When the DOS, which is programming, is brought into the computer's memory at startup, it goes into RAM (Random Access Memory), also known as the working memory. An applications program such as word processing also goes to working memory, and some of the data are stored there, at least temporarily.

Because the computer may have as little as 64K (64,000 bytes) of RAM, it is not feasible to put *all* of the programming needed to control *all* of the DOS functions into the memory at once and leave it there, ready to do our bidding. This would not leave enough room for doing the real work of the computer. That is why only certain DOS functions are included in the programming that is placed in the computer's memory at startup. The commands that control these particular functions are called **internal commands,** because the programming necessary to carry them out always resides in the computer's memory; it is stored internally.

What about functions for which the programming is not in the memory, readily available for our use? If the programming is not in the memory, then the only other place it can be is on disk. Commands that must go and get programming from the disk to control their functions are called **external commands.** Appendix C contains a list of the internal DOS commands and a list of the external ones.

Obviously there must be some way for the DOS to differentiate between the two types of commands. The component of the DOS that takes care of this is called the command processor. When a command is entered, the command processor checks it against its list of internal commands. If it matches one of these commands, the appropriate programming routines already in the memory are accessed immediately.

This "list" of internal commands is what is contained in the COMMAND.COM file. This file also contains the programming that makes up the command processor. Occasionally, an applications program will need the memory space where this part of the DOS resides and will overwrite it. When such a program is finished, DOS will automatically try to reload the command processor. If COMMAND.COM is not available for reloading (maybe we switched disks), then the following message appears:

Insert COMMAND.COM disk in drive A:
and strike any key when ready

DOS is asking us to put in any disk that has COMMAND.COM on it, so that the programming to take care of internal commands can be reloaded.

If the command that was entered does not match any of the internal commands in the list, then the DOS assumes that it is the name of a program and goes looking for that program on the disk. If no disk drive specification or path is included in the name, the search takes place on the current drive and directory. If there is no program on the disk with that name, DOS displays this error message:

Bad command or file name

If the program is there, then all the programming instructions recorded on the disk under that name are brought into the memory, and execution of those instructions begins. Normally, this is referred to as "running a program."

The key point here is that unless a command is an internal one, one of the disks in the computer, one that is "on line" as we say, must have the programming recorded on it under a name matching what is being called "the command." As far as DOS is concerned, and as far as we are concerned, for that matter, there is no difference between running a DOS program (that is, any external DOS command) and running an applications program. The DOS goes out to the disk, gets the programming, puts it in the memory, and begins execution of that programming.

7.2 On the Current Drive: .COM and .EXE Programs

It is actually quite inadequate to say that if DOS does not find a command in its list of internal commands, it checks for that name on the disk. Think

about the FORMAT command. It is an external command; that is, a program stored on the disk. So to run the FORMAT program, we type in FORMAT and then hit the ENTER key. This causes the FORMAT program to run. But when we look at the name of this program on the disk (in a directory listing), we see that it is actually FORMAT.COM.

On the same disk we have a word processing document file that contains nothing but the characters that were typed in, and its name is START. We type in START and then hit the ENTER key. What happens? Everyone's favorite message appears:

Bad command or file name

Not all files on a disk are executable programs. The ones that are programs have either .COM or .EXE as their extension. COM is short for command. Files with .COM are referred to as command files. EXE is short for executable, but files with .EXE are just called E-X-E files. The reason that there are two types (.COM and .EXE) is fairly technical and not worth exploring. To the user, these two types of files are identical.

What is important to know is that if a filename ends with either extension, .COM or .EXE, then it is a program that can be executed, and that in order to execute it, or run it (the more common terminology), we just type in the first part of the name and leave off the .COM or .EXE extension. After DOS checks its list of internal commands and does not find the name, it will look on the disk for a file with that name and a .COM or .EXE extension.

What disk will it look on? Of course, the current disk drive. Just as with copying files, if no disk drive is specified, the current drive is assumed.

7.3 Finding the Program on Another Drive

In order to run a program that is located on a drive other than the current one, we proceed as we do when specifying that a file to be copied is on a different disk drive. The drive specification is included in the name. Here's a sample situation: The disk in drive A has the CHKDSK program on it, the current drive is B, and the disk in drive B does not have the CHKDSK program on it. If we type in just CHKDSK, DOS will look for the CHKDSK program on drive B, the current drive. It will not find the program and the "Bad command or file name" message will appear. The command must be:

A:CHKDSK

Now here comes the part that often confuses people. When the command is typed in like this, which disk will be checked by the CHKDSK program? It turns out that the status of drive B will be reported. Why? Because no disk

drive specification *followed* the CHKDSK command, and that means that the current drive will be checked. Drive B is the current drive.

In order to have the status of drive A checked while B is the current drive and the CHKDSK program resides on drive A, the command must be:

```
A:CHKDSK A:
```

This command answers two separate questions: First, where is the CHKDSK program? Second, which disk will it act on?

If the program to be run is not on the current drive, then its drive specification must be included in its name. A drive specification may always be included in a name, but if it is the same as the current drive, it is redundant, so why bother to go to the extra work of typing it in?

This is a good time to distinguish between the terms "filename" and "program name." Filename is the more general term. A file may be a program, or it may be data, a word processing document, for example. We need to recognize one difference between filenames and program names. "Filename" refers to the entire name, including any drive specification, path, and extension. "Program name" does not normally include the .COM or .EXE extension. For example, we talk about the FORMAT and CHKDSK programs. That's because to run them, we type in just these portions of the names (and if necessary the drive specifications), and not the extensions. The filenames for these two are FORMAT.COM and CHKDSK.COM.

There are times when the distinction between filename and program name becomes important. For example, to COPY the CHKDSK program from one disk to another, the following command would not work:

```
copy chkdsk b:
```

There is probably no file on the disk with the filename CHKDSK. The correct filename for the CHKDSK program is CHKDSK.COM. The same would apply to deleting the file. Running a program is not the same as referring to its filename in order to copy it or delete it. To run it, we type in its name, minus extension. DOS then looks for a such a name with a .COM or .EXE extension. If it finds such a file on the disk, the programming instructions recorded in this file are brought into the computer's memory and executed.

7.4 Finding the Program in a Subdirectory: PATH

In the previous section we saw how to run a program that was on another disk drive. That discussion ignored the issue of subdirectories. There was an implicit assumption there that only the root directories were involved. What if a program is down two levels in the tree structure, in a subdirectory?

If we are located at the root level, and the CHKDSK program we want to run is in the subdirectory \DOS, we cannot just type in its name, because of course the DOS will look for it at the root level. Since we can include a drive specification as part of a name and in this way tell the DOS where to find it, it would seem logical that we could include the path to the file in a program name to tell DOS to run it from there. Yet, this is not the case. The path may *not* be included in a program name when we are trying to run the program. So the following command would not work:

```
\dos\chkdsk
```

DOS does not even give an error message to this; it just ignores it and puts the DOS prompt back on the screen, waiting for a valid command.

This does not mean that we cannot copy or delete the file by referring to its full filename, including the path. It means only that DOS does not have the capability of reacting to a path in a program name when we are trying to run the program.

One way to get the program to run is to move into the subdirectory that contains it. Then that subdirectory is the current one, and when the command to run the program is entered, DOS can find it there in the current directory.

But there are times when we want to stay in one subdirectory and yet still be able to run a program that is located in another. Here's an example of such an instance. The CHKDSK program is in a subdirectory named \DOS. We are located in a different subdirectory that contains our word processing program and documents. We want to run the CHKDSK program once, then immediately go back to word processing.

There is an alternative to using the CD command to move to the \DOS subdirectory, running CHKDSK, then moving back to the word processing subdirectory again. The PATH command can be used to tell the DOS where to look for a program that we try to run, if it is not found in the current directory.

The name of this command can be misleading. It does not have anything to do with *creating* paths. Instead, it sets up one or more paths that are to be *searched* if the program we are trying to run is not found in the current directory. For the previous example, we could enter these two commands:

```
PATH \DOS
CHKDSK
```

The DOS will first look in the current directory for the CHKDSK program. Not finding it there, it will then check in the \DOS subdirectory. When it finds it there, it will run the program. The \DOS subdirectory is searched because the PATH command has been used to establish it as the second place to look.

The general form of the PATH command is:

```
PATH path;path;path;etc.
```

It is an internal command. More than one path can be entered. When multiple paths are specified, the path names are separated by semicolons (;). This means that we can order a list of subdirectories to be searched. The first path entered will be searched first, then the second one entered, and so on until the program is found. If it is found somewhere along the line, the search stops and the program is executed. If the search goes all the way through to the last path and the program has not been found, the "Bad command or file name" error message is displayed.

Here's an example of establishing multiple paths:

```
path \dos;\wp\ws;\wp\multi;\wp\vw
```

If we were trying to tell a human being what to do here, the statement would go like this: "If you don't find your program in the current directory, then go check the \DOS subdirectory. If it's not there, then check the \WP\WS subdirectory next. If you still haven't found it after that, go on to the \WP\MULTI subdirectory and check there. Finally, if you still have had no luck, as a last resort, try the \WP\VW subdirectory. If it isn't there, then you are plain out of luck, and you get no more tries."

The search paths that are established with this command remain in effect until the computer is turned off, or until they are nullified. To nullify search paths, enter the PATH command and include a semicolon only.

```
PATH;
```

After working feverishly all day long on the computer, it is very easy to forget what paths were established hours ago at startup. To have DOS show what search paths are in effect, we enter just the PATH command with nothing else. Doing this after having earlier entered the example with multiple paths would produce on the screen:

```
C> PATH
PATH = \DOS;\WP\WS;\WP\MULTI;\WP\VW
```

A few problems can develop. What happens if the wrong path is entered? Well, that depends on what "wrong" means here. If a valid path was entered, but it was the wrong one, then that wrong subdirectory will be searched. If the path entered does not exist, then it is just ignored, and DOS goes on to the next path that is to be searched (if there is another).

For anyone who has to set up search paths with this command every time he or she starts up, batch files will be very helpful. Batch files provide a way to have the computer enter commands for us. They are introduced in Chapter 8.

7.5 Programs with Overlays

It would surely be nice if the PATH command was an end-all solution to the problem of running programs that are not in the current directory, but unfortunately that is far from true. If all programs were small enough to fit in the memory of the computer and still leave room for data manipulation, there would be no further problems. But as the tasks get more sophisticated, the programs must also get more sophisticated. The more complex a program is, the more programming instructions it requires.

When the number of instructions in a program gets so high that they cannot all be put into the computer's memory at once, then the program must be broken up into modules. Some of the programming is put into the memory, and the program begins to do its job. At some point, a certain task may require programming that is still on the disk, in module two. This module of instructions will be brought in and used. Since some space has to be cleared for it, the programming that was wiped out to make room may have to be brought back in again from the disk when the program has finished with module two.

Most major programs have to do this. A buyer of WordStar, a well-known word processing program, finds that to run it, three files must be present on the disk. The three are WS.COM, WSMSGS.OVR and WSOVLY1.OVR. The WS.COM file is the main part of the program. The other two are what are called **overlay** files, which is what the .OVR extension stands for. They contain more programming, and whenever it becomes necessary, the programming in these overlays will be brought into the computer's memory.

The crucial point here is that the WordStar program must periodically tell the DOS to go to the disk and get something from an overlay file (when a program accesses a disk, it normally goes through the DOS to do so). And so, for the computer, the issue arises: Where is the overlay file located?

If all the files involved are in the same directory, and that directory is the current one, everything runs fine. But if the main program file and its overlays are in one directory, and the current directory is a different one, there is a problem. If the program is run from a different directory, this means that the PATH command must have been used. Setting up search paths makes it possible for DOS to find the .COM or .EXE file initially and start up the program. The problem is that when the program asks DOS to get more programming from an overlay file, DOS can only look in the current directory. The PATH command does not establish which subdirectories should be searched when a program calls for more programming from overlay files.

The DOS just does not have the capability of keeping track of other places to look for overlays. The PATH command only affects the search for a program when it is being started. Once it is running, overlays are assumed to be in the current directory.

If they are not, then the program will not function properly. With WordStar, for example, an error message occurs immediately when it is run from another directory. The message says:

```
File WSMSGS.OVR not found. Menus &
messages will display as @@@@ only.
```

The program cannot function without the overlay files being available to it. There are only two solutions to this problem. The first, and the best, is to work out of a directory that has all the necessary program files in it. This directory will then have the programming and data files in it.

The second is to have the main program file (for WordStar this is the WS.COM file) in some other subdirectory but to have the overlay files (for WordStar, WSMSGS.OVR and WSOVLY1.OVR) in the current directory.

This will work, but what does it buy us? There is no point in putting overlay files in just one subdirectory. If we do this, all the files, including the main one, should then be in this one subdirectory only, and there will be no need to use the PATH command to tell DOS where to find the WS.COM file.

This kind of setup is useful if there are multiple subdirectories. For example, if the system has a hard disk, and if several people will be using WordStar to do their word processing, it makes sense to have a separate subdirectory for each person, so they don't have to worry about other people's documents cluttering up their subdirectories.

Let's say that there are seven people, and each one has a subdirectory. Rather than having to copy all three of the WordStar files into all seven subdirectories, the WS.COM file can be put in the parent directory only. The other two, the .OVR files, must be copied into each subdirectory.

By the way, this can be done more quickly than it might seem. Let's say that the WordStar disk is in drive A, and we are located on drive A as well. The subdirectories for each individual are on drive C, a hard disk. Each person has a subdirectory under \WS (for example, Sandy's is \WS\SANDY). The first command might be:

```
copy *.ovr c:\ws\sandy
```

This says "Copy all files from the current drive that end with the extension .OVR to drive C and put them in the \WS\SANDY subdirectory." Next, rather than retyping the whole command, replace \WS\SANDY with the name of the next person's subdirectory. The F3 key can be used to tell DOS to type out the same command as last time. Before hitting ENTER, the Backspace key can be used to back up and erase SANDY. After typing in the name of the next subdirectory, all that has to be done is hit the ENTER key.

Figure 7-1 shows the tree structure of this setup, with subdirectories in

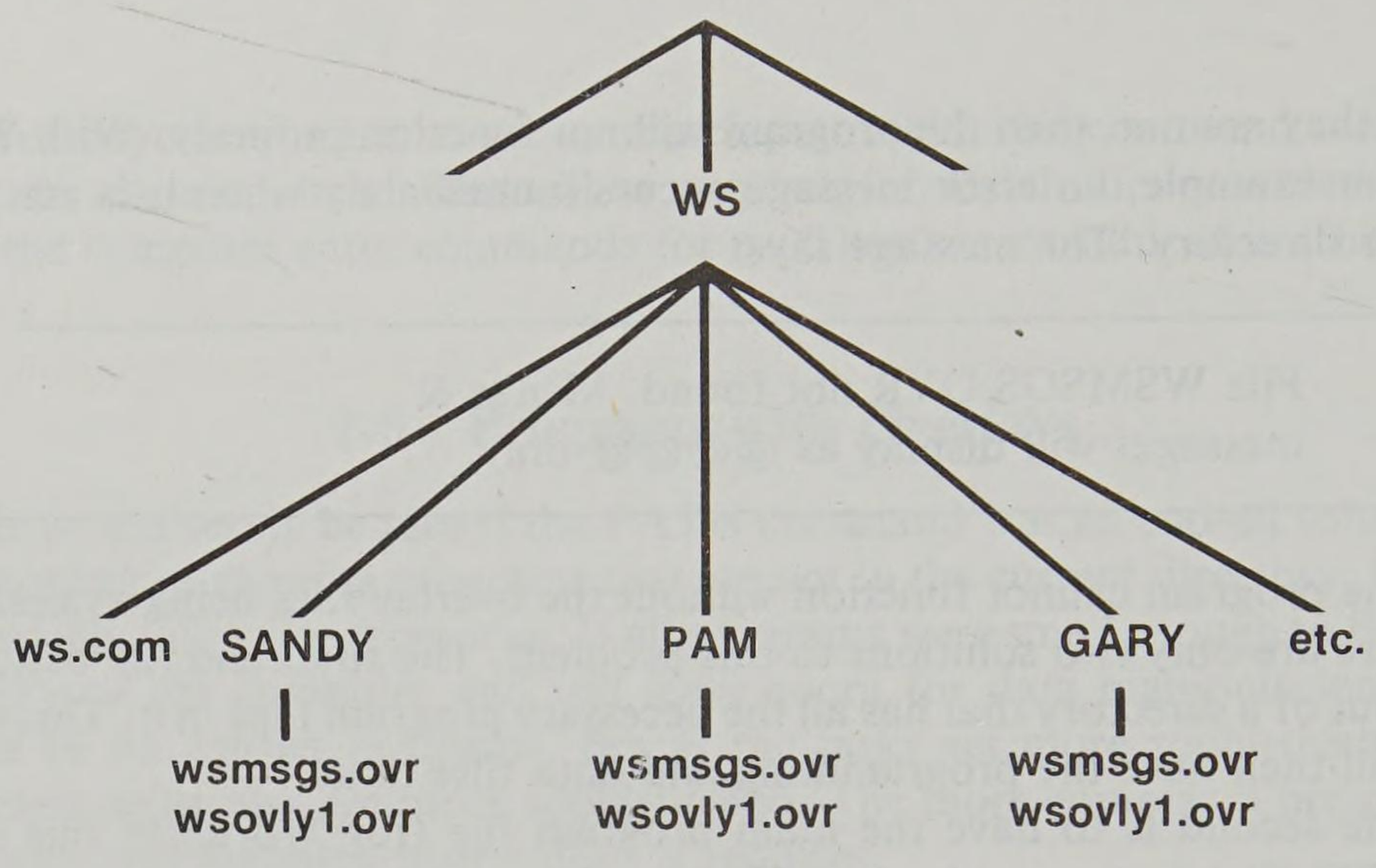


Fig. 7-1. Subdirectories with Overlay Files in Them

capital letters to differentiate them from the files, which are shown in lowercase. Of course, in DOS they are all converted to uppercase. The original question was, "What does this buy us?" In this case, it saves having WS.COM in every subdirectory. It happens that the WS.COM file takes up 21K of disk space. It is in one subdirectory, not seven, so 6 times 21, or 126K of space is saved.

On a 10 MB (10 megabytes, or 10,000K) hard disk, 126K may not be terribly significant. Having to replicate the .OVR files seven times uses up a lot more space. The WSOVLY1.OVR takes up 41K of space, and the WSMSGS.OVR takes 29K. That's 70 times 7, or 490K of space.

Anyway, these are the kinds of tradeoffs that have to be considered on a hard disk system. The method we have just explored creates totally separate work areas for each person. The alternative is to have one subdirectory with the programming and everyone's document files. The issue really does not arise for computers with only one or two floppy disk drives. On those, all users have their own disks.

7.6 Rerouting Predefined Programs: ASSIGN

It's not bad enough that we have to worry about keeping track of programs with overlay files, but some programs won't even let DOS look for those overlays in the current directory. They come preprogrammed to look for overlays on drive A. When hard-disk computers like the IBM XT started to become popular, people began to discover that some of these programs did not work properly the first time they tried to run them on drive C.

Also, certain programs were preprogrammed to put data files on drive B. On floppy disk systems, it is the norm to put the disk with all the programming on it in drive A, and a disk to have the data files recorded on it in drive B. Not anticipating future changes, such as hard disks that are set up as drive C, some

software manufacturers locked their programs into this mode of operation and did not include provisions for the user to specify what drives should be used.

The people at IBM were aware of this problem, and so in DOS 2.0 and 2.1 they included a command to allow us to work with these types of programs from a drive other than A or B. The command is ASSIGN, and it is external. It would normally be used in one of these three forms:

```
ASSIGN A=C
ASSIGN B=C
ASSIGN A=C B=C
```

The third one is a combination of the first two. It says to the DOS, "Anytime drive A is supposed to be used, use C, and anytime drive B is supposed to be used, use C." The application program will try to access drive A or B, and DOS will actually go to C, allowing the program to work from drive C.

This command should be used ONLY if absolutely necessary to get a program to work properly. It remains in effect as long as the computer stays on, or until canceled. It is canceled by entering the ASSIGN command with no parameters following it:

```
ASSIGN
```

The programmers who wrote the ASSIGN program also realized that having a reassignment in effect would not be desirable when the user was trying to run the DISKCOPY or DISKCOMP program, so these programs are not affected by the use of the ASSIGN command.

7.7 Seeing Where Everything Is: TREE

After a tree structure with many branches has been created, it may be difficult to remember not only the exact organization, but especially where all the different files are. The TREE command is intended to help in this kind of situation.

To use this DOS program, we simply type in TREE and hit the ENTER key. It is an external command, so it needs to be in the current directory or in one that will be searched (because the PATH command was used). TREE displays on the screen a list of all the directory paths on the current drive. If a list of the directories on a different drive is desired, that drive can be specified after the TREE command:

```
TREE B:
```

The list will scroll up the screen at a faster pace that it can be read. As is the case with the DIR command, this scrolling can be temporarily halted with a Ctrl-NumLock. Striking any key causes it to resume listing.

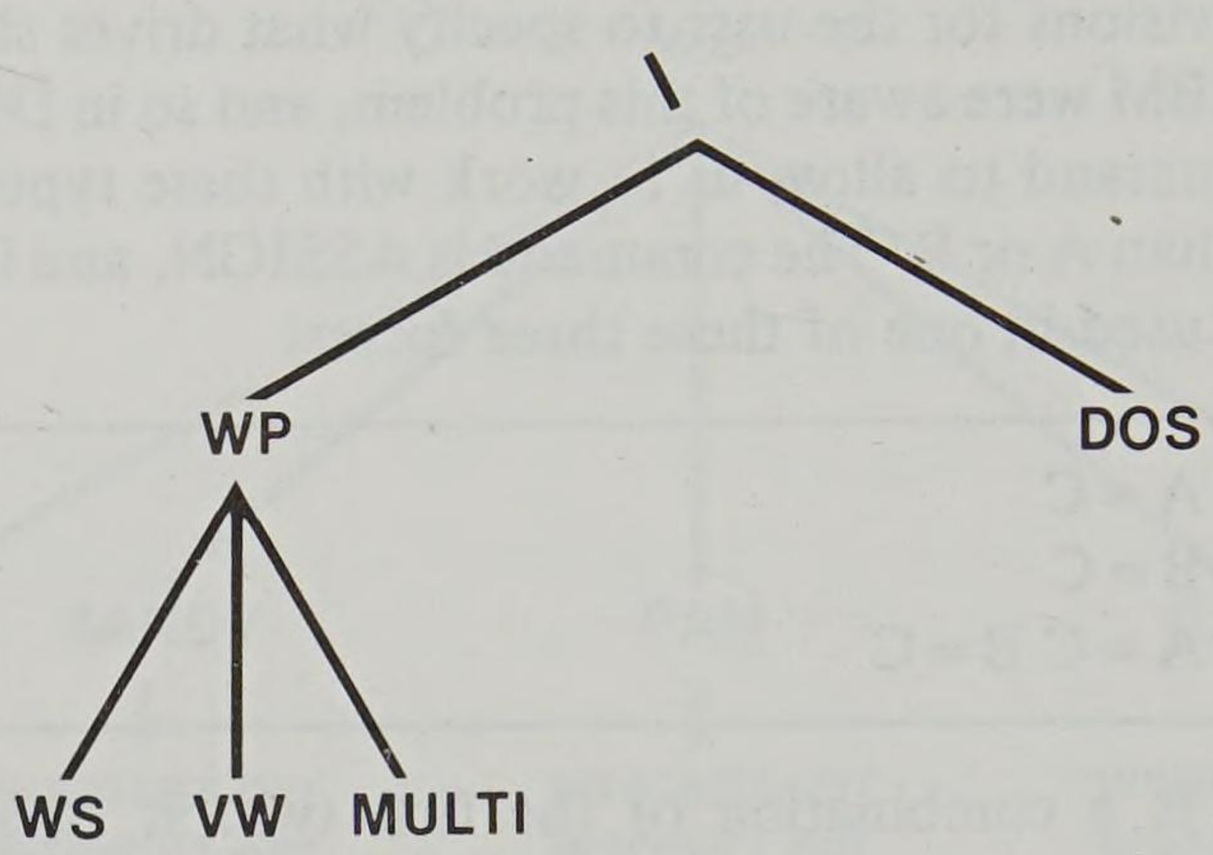


Fig. 7-2. Sample Tree Structure

For a tree structure such as that in Figure 7-2, the listing of directory paths produced by the TREE command would be:

```
DIRECTORY PATH LISTING FOR VOLUME ???????????
Path: \WP
Sub-directories: WS
                 VW
                 MULTI

Path: \WP\WS
Sub-directories: None
Path: \WP\VW
Sub-directories: None
Path: \WP\MULTI
Sub-directories: None
Path: \DOS
Sub-directories: None
```

This provides an easy way to review what the actual tree structure looks like. But it does not help in locating a particular file. It could be very tedious to have to get a directory listing of each subdirectory, using DIR followed by the path of the subdirectory to be checked.

Reminder: It would not be necessary to scan through the entire directory listing in each subdirectory. The DIR command can be followed by a filename. This tells the DOS to search in the directory for that filename and display the listing only for that filename if it is found.

But all is not lost. The TREE command has one option, /F. The F stands for FILES. Using the /F option directs TREE to include the names of all files along with each subdirectory as it is listed. The files that are at the root level will not be shown, as only subdirectories are listed by the TREE command. The root level must be checked separately with a DIR command.

This listing can be sent to the printer by using the Ctrl-P toggle. Type in the TREE command, then a Ctrl-P, then hit the ENTER key. The listing will go to the printer as well as to the screen. When the listing is done, type in another Ctrl-P so that the output of subsequent DOS commands does not continue to go to the printer.

7.8 Running Programs in BASIC

Up to this point, the only types of programs discussed have been the ones that are ready to run immediately, the .COM and .EXE types. Next, we'll look at how to use programs that are written in BASIC, the programming language that normally comes free with the purchase of a microcomputer. BASIC stands for "Beginner's All-purpose Symbolic Instruction Code." It was originally developed in the early 1970s. There are many different versions of BASIC. The one used with most computers running DOS 2.0 or 2.1 is Microsoft's version. It is usually referred to as Microsoft BASIC.

Before we get into the details of using BASIC, we need to run through what a programming language is. A microcomputer's processor chip reacts to instructions that are sequences of zeroes and ones. It is not easy for human beings to put instructions into zeroes and ones, so most of the time programming languages are used instead. The instructions in a programming language look a lot more like English. Before they can be used by the computer, these instructions must be translated into the zeroes and ones the microprocessor needs.

There are two ways this translation process can take place – by compiling or interpreting. The whole set of instructions may be run through a compiler, which is a program that translates the instructions into sequences of zeroes and ones. The original version is called the source code and the compiled version of the program (that is, the translated version) is called the object code. The compiled version will then be a .COM file and ready to use.

Translation can also take place while the program is actually running. This is done by an interpreter, a program that reads the first instruction of the program, translates it into zeroes and ones, allows the microprocessor chip to execute that instruction, and then proceeds to the next instruction.

What we refer to as BASIC is essentially an interpreter. So to learn how to program in BASIC, all we have to do is learn the commands that it can interpret and learn how to string them together effectively. It is not really as hard as

it may seem at first glance. Here is an example of what a small program in BASIC looks like:

```
10 CLS
20 PRINT "Hello, I am a sample BASIC program."
30 PRINT
40 PRINT "Please type in your full name for me now."
50 INPUT N$
60 PRINT
70 PRINT "Hmmm, nice name. Watch this!"
80 PRINT
90 FOR X = 1 TO 12
100 PRINT N$
110 NEXT X
120 PRINT
130 PRINT "Want to do it again (y/n)?"
140 INPUT MORE$
150 IF MORE$ = "y" THEN GOTO 10 ELSE END
```

The numbers are called line numbers, and they tell BASIC the order in which the commands come in. The command on the line with the lowest number comes first, then the next, and so on. BASIC actually interprets the commands one line at a time.

In line 10, the command CLS causes the screen to be cleared and the cursor to go to the upper left corner (CLS is short for CLear Screen). The PRINT command is used to display something on the screen. A PRINT statement followed by nothing is used to put a blank line on the screen. The INPUT command takes in whatever the user types in. The full name that the user enters in response to line 40 is stored in the computer's memory under the name N$. Lines 90 through 110 create what is called a loop. The FOR NEXT loop has the computer display the person's name on the screen twelve times. The IF THEN ELSE statement in line 150 checks what the person enters in response to the "Want to do it again" query. If the answer is y for yes, then the program goes back to line 10, which starts the whole process again. If the answer is anything other than y, the program ends.

Since the BASIC interpreter is needed to interpret our English-like commands into zeroes and ones, it must be in the computer's memory when we want to run a program written in BASIC. The interpreter that we call BASIC is a program on the DOS disk. To use BASIC, we run that program.

A directory listing of the DOS System disk shows two versions of BASIC, listed as BASIC.COM and BASICA.COM. The A on the second one stands for Advanced. Advanced BASIC has more capabilities than the first version, which is usually called Disk BASIC. Actually, on IBM PCs and XTs there are three versions. The third is Cassette BASIC. Cassette BASIC is stored permanently in the ROM, the Read Only Memory. When an IBM PC is turned on with no disk in drive A, Cassette BASIC is started up automatically.

Normally, either Disk BASIC or Advanced BASIC is used, though, and these are started up just as any other .COM type program is – by typing in the name, minus the .COM. Typing either BASIC or BASICA and hitting the ENTER key causes the DOS to put the BASIC interpreter into the working memory, and then the screen displays information about the particular version that is running. The "Ok" on the screen is a message from BASIC saying that it is ready to interpret a command or run a program. Since it prompts us that BASIC is ready, just as A> or B> does for DOS, the Ok is called the BASIC prompt.

Some commands can be entered directly at this point; others can be used only in programs. Programs can be entered into the computer's memory (simply by typing them in) and saved to disk. The program listed above can be typed in exactly as shown. To run it, just type in the BASIC command RUN and hit ENTER; the program executes. After it is has finished running, the Ok prompt returns. To see the program lines again, use the LIST command (remember that Ctrl-NumLock temporarily stops a listing that is scrolling up the screen, and any key starts it again). To record the program on disk, the SAVE command is used. Because of this, recording something on disk is usually referred to as "saving it on disk." The SAVE command is followed by a quotation (or quote) mark and then the name of the file to be saved:

```
save "nameplay                    (and hit <ENTER>)
```

Any valid corename can be substituted for "nameplay." There is only one opening quote mark. A closing quote is not needed. The full name of this file when it is recorded on the disk will actually be NAMEPLAY.BAS. The .BAS extension is put on automatically by the SAVE command.

Note: If a word processing program or commercial editor is being used to type in the program, the .BAS will need to be included in the filename.

To erase this program from the memory of the computer (but not from the disk), the command used is NEW. To delete a range of line numbers from a program in memory at the time, the DELETE command is used (it may not be abbreviated). ERASE is used to eliminate arrays.

There are entire books on BASIC. This discussion is intended to show only that DOS commands and BASIC commands are not the same. While the computer is in BASIC "mode," DOS commands are not recognized. There are some BASIC commands that are the equivalent of DOS commands. To see a list of the files on disk, the BASIC command is FILES. To delete a file from the disk, the command is KILL" followed by the filename with no space after the quote. (It must be the full filename, including the .BAS extension.)

When it is time to leave BASIC and get back to DOS, the command to use is:

```
SYSTEM
```

This returns control of the computer to DOS (it can only be entered when the

Ok prompt has just come on the screen). Whatever drive was current when BASIC was started is still the current drive.

There are two ways to start programs written in BASIC. One is get into BASIC (by entering either BASIC or BASICA) and then use the LOAD and RUN commands. If the NAMEPLAY program is to be used, the screen would look like this:

```
Ok
load "nameplay
```

No closing quote is needed. If the opening quote is left off, the "TYPE MISMATCH" error message appears. Once the program has been loaded into the memory, the Ok prompt reappears. To start the program, the command is:

```
run
```

When the BASIC program is finished, the Ok prompt reappears.

The second way is to type in BASIC or BASICA, then a space, then the name of the program in BASIC (without the .BAS extension). To start our NAMEPLAY program this way, the command would be:

```
basic nameplay
```

This tells the DOS to run BASIC. Once BASIC is in the computer and running, it takes the parameter included in the command as the name of a program with a .BAS extension that is to be loaded and run. This method is obviously faster and easier to use.

There is still more to this story of running programs in BASIC. The commands above will work if both files, the BASIC.COM or BASICA.COM and the actual program (with .BAS extension), are on the current disk and current directory. The question is, will things still work if BASIC.COM or BASICA.COM is in one place and the program in another?

On IBM PCs and XTs, they will. When already in BASIC, a command like this one can be used:

```
LOAD"C:\FILES\NAMEPLAY
```

The PATH command can be used to make sure that BASIC.COM can be found in some other subdirectory. Even commands like this can work:

```
B> a:basic c:\files\nameplay
```

Note that this applies to IBM PCs and XTs. What will happen with a computer running MS-DOS will vary. On some, all commands will work; on others they will not. There will probably be no problem with having BASIC on drive A and the program on B for example, but problems may arise when subdirectories are involved. Experimentation will tell.

7.9 Summary

There are two types of DOS commands, internal and external. Internal commands are already in the computer's memory, and so are always recognized, no matter what disks are in use. External commands are programs that must be found on one of the disks in use, so that the programming to carry out their function can be brought into the computer's memory.

Files that have .COM or .EXE as their extension are executable programs. An executable program can be started up, or "run," by typing its name, minus the extension. If the program is not on the current disk, the name must include the disk drive specification. Example:

```
B> a:chkdsk
```

The PATH command is used to tell DOS where to look for programs if they are not found on the current disk. These locations are specified by including their path in the PATH command. Multiple paths can be specified, separated by semicolons. Example:

```
path \dos;\wp\ws;\wp\vw
```

These "search paths" remain in effect as long as the computer stays on, or until negated (PATH;).

Many sophisticated programs are broken up into modules. Not all the modules of programming are loaded into the computer's memory at one time. The files that contain the remainder of the programming are usually called overlay files (and most often have an extension of .OVR). The PATH command does not help DOS find these when they are needed. DOS can find them only if they are in the current directory. This may mean that, to be used effectively, they must be reduplicated in several subdirectories on a hard disk.

The ASSIGN command, which is external, takes care of troublesome programs that are preprogrammed to find overlay files on drive A or data files on drive B. It tells DOS to route anything intended for a specific drive to another. Here's the command to switch everything from A to C and from B to C also:

```
ASSIGN A=C B=C
```

The TREE command, another external one, displays a list of all subdirectories on a disk. If the /F option (for Files) is included TREE also lists all the files in each subdirectory. It does not list the files that are at the root level.

Programs in BASIC can be run by typing in BASIC or BASICA followed by the name of the file that contains the programming (minus its .BAS extension). Example:

```
BASICA STARTREK
```

Programs in BASIC are interpreted line by line as they are executed. This makes them relatively slow. The prompt for BASIC is "Ok," and the command to return to DOS from BASIC is SYSTEM.

CHAPTER 8

Batch Files and Automatic Start-ups

Batch files are a feature of DOS that is often overlooked, but shouldn't be. Once they are understood, batch files can be a very handy tool for simplifying many tasks that must be repeated often.

8.1 What Batch Files Do

Almost every computer user encounters a sequence of DOS commands that must be entered almost every time the computer is used. For example, someone using WordStar with a two-floppy-disk system and a serial printer might start up like this (the MODE command is explained in Chapter 12):

```
A>mode lpt1:=com1:
A>mode 1200,n,8,1,p
A>b:
B>erase *.bak
B>a:ws
```

Batch files provide a way of having DOS enter commands like these for us. A batch file is nothing more than a sequence of valid DOS commands recorded in a text file on the disk. The filename must have a .BAT extension. When instructed to do so, DOS will go look at this file and then execute each of the commands just as if we had typed them in one by one.

Two benefits are derived from using a batch file. The first is that it avoids

having to type in all those commands, and the second is that it dramatically decreases the time it takes to complete all the commands.

Telling DOS to execute a batch file is quite simple; we just type in its name, minus the .BAT. If the batch file has been named GO.BAT, then it is executed by typing in GO and hitting the ENTER key. Sound familiar? This is the same way we run .COM and .EXE programs. So it turns out that there is a little more to the story of what DOS does when a command is typed in. In Chapter 7 it was stated that DOS first checks to see if the command is an internal one (such as DIR), and if it is not, then it looks for a file by that name with a .COM or .EXE extension. Now we see that it will also check to see if there is a file by that name with an extension of .BAT.

The next question is, what happens if there are three different files on the disk with the same corename, but ending with .COM, .EXE, and .BAT? This should never be allowed to happen, but if it does, DOS will find and execute the .COM program. It works this way: DOS checks for an internal command with that name; if none exists, DOS checks for a .COM file. If none is found, DOS then checks for an .EXE file. Finally, if an .EXE file is not found, DOS checks for a .BAT file by that name. If none of these exist, the "Bad command or filename message" appears.

When we say that "any valid DOS command" can be in a batch file, this includes the name of a program that is to be run. The following list of commands could be placed in a batch file to start up WordStar on a computer with two floppy disk drives.

Filename: WSGO.BAT	*Explanation*
b:	Shift to drive B.
dir/w	Get directory of B.
del *.bak	Erase all files ending in .BAK.
a:ws	Run WordStar from drive A.
dir/w	After exiting WordStar, get a directory of B.
a:	Go back to drive A.

This list brings up an important point about batch files. After the DEL *.BAK command, the A:WS command runs WordStar from drive A. When the user is finished and exits WordStar, DOS comes back to this batch file and continues to execute the list of commands, starting where it left off. The DOS keeps track of the commands it executes from a batch file. In fact, the DOS also keeps track of what disk it started a batch file from. If we switch disks for some reason in the middle of processing a batch file, when it is time for the DOS to get the next command in the list, we are prompted to replace the disk with the batch file on it so execution can continue.

If one of the commands in the list is invalid, DOS gives a "syntax error" message, execution of the batch file ends, and the DOS prompt reappears on the screen. Processing of the batch file can be interrupted by typing in a Ctrl-C or Ctrl-Break. The following message appears:

```
Terminate batch job (Y/N)?
```

Answering with a y for Yes aborts the processing of the batch file, while answering n for No causes the current command to be skipped and processing to continue.

Batch files are especially useful for dealing with subdirectories. They can greatly simplify moving among subdirectories. Let's look at a situation that shows this. We have a computer with a hard disk, and the hard disk is organized as shown in Figure 8-1. Pam needs to do some word processing. She has her own subdirectory (\wp\pam) for word processing files and must first move into that subdirectory. Also, as discussed in the previous chapter, the "overlay" files of WordStar must be copied into that subdirectory before she can start using the WordStar program. Rather than have to type in the various commands involved here every time she wants to do word processing, she creates a batch file containing all these commands and names it WPPAM.BAT.

Filename: WPPAM.BAT

```
path \wp
cd \wp\pam
dir/w
del *.bak
copy \wp\ws*.ovr
ws
del ws*.ovr
dir/w
cd \
```

To start this batch file, Pam simply types in WPPAM. This assumes that the batch file is located at the root level of the hard disk, and that the root level is the current directory. DOS then finds this batch file, looks at the list of commands, and begins to execute them in the order they are listed.

The first command establishes that if a program to be run is not found in the current directory, then the \WP directory should be checked. The second command makes Pam's subdirectory the current directory. The DIR/W command puts a listing of the files in Pam's directory on the screen. The DEL command gets rid of all the files ending in .BAK (something that must be done periodically when using WordStar). The COPY command gets the two overlay

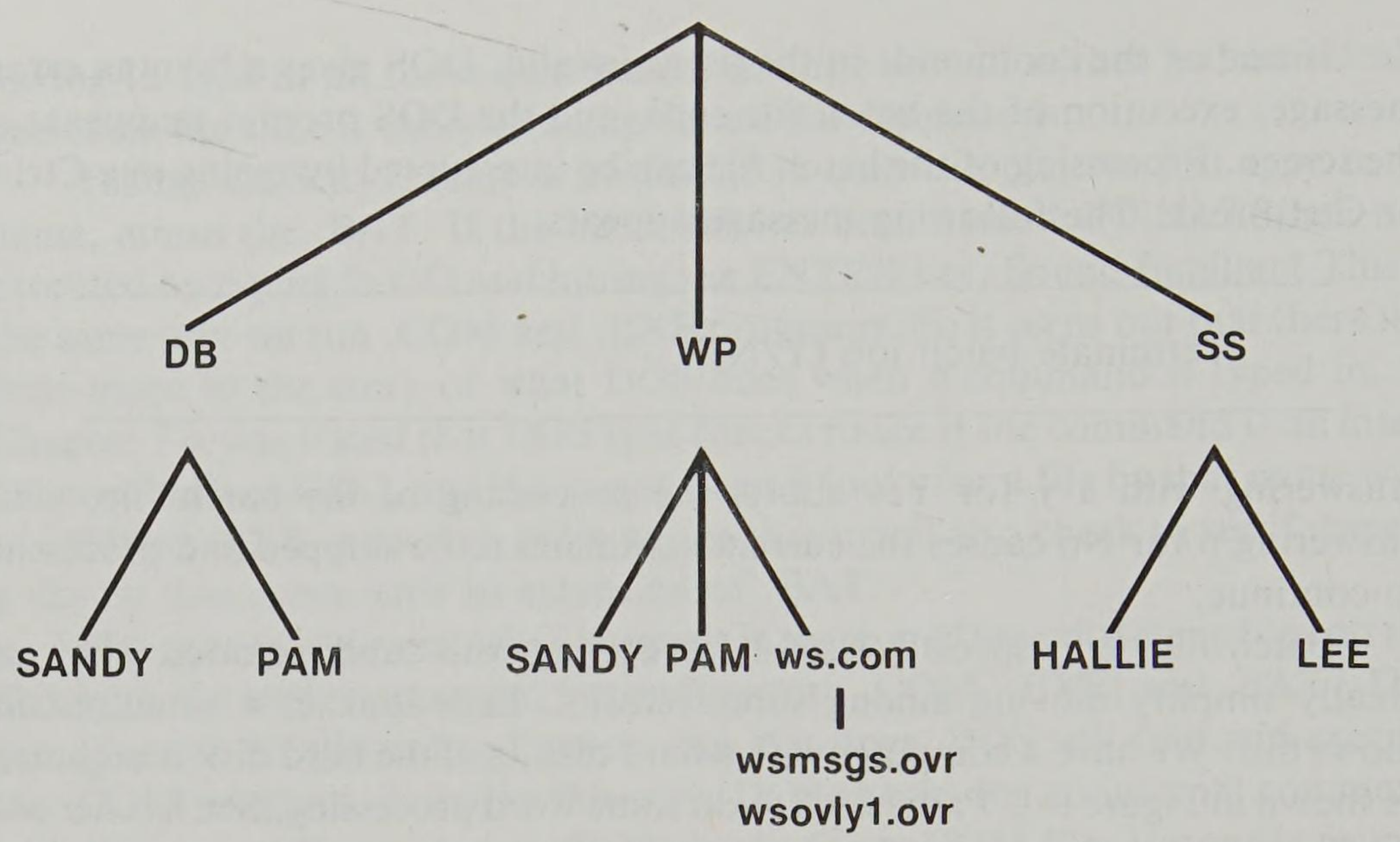


Fig. 8-1. Organization of a Hard Disk

files from the \WP subdirectory and puts them in Pam's subdirectory. No destination is specified in the COPY command, so the files are copied into the current directory, which is Pam's directory. The sixth command runs the WordStar program. DOS first searches for the WS.COM file in the current directory. Since it is not there, the \WP directory is then searched, according to the PATH command. When Pam is finished using WordStar and exits that program, DOS goes back to the batch file and continues executing the commands in the list. The DEL command erases the two overlay files, and then a directory of Pam's drive is listed on the screen. Finally, the root directory is made the current directory.

This batch file not only saves Pam from typing in a lot of commands, but it also executes all of those commands much, much faster.

8.2 How to Create a Batch File

A batch file is a text file. There are three different ways a text file can be entered and saved to disk. One is with a word processing program, another is by using the COPY command, and a third is by using the DOS editor, the EDLIN program.

A word processing program is a text editor that also has sophisticated capabilities for formatting the output sent to a printer. Using a word processing program to enter a batch file is contingent on the program being able to save a file as a standard ASCII text file (see the discussion of what a text file is in Section 5.5). The word processor Volkswriter, for example, saves its files as text files, so it can be used to type in a batch file with no problem.

Some word processing programs store their documents on the disk in formats other than standard ASCII text files. Programs that do this may have a way to convert their files to text files, or a way to switch modes.

For example, MultiMate does not use text files, but it has a conversion program. After exiting the main MultiMate program, the user runs the FILECONV program, using the Utility disk. A menu appears that includes "MultiMate to ASCII" conversion. WordStar also does not normally use standard ASCII text files but can be told to do so. This is done by using the "nondocument" mode (opening a file with the N command) as opposed to the "document" mode (opening a file with the D command).

Using a word processing program is the best way to enter a batch file when the file is a long one with lots of commands. The drawback is that the word processing program must be on one of the disks currently in use by the computer. For small batch files with just a few commands, it may be easier to use the COPY command.

Since COPY is an internal command, it can be used at any time. There is a way to take input from the keyboard and COPY it directly into a file on the disk. We must be aware, though, that the keyboard-and-screen combination is often referred to as the CONSOLE, and that DOS has its own name for the console, which is CON. Just as the disk drive names, A, B, and so on are followed by a colon when used in commands, so is CON. So, to tell DOS to take the input from the keyboard and place it in a text file and then save this file to disk when we have finished typing it in, we tell it to COPY from the console to the specific file. To create a batch file named START.BAT, the command would be:

```
COPY CON: START.BAT
```

This says, "Take the input from the keyboard, store it temporarily in the computer's memory, and when finished, save the input to disk as a text file, using the name START.BAT." Controlling where the input comes from is covered in detail in Chapter 12.

To take input from the keyboard and COPY it directly into a file on disk, we must also know how to indicate that we have finished typing in the text. When a command such as the one shown above is entered, the cursor goes to the next line, and whatever we type in will be in the text file. At the end of every line, we hit the ENTER key, and the cursor goes to a new line. After hitting ENTER at the end of the last line, we use a Ctrl-Z followed by ENTER to tell DOS we have finished. Ctrl-Z is the "end-of-file" command in text files. It can be typed by holding down the Control key and pressing Z one time, or by using the F6 key.

Here is an example of what the screen would show if COPY were used to enter this small batch file:

```
A>copy con: test.bat
date
time
dir/w
dir b:/w
^Z
A>
```

The caret and Z actually appear on the screen when a Ctrl-Z is entered. After the DOS prompt returns to the screen, this batch file can be used by typing in TEST.

DOS has its own editor, the EDLIN program, that can be used to enter large batch files. Using the editor portion of a word processing program is preferable, because EDLIN is not very sophisticated. But if none is available, EDLIN can get the job done. How to use EDLIN is covered in Chapter 9, before we get into really complicated batch files.

8.3 Auto-Starting with AUTOEXEC.BAT

The booting process includes a step that has not been mentioned before. It was stated earlier that after the bootstrap loader program loads all the DOS programming from the disk into the memory, the DOS asks for the date and time. Actually, something else happens in between.

Before asking for the date and time, DOS checks the current disk for a special batch file. If this batch file exists on the disk, DOS will immediately begin executing the commands contained in the file. Its name must be AUTOEXEC.BAT.

Here is an example of a simple AUTOEXEC.BAT file that asks for the date and the time and then runs the Lotus 1-2-3 program.

Filename:
AUTOEXEC.BAT

	Explanation
date	Get the system date.
time	Get the system time.
123	Run Lotus 1-2-3.

The DATE and TIME commands are included so that if an AUTOEXEC.BAT file is found, only the commands contained in it will be executed. It is a good idea to have the DATE and TIME commands always included in an AUTOEXEC.BAT file. Of course, if the computer has a clock board or a multifunction board that includes a clock, the command to read

this clock can be included, rather than the DATE and TIME commands. For example, with boards from AST Research that include a clock, the manufacturer includes a program, ASTCLOCK.COM, that reads in the date and time automatically. As long as this program is on the boot disk, the DATE and TIME commands can be replaced:

Filename:
AUTOEXEC.BAT *Explanation*

astclock	Run the ASTCLOCK program.
123	Run Lotus 1-2-3.

AUTOEXEC.BAT files can be used not only to start up a program automatically but can also include many preliminary commands that might be needed to set up the computer properly. For example, if a serial printer is being used, it may be necessary to tell DOS to send output to the serial port, and to specify what the format of the output is. This can be done with the MODE command. To set all this up and run WordStar on a two-floppy-drive system, we might have an AUTOEXEC.BAT file that looks like this:

Filename:
AUTOEXEC.BAT *Explanation*

date	Get the system date.
time	Get the system time.
mode com1:1200,n,8,1,p	Set up serial port one.
mode lpt1:=com:1	Use serial port one.
b:	Switch to drive B.
del *.bak	Erase all files ending in .BAK.
a:ws	Run WordStar from A.
dir/w	Get a directory of B.
a:	Go back to drive A.

The MODE command is covered in Chapter 12. It is an external command, so MODE.COM would have to be on drive A (or drive C for an XT).

In a different situation, where this "setup" is taking place via an AUTOEXEC.BAT file, but no program is automatically being started, it would be nice to be able to leave a message about what has been setup. In Chapter 10, "Advanced Batch File Usage," some commands are covered that put out one-line messages, but they have limitations. What we want is to be able to put a nice, neat message on the screen with some lines centered, and so on.

To do this, we can use the TYPE command. It lists a text file on the screen. So all we have to do is create a text file that contains the messages we want, then use the TYPE command to display the contents of this file. A word processing program's editor or the DOS editor EDLIN.can be used to enter the file. For example, we might have an AUTOEXEC.BAT file that contains the following commands:

Filename:
AUTOEXEC.BAT

	Explanation
date	Get the system date.
time	Get the system time.
mode com1:1200,n,8,1,p	Set up serial port 1.
mode lpt1:=com1:	Use serial port 1.
cls	Clear the screen.
type start.msg	Display the text file on the screen.

On the same disk, we have a text file named START.MSG that contains the following lines:

Filename: START.MSG

The serial port has been set up properly.

The printer can now be used.

When the disk is booted, the AUTOEXEC.BAT file is found, and so each of the commands in it is executed. The DATE and TIME commands ask for input, the MODE command sets up the computer to use the serial port, the screen is cleared with the CLS (CLear Screen) command, and the START.MSG file is listed at the top of the screen. When that process is finished, the DOS prompt appears, and any valid command can then be entered. There are more examples of AUTOEXEC.BAT files in Chapter 10, "Advanced Batch File Usage."

8.4 Replaceable Parameters

There are times when we need to vary what goes into a batch file. Rather than having to redo the batch file every time or to edit it and change a line or two, we can include from the beginning a replaceable symbol. This is especially useful if one element of the batch file varies every time it is invoked.

The symbol used is the percent sign (%), followed by a number between zero and nine. Wherever the percent sign appears, along with its number, in the file, it will be replaced by something when the batch file is executed. The

item that replaces it must be typed in with the command to start the batch file. For example, let's say we have a batch file named DOBASIC.BAT, and that file contains the command BASIC %1.

Filename: DOBASIC.BAT

```
basic %1
```

The %1 is replaced by whatever is typed in right after the DOBASIC command.

```
dobasic game
```

This command causes the batch file to run, and when it reaches the %1, to plug in GAME in place of %1. After the BASIC interpreter is placed in the computer's memory, the GAME.BAS program will be run. The next time we start up, if we want to run STARTREK, the command would be:

```
dobasic startrek
```

Of course, this is a simplistic example; rather than have a batch file to do this, it might be just as easy to type in BASIC GAME or BASIC STARTREK ourselves.

Let's look at a more realistic and useful example. Sandy wants to use the dBASE II program while located in her subdirectory \DB\SANDY. dBASE II is located in the \DB subdirectory. Sandy has three different data base files she uses, a ROSTER of people, a PHONE directory, and an inventory of BOOKS, and these are located in her subdirectory.

Each time she wants to use one of these data base files, she will have to move into her subdirectory and set a PATH to the \DB subdirectory (so dBASE II can be found). Then she will have to give dBASE II a couple of directions, including which file to use. While she's actually using dBASE, the sequence of commands to start it and then use the data base file called PHONE might look like this:

```
A> dbase
.set bell off
.set intensity off
.use phone
.erase
```

In the same way that a batch file can be used to enter a list of commands to

DOS automatically, a "command" file can be created in dBASE to enter a sequence of valid dBASE commands. One of these can be created for each of the three different startup routines that she needs. Sandy creates these, naming them ROSTER, PHONE, and BOOKS. These "command" or "program" files can then be used when starting up by typing in:

```
dbase roster
or
dbase phone
or
dbase books
```

Each of these has the appropriate commands in it to initiate use of the appropriate data base file. Now if she wants to use a batch file to start up, it is *not* necessary to have three different files, one saying DBASE ROSTER, one with DBASE PHONE, and one with DBASE BOOKS. Instead, the batch file contains a replaceable parameter: DBASE %1. The batch file, which can be named DBGO.BAT, looks like this:

Filename:	
DBGO.BAT	*Explanation*
path \db	Set a search path to \DB.
cd \db\sandy	Move into Sandy's subdirectory.
copy \db*.ovr	Bring in the dBASE overlay file.
dbase %1	Run dBASE. Plug in the first parameter.
del *.ovr	Erase the overlay file.
cd \	Go back to the root level.

If Sandy wants to use the ROSTER file, the command she enters to start is:

```
DBGO ROSTER
```

The batch file DBGO is executed. The PATH command tells DOS to search for programs in the \DB subdirectory. Sandy's subdirectory is made the current directory, and the dBASE overlay file is copied into that directory so dBASE can be used. Then, when dBASE is run by the next command, the %1 is replaced by ROSTER, because this was entered along with the command to start this batch file. This causes DBASE ROSTER to be entered as the command. When Sandy has finished using dBASE, the DEL command is executed, erasing the overlay file, and the root directory is made the current directory. When that is done, the DOS prompt reappears on the screen, and DOS waits for the next command to be typed in.

In the examples above, we've had only one replaceable parameter. There can be up to nine of them (numbered one through nine). Furthermore, the batch file was set up to be used only by Sandy. It would probably be wise to name it something like DBSANDY.BAT (or something similar) because Pam would also need a batch file to start up dBASE and use her particular files. After a while, we could have the root level of the disk cluttered up with all kinds of batch files. It is not difficult, though, using more than one replaceable parameter, to have one batch file that can be used by everyone, to start up word processing, data base, or spreadsheet operations!

Instead of entering exact paths with the CD command, and exact names of the programs which are to be started, we can use replaceable parameters. Then, when invoking the batch file, the user follows the file name with the appropriate items to be plugged into the batch file. The first item that comes after the batch file name gets plugged into %1, the second into %2, the third into %3, and so on. If %0 is used, the batch file whose name contains %0 is plugged in.

For example, let's look at a file that would accommodate everyone using a hard disk. The hard disk is organized as shown in Figure 8-2. Note that the subdirectory names are the same as the names of the main program they hold. This is done to make it easier for the users to remember these subdirectory names. Further, each person's subdirectory has his or her own name on it.

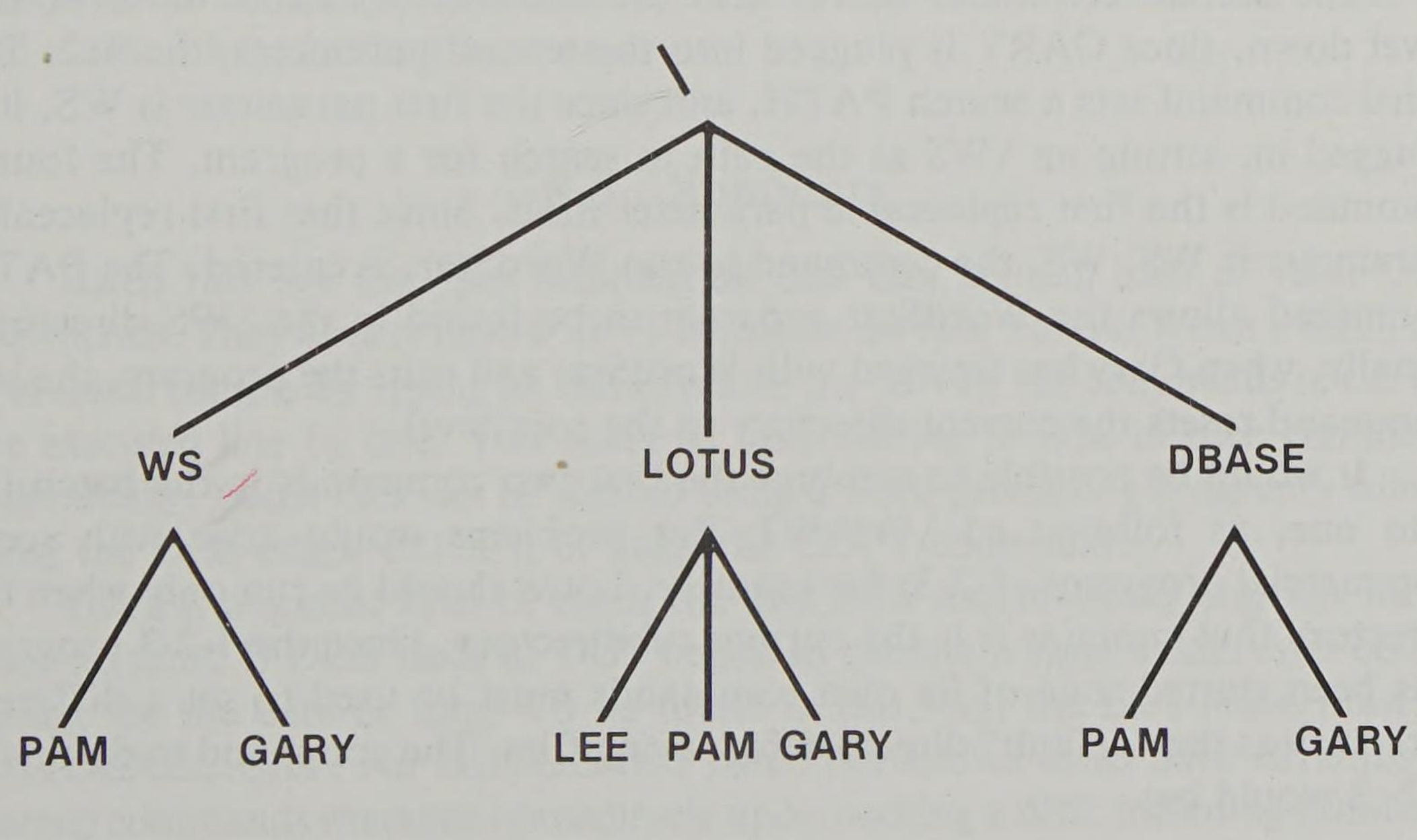


Fig. 8-2. Naming Subdirectories after People and Programs

The batch file START.BAT has the following commands in it:

Filename:	
START.BAT	*EXPLANATION*
cd %1	Change directory to <first name>.

```
cd %2       Change directory to <second name>.
path \%1    Set search path to <first name>.
%1          Run the program.
cd \        Go back to the root level.
```

The command for Gary to use WordStar is:

```
                parameter number 1
                |
START WS GARY
                |
                parameter number 2
```

The batch file named START is executed. The first command moves into the subdirectory WS, since WS is plugged into the %1 parameter.

Reminder: When the CD command is followed by one name and no \or other path information, it moves down one level into the subdirectory with that name.

The second command moves into the subdirectory called GARY, one level down, since GARY is plugged into the second parameter, the %2. The third command sets a search PATH, and since the first parameter is WS, it is plugged in, setting up \WS as the path to search for a program. The fourth command is the first replaceable parameter itself. Since that first replaceable parameter is WS, WS, the command to run WordStar, is entered. The PATH command allows the WordStar program to be found in the \WS directory. Finally, when Gary has finished with WordStar and exits the program, the last command resets the current directory to the root level.

It would be possible to combine the first two commands in the batch file into one, as follows: cd \%1\%2. But problems would arise with some commercial programs – 1-2-3, for example. Lotus should be run only when the directory that contains it is the current subdirectory. Once the 1-2-3 program has been started, one of its own commands must be used to set a different directory as the "default" directory for saving files. The command to do this in 1-2- 3 would be:

```
/ WORKSHEET GLOBAL DEFAULT DIRECTORY
```

After this command is entered, the old directory must be erased with the BACKSPACE key, and the new path entered – for example, C:\LOTUS\GARY.

Since this is the case, if Gary wants to use the Lotus 1-2-3 program, all he enters is:

```
START LOTUS
```

Since no second parameter is entered, when the second command in the batch file (CD %2) is encountered, there is nothing to plug in for the %2. This causes the command to be seen as simply CD, which is a valid command. This command asks to see the path to the current subdirectory. After this is displayed on the screen, processing of the batch file continues.

If the alternative command, CD \%1\%2 is used, an error message results when there is nothing to plug into %2:

```
Invalid directory
```

The batch file processing is halted, and the DOS prompt reappears. So, we see that putting in the command on two lines instead of one leaves more options open.

This particular batch file ignores the issue of overlay files. To use it, the appropriate overlay files would already have to be in any of the subdirectories where they would be needed.

This batch file also points out that as we get more and more familiar with the various DOS commands covered in this book, and with how batch files work, we can create a lot of nifty shortcuts.

8.5 Summary

Batch files are text files recorded on disk that contain lines of valid DOS commands. They always have a .BAT extension on their names. When a batch file is invoked (simply by typing its name, minus the .BAT), the commands in the file are executed line by line. This saves us from having to type in each command individually. Batch files can be entered using a word processing program's editor, using the DOS editor EDLIN, or using the COPY command.

There is a special kind of batch file that DOS automatically searches for at boot-up time. If DOS finds it, DOS begins to execute it right away, even before asking for the date or time. To be found in this way, the file's name must be AUTOEXEC.BAT. An AUTOEXEC batch file allows us to have all kinds of startup commands executed immediately upon booting a disk, including running a particular applications program.

The command to start a batch file (its name) can be accompanied by one or more parameters. These extra strings can then be plugged into the batch file, replacing special codes that are embedded in the file. The special codes consist of a percent sign (%) and a number between zero and nine. The first parameter is plugged in anywhere that the symbol %1 appears, the second anywhere there is a %2, and so on. The %0 parameter causes the batch file name to be plugged in. These replaceable parameters allow us to use the same batch file to start different programs, or to move into different subdirectories each time.

CHAPTER 9

Using the DOS Editor: EDLIN

DOS comes with its own editor, EDLIN. EDLIN is short for EDIT a LINE. The name comes from the fact that it is a line editor, as opposed to a screen editor. This means that editing must be done a single line at a time. We cannot just move quickly around the screen changing lines, as with most word processing programs. If you have a word processing program that can produce standard ASCII text files, just ignore the EDLIN editor and use your own word processing capability. Do read Section 9.4, as this covers the editing keys. These editing keys can be used when entering DOS commands, as well as when using EDLIN.

EDLIN is a program on the DOS disk. The EDLIN. COM file must be available on a disk that is "on line" in order to use this editor. EDLIN is used for entering text into the computer's memory and then saving this text onto a disk as a standard ASCII text file. For a description of ASCII text files, see Section 5.5, "Listing a Text File on Screen."

Probably the most common use for EDLIN is to create batch files. It may also be used to type in what is called the "source code" of a program. Source code files contain all the lines of instructions that make up a program.

9.1 Creating a File

To start up the EDLIN program, we simply type in its name, followed by a space and then the name of the file to be edited. To start a file named SAMPLE, the command is:

```
EDLIN SAMPLE
```

When we use this command, we are assuming that the EDLIN. COM file is on the current disk, and that we want the new file to be on that same disk also. If we are located on drive B, which is where we want the file to go, and EDLIN is on A, the command is:

```
A:EDLIN SAMPLE
```

If EDLIN alone is typed in, DOS returns the error message:

```
File name must be specified
```

If SAMPLE is a brand-new file, the screen looks like this:

```
B>A:EDLIN SAMPLE
New File
*
```

The asterisk is the "prompt" from EDLIN. It tells us that the EDLIN program is ready for one of its commands.

If SAMPLE already exists on the disk, then EDLIN loads it into the memory. Then, instead of "New file," the message says either "End of Input" if the entire file could fit in the memory, or it says nothing at all if the file could not all fit into the memory (rather insufficient in the message department, isn't it!), and loads as much as it can until 75 percent of the memory is occupied.

Once the EDLIN prompt (*) is on the screen, we enter text, change lines, add lines, delete lines, search for certain strings, and so on. Let's look first at how text entry is started. EDLIN does not automatically go into a "text entry" mode as we might expect. An explicit command to do so must be entered. This command is I for Insert. EDLIN commands can be entered in uppercase or lowercase.

By typing in I and then hitting the ENTER key, we tell EDLIN that we want to add lines to the file. The screen looks like this:

```
B>A:EDLIN SAMPLE
New File
*i
     1:*__
```

Now the first line of the text can be entered. After each line is typed in, the ENTER key is used to go on to the next line.

EDLIN displays whichever line is current. After we have typed in one line and hit ENTER, the screen looks like this:

```
B>A:EDLIN SAMPLE
New file
*i
     1:*The DOS editor, which is named EDLIN, is not
     2:*__
```

As with many other computer applications, learning to use EDLIN really requires some first-hand experience with the program. Everyone should try to do what is described in this chapter.

When text is being entered, the BACKSPACE key is used to back up and erase any mistakes. Blank lines are put in simply by hitting ENTER and going on to the next line. Control characters can be entered by typing in a Ctrl-V followed by the uppercase letter of the control character desired. To enter a Ctrl-A, a Ctrl-V is entered, then A. This initially shows as ^VA on the screen. Later, when the file is redisplayed, it shows as ^A. The caret symbol (^) is often used to denote control characters.

Here is a small piece of text to type in and practice with. The text is shown as it looks on the screen.

```
 1:*The DOS editor, which is named EDLIN, is not
 2:*very good. It can only work on one line at a
 3:*time, and there isn't much consistency in the
 4:*format of the commands.
 5:*It can be useful when used to enter a batch
 6:*file that has a dozen or two dozen lines.
 7:*This can be done with the COPY command, too,
 8:*but if a mistake is made and it is not noticed
 9:*until after the line has been entered, the file
10:*must be reentered from the beginning. Using
11:*EDLIN, changes can be made fairly easily.
12:*_
```

When we have finished entering the lines of text, we can tell EDLIN to leave the text entry, or Insert, mode by entering a Ctrl-C. The screen then looks like this:

```
     10:*must be reentered from the beginning. Using
     11:*EDLIN, changes can be made fairly easily.
     12:*^C
*_
```

We are out of the Insert text mode and back to the EDLIN command prompt. The command to save this text file on the disk is E for End. When this command is used, the file is recorded on the disk under the name used to start up, and use of EDLIN ends. The EDLIN program automatically adds the Carriage Return and Line Feed characters to the end (these make a computer or printer go to a new line) and the End-of-File marker, a Ctrl-Z. The DOS prompt reappears, and we can go on with our work. The fact that we cannot save this file and continue working in EDLIN is a good example of why EDLIN is not a very good editor.

If, after typing in some lines, we decide not to save the file on disk, we can use the Q for Quit command. When this command is entered at the EDLIN prompt, the message "Abort edit (Y/N)?" appears. Answering y for Yes ends our use of EDLIN without saving the file on disk.

To get back into EDLIN and the SAMPLE file, we use the same command as we did the first time. Only this time, since the file already exists, it is loaded into memory from the disk, and the "New file" message does not appear. Instead we see "End of input file" (because the file is small, and all of it fits into the available memory):

```
B> a:edlin sample
End of input file
*_
```

9.2 Referring to Specific Lines

Almost all of the commands in EDLIN require parameters. These parameters are usually line numbers, and most of them come before the command itself. For example, the Insert command can be preceded by a line number to indicate where in the text a new line should be added. If the command says 6I, then the new line will be line 6, and the old line 6 will become line 7. All subsequent line numbers will also increase by one. The screen shows this:

```
*6i
    6:*The new line is entered here.
```

The general form of this command can be stated as:

```
nI
```

where n means put in some number. This is a direct way of referring to a

specific line. There are actually five different ways to refer to lines in EDLIN commands. To understand them, we need an explanation of the notion of "current" line.

If line 6 was entered into the SAMPLE file as described previously, the cursor would then be on line 7. If we terminate text entry with Ctrl-C, and then use the L (List) command to see what's there, the screen will show:

```
    1: The DOS editor, which is named EDLIN, is not
    2: very good. It can only work on one line at a
    3: time, and there isn't much consistency in the
    4: format of the commands.
    5: It can be useful when used to enter a batch
    6: The new line is entered here.
    7:*file that has a dozen or two dozen lines.
    8: This can be done with the COPY command, too,
    9: but if a mistake is made and it is not noticed
   10: until after the line has been entered, the file
   11: must be reentered from the beginning. Using
   12: EDLIN, changes can be made fairly easily.
*
```

The asterisk by line 7 marks it as being the "current" line. Usually the current line is the last one that was used. The asterisk can be thought of as a pointer.

The five different ways to refer to lines are by entering:

1. The line number itself, as in this last example.
2. A period. This can be used as an abbreviation or standin for the current line number. The command .I will cause a new line to be inserted at the current line number.
3. A pound sign (#). This means go to the line that is one more than the last line. The command #I will cause a new line to be inserted after the last line that exists at the time.
4. A number relative to the current line number. A −6 means six lines back; a +8 means eight lines forward from the current line. The command −5I will cause a new line to be inserted five lines previous to the current line.
5. Nothing. Entering nothing causes the parameter to use what is called "the default." This is some arbitrary value that will be put in for us automatically. The default varies according to the command being used. The command I will cause a new line to be inserted at the current line. That's just how Insert happens to work.

In giving the general form of the EDLIN commands in this chapter, a

consistent notation will be used. Whenever a command takes just one line number as a parameter, as does the Insert command, we use a lowercase letter n to indicate this. Many commands take starting and ending line numbers. For these, x indicates the starting number and y indicates the ending number. For those that also have a number indicating where things go to, this number is indicated by a z. In one command, r is used for Repeat.

9.3 Looking at What's in a File

Once lines have been entered and we are no longer in the Insert mode (the * prompt is on the screen by itself, with no line number), these lines can be displayed on the screen with the List command. The general form of this command is:

---

x,yL

Very often, L will be used all by itself. That's an example of putting in "nothing" where line numbers are normally entered. With this command, the default situation is then to display 23 lines, 11 before and 11 after the current line. Other abbreviations are possible with this command. 6L says, "Start the listing at line 6 and go for 23 lines (or until there are no more)." The command ,25L tells EDLIN to list the 11 lines that lead up to 25 (the ,25 is the y or ending number). Note that EDLIN will display on the screen *only* lines that we explicitly ask to see. It never lists lines on its own.

If a listing of more than 23 lines has been specified (e.g., 1,46L) then the lines will continue scrolling up the screen until the last one is reached unless Ctrl-NumLock or Ctrl-S is used. Either of these halts the scrolling temporarily. It is resumed by hitting any key.

The List command does not change the current line number. The Page command is different. It causes the current line number to switch to whatever line showed up last in the screen listing. Otherwise, it behaves like the List command. The general form of the command is:

---

x,yP

Sometimes we don't know what line number we need. EDLIN has a Search function that allows us to go looking for a specific string of characters, so we can find the line where we need to be. The general form of this Search command is:

---

x,y(,?)S<*string of characters*>

The x and y give a range of line numbers to search through. The string of characters is typed in immediately after the S. The right and left angle brackets are just part of the notation here. The question mark is optional, which is why it is shown in parentheses. The parentheses are never typed into an actual command, they are just used here to indicate optionality. If the question mark is included, every time EDLIN finds the specified string, it displays the line it is on and then stops and asks:

O.K. ?

If we answer y for Yes, then searching halts. If we answer n for No, then searching continues. If the string is not found, EDLIN comes back with the message:

Not found

If the question mark option is not included, then searching only goes as far as finding the first instance of the string. When a string is found, the whole line that contains it is displayed on the screen. If the search process has ended, the last line displayed becomes the current line. This makes it very easy to edit this line if necessary.

9.4 Using the Editing Keys and Editing Commands

So far we have only seen how to enter and display lines. Now we need to get into making changes. Only one line at a time can be edited with EDLIN. In order to select a line for editing, we just type in its line number at the asterisk prompt. If no line number is used, we just hit the ENTER key by itself, and the current line is displayed for editing. Pushing the ENTER key again causes EDLIN to go on to the next line. Relative numbers like −5 (go back five lines) or +10 (go forward ten lines) can also be used.

Once a line has been selected for editing, we must use the DOS Editing keys to keep the text we want, insert new text, and take out unwanted text. The line is displayed in its entirety, and editing takes place below this display. For example, if we are still working on the file called SAMPLE, and we want to change line 3, we type in a 3, hit ENTER, and the screen looks like this:

```
*3
     3:*time. There isn't much consistency in the
     3:*__
```

The line is being held in a buffer. The DOS editing keys are used to do such things as bringing back the entire line, bringing back the rest of the line, bringing back one character at a time, bringing back all the characters in the line up to a specified character, and skipping through the line to a specified character. Most of these functions are invoked by using one of the ten function keys. For computers that do not have these function keys, the documentation will say how to type the command in. In the discussion here, we'll refer to the appropriate function keys.

After line 3 has been brought up for editing, there are a couple of ways to change it. One is to retype the entire line, but this is not very efficient if only one or two characters need to be changed. Let's see how to change "isn't" to "is not." To bring back the portion of the line that will stay the same, it is necessary to use the edit key that says "Bring back everything up to character x." The F2 key does this. When it is pushed once, nothing happens right away, but as soon as some character on the keyboard is pushed, the first part of the line, up to but not including that character, is brought back. To bring back the appropriate portion of line 3, we would enter F2n. This says "Bring back all the characters in the line until an n is reached, but do not include the n." The screen looks like this:

```
*3
3:*time. There isn't much consistency in the
3:*time. There is
```

The rest of the line is still being held in the buffer. Next it is necessary to push the Insert key (labeled Ins) once to tell EDLIN to add extra characters that are not contained in the buffer. Then a space and the word "not" are typed in. Now we have to tell EDLIN to skip over the "n't" still being held. To do this the edit key that says "Skip one character" is used. The Delete key (labeled Del) does this. Pushing it three times skips three characters. Then the edit key that says "bring back the remainder of the line," the F3 key, can be used. At this point the screen shows:

```
*3
3:*time. There isn't much consistency in the
3:*time. There is not much consistency in the_
```

The ENTER key is used to finish the editing. This new line is now line 3, and the old one is gone. The EDLIN prompt comes back, and the program is ready to go on to the next step.

Not all the editing keys were covered in this small example. Here is a summary of them:

F1	Bring back one character.
F2x	Bring back everything up to but not including the character that replaces x.
F3	Bring back the remainder of the line. If the line is still intact in the buffer, the entire line is brought back.
F4x	Skip over everything up to but not including the character that replaces x.
Del	Skip over one character.
Ins	Insert mode. Allow characters to be typed in while still leaving the line in the buffer as it is.
Esc	Cancel editing the current line.
BkSpc	Back up and erase one character on the screen and replace it in the buffer.

The only way to become really polished at using these commands is to use them a lot. Sometimes it takes a little ingenuity to accomplish what we want with these editing keys. There is always the old standby solution, which is to retype the entire line. These commands give us nowhere near the power and flexibility that are desirable in a good editor. For serious editing work, it is necessary to purchase one.

There are times, though, when these editing keys can be very handy. They work not only when using the EDLIN program, but also when entering DOS commands. The last DOS command line that was entered is stored in a special little buffer. All or part of that line can be brought back, just as a line can be brought back while using EDLIN.

Occasionally, the same DOS command must be entered several times in a row. An example might be when a set of files has to be copied onto several disks. Assuming that a wildcard can be used to copy all the files with one command, then after the first copy is made, another disk is placed in the target disk drive. Rather than having to type the entire command again, we can use the F3 key to bring it back. The line is only brought back, not actually entered, so the ENTER key must still be pushed after the line reappears.

Now we know how to edit a particular line. There are a few more EDLIN commands that are relevant to changing the text – in particular, the one to delete lines. The general form of the command is:

```
x,yD
```

As with the other EDLIN commands, the line numbers may be specified in different ways. If the D is entered all by itself, this means "Delete the current line." All subsequent lines will be renumbered to one less than what they were. A command like ,16D would mean delete the lines between the current line and

line 16. The command 6D would delete that extra line we inserted at line 6. The command 1,.D would delete from line 1 through the current line (the period is a way of referring to the current line).

Another command to change the text is the Move command. Its general form is:

```
x,y,zM
```

Again, x is the starting line number, y is the ending line number (defining the range of lines to be moved), and z is the line number where those lines will start after being moved. No lines are erased in this process. If the line number that the line or block of lines is being moved to already has text, the text will be pushed out of the way. The lines being moved may wind up being numbered differently than expected, if they are being moved to a higher line number. For example, if we issue this command:

```
2,4,10M
```

then lines 2, 3, and 4 are supposed to be moved so that they start at line 10. But when they are removed, this leaves a vacuum. Subsequent lines (5, 6, etc.) shift back three, and so does our new line 10. So instead of starting at 10, the three moved lines start at line 7. The old line 10, which can be thought of as having been temporarily moved up to line 13, slides back to line 10, ending up right where it started.

Sometimes lines must be reduplicated farther on in the text. The Copy command is used to do this. The general form of this command is:

```
x,y,zC
```

This will take all the lines starting at line number x and going through line number y and copy them to line number z. Line number z is the starting point where the lines are reduplicated. If text is already there (and in subsequent lines), all these lines are moved forward to make room for the new, reduplicated lines. This can save a lot of typing when a certain line or lines must be reused in the text.

Another useful command is Replace. It is used to find a string of characters and replace them with a different string. Its general form is:

```
x,y(,?)R<string1> ^Z<string2>
```

The x and y parameters specify the range of lines to be searched. The characters to be searched for are entered immediately after the R (the angle

brackets <> are not actually typed in), and then a Ctrl-Z is entered, followed by the string of characters to be used as the replacement. The Ctrl-Z shows up on the screen as a caret and Z (^Z). It separates the search string from the replacement string. A character like this is referred to as a "delimiter." When searching for a string of characters, the EDLIN program distinguishes between uppercase and lowercase. A search for "the book is mine" will *not* find "The book is mine."

If the optional ? is used, then each time string1 is found, EDLIN makes the replacement in the line, shows the line as it has been changed, and below it asks:

O.K.?_

Answering y for Yes accepts the change, and EDLIN continues its search for the next instance of string1. If string1 is found again, the same thing occurs. Each time the string is found, a Yes answer must be given to accept the change. Anything else disallows the change.

If the ? option is not included in the command, then EDLIN just goes through the entire range of lines specified, changing every instance of string1 that is found to string2. When this is done, all the lines that were modified are displayed on the screen.

The Replace command also can be used to remove all instances of a string. To do this, we just enter the first part of the command, including the search string (the string of characters to be removed), then the Ctrl-Z, and then nothing. This says, "Replace every instance of string1 with nothing." This is a fast way to delete material, but we must not overlook the fact that if string1 is a couple of words long, this may leave lines with very little text on them. EDLIN does not automatically combine lines or regroup lines.

9.5 Summary

It does not take long to realize that the DOS EDLIN is a rather crude, limited editor. For any serious editing, a good editor (which may be the editor of a word processing program) should be acquired. EDLIN may come in handy from time to time to quickly type in a batch file that is a little bit too long for using the COPY command.

To start EDLIN, its name is typed in, followed by the name of the text file to be created or, if the file already exists, to be edited. Example:

EDLIN SAMPLE

When EDLIN is up and running, its prompt, an asterisk, appears. To start entering lines of text, the Insert command must be used. To make changes to files, lines can be edited one at a time. The EDLIN commands normally consist of parameters referring to line numbers, and then a one-letter command. When giving generalized forms of the command, n is used to mean any line number, x to mean the starting line number, y to mean the ending line number, and z to mean the line number to move to. The EDLIN commands that were covered are:

x,y,zC	Copy. Copies lines in the range x through y to a different place (z) in the text file.
x,yD	Delete. Deletes a range of lines.
n	Edit. Edits a line number n.
E	End. Ends the edit session and saves the file to disk.
nI	Insert. Inserts a line at line number n.
x,yL	List. Lists a range of lines.
x,y,zM	Moves a range of lines to line number z.
x,yP	Page. Lists a range of lines. Makes the last line shown the current line.
Q	Quit. Leaves the EDLIN program without saving the file on disk.
x,y(,?)R<string1> ^Z<string2>	Replace. Searches through the range of lines to find string1 and replaces it with string2. If the ? option is used, stops and asks at each finding if the string should indeed be replaced.
x,y(,?)S<string>	Search. Searches through the range of lines for string. When it is found, displays that line. If the ? option is used, stops and asks whether the search should continue.
nT<filename>	Transfer. Merges a file from disk into the file currently being edited.

The editing keys are used to rework a particular line in the file. The editing keys that were covered are:

F1	Bring back one character.
F2x	Bring back everything up to but not including the character that replaces x.
F3	Bring back the remainder of the line. If the line is still intact in the buffer, the entire line is brought back.
F4x	Skip over everything up to but not including the character that replaces x.
Del	Skip over one character.
Ins	Insert mode. Allow characters to be typed in while still leaving the line in the buffer as it is.
Esc	Cancel editing the current line.
BkSpc	Back up and erase one character on the screen and put it back in the buffer.

CHAPTER 10

Advanced Batch File Usage and DOS Menus

10.1 Batch Subcommands

There are several DOS commands that can be used only in a batch file. They make a one-line message appear or halt the processing temporarily, for example. Some are very complicated to use and may be of little value to the average user.

ECHO

ECHO is probably the most useful of these batch subcommands. The command can be followed by nothing, or by one of these three items: ON, OFF, or a MESSAGE. Normally when batch files are being used, the command being executed is shown on the screen, just as if we had typed it in. If we use the ECHO OFF command, this no longer happens. Any output that the command might produce is still shown, but the command itself does not appear on the screen. For example, if ECHO OFF is followed by DIR C:, then the command DIR C: does not appear on the screen, but the directory listing that it produces is shown. If ECHO is ON, then the command DIR C: will also appear on its own line on the screen.

The ECHO command followed by nothing else causes the display of a message giving the status of ECHO – whether it is ON or OFF. This is a rather useless function, since we can tell from what happens on the screen whether it is ON or OFF. ECHO's default status is on. That is, unless a change is made, ECHO is ON.

Whether it is ON or OFF, when ECHO is followed by a one-line message, the message appears on the screen. For example, if the following command appears in a batch file:

```
ECHO This batch file erases *.bak files automatically
```

then the message will appear on the screen. What it looks like depends on whether ECHO is ON or OFF. If it is ON, this is what we get:

```
ECHO
ECHO This batch file erases *.bak files automatically.
```

If ECHO is OFF, then we get this:

```
This batch file erases *.bak files automatically.
```

Most of the time ECHO is turned OFF at the beginning of a batch file. When ECHO is OFF, messages that can be put in with the REM and PAUSE subcommands are *not* displayed.

REM

REM is short for REMARK. The REM command is followed by a message. If ECHO is ON, the letters REM and the message appear on the screen. If ECHO is OFF, no part of the REM line appears on the screen. Since ECHO can be used to display nice, neat messages, the only thing REM need be used for is to put a one-line reminder in a batch file about what the file is actually accomplishing. As long as ECHO is OFF, the message never appears on the screen. It is for use by the person who made up the batch file, allowing notes about how the batch file works, or why a particular command was put in where it was.

PAUSE

The PAUSE command causes execution of the commands in the batch file to be halted temporarily. The following message appears on the screen:

```
Strike any key when ready . . .
```

PAUSE can be followed by a message. For example:

```
PAUSE The copying is done.
```

If ECHO is ON, this will produce:

```
PAUSE The copying is done.
Strike any key when ready . . .
```

If ECHO is OFF, only the line with "Strike any key . . ." appears. It should be quite obvious by now that the way to put neat, tidy one-line messages on the screen is to turn ECHO OFF and use the ECHO command followed by the one-line message. PAUSE should be used anytime a message is placed on the screen that will require some time to read.

ECHO cannot be used to put in blank lines, since an ECHO command followed by nothing produces an ECHO ON or ECHO OFF message. Also, ECHO can be used only for one line at a time. If a long message is needed, create a text file that includes the message. This makes it easy to use blank lines and to center text. Then the TYPE command can be used in the batch file to list this little text file on the screen.

GOTO

The GOTO command is accompanied by a LABEL. Labels in a batch file are preceded by a colon. When a GOTO LABEL is used, execution of the batch file jumps to the line after the one that has the LABEL. Each LABEL must be all alone on a line. When batch file processing reaches a line with a label, this line is simply ignored, and the processing continues on to the next line. Here's an example of how a batch file can be used to copy the same files over and over again from the disk in drive A onto different disks placed in drive B:

Filename: REPEAT.BAT

```
echo off
:start
cls
echo Place the disk to be copied to in drive B.
pause
copy abc.zzz b:
copy xyz.456 b:
copy 999.old b:
echo Files copied.
echo To end the copying process, enter a Ctrl-C.
echo Otherwise, to continue,
pause
goto start
```

When the last line is reached, it tells DOS to go back and start again right after the :START label. Processing continues with the next line, the CLS command. This creates what is called, in programming jargon, an infinite loop. Every time the end is reached, processing is sent back to the beginning again. The only way to break out of this loop is with a Ctrl-C or Ctrl-Break. That's why a message to that effect is needed in the file.

IF

The IF command can be used to check for certain conditions. If a condition is true, then one thing happens; if it is false, something else happens. The general form of the command is:

IF (NOT) condition command

The NOT is in parentheses to indicate that it is optional. There are three types of conditions that can be checked. One is insignificant and won't be covered here. The two that occur frequently are:

1. If a file exists on disk.
2. If a certain string of characters is exactly the same as another.

The exact way to enter these is shown here:

---

if exist <filename> <command>
Example: if exist *.bak erase *.bak

if string1 = =string2 <command>
Example: if %1 = =dbase cd\dbase

After the condition (if exist <filename> of if string1 = =string2) comes any valid DOS command. The command following the condition is reached *only* if the condition is TRUE. If it is FALSE, then execution of the batch file drops down to the next line. One way this can be stated is, "If the condition is true, then do the command; otherwise go on to the next line."

Here's an example that might be placed in a batch file that automatically moves into a subdirectory and runs WordStar. Before starting WordStar, it checks to see if there are any of the bothersome backup files that need to be cleaned off all the time. If there are any, then the ERASE command gets rid of them.

Filename: WSGO.BAT

```
path \ws
cd \ws\gary
if exist *.bak erase *.bak
ws
```

This points out the fact that wildcards can be used in the conditional check to see if a file or files exist. Here's a more complex example, showing how the string comparison can be used, in conjunction with a replaceable parameter and GOTOs, to give a lot of flexibility to a batch file. Note that in string comparisons, there is a difference between uppercase letters and lowercase letters.

Filename: STARTER.BAT

```
echo off
cls
if %1 = = WS goto ws
if %1 = = DBASE goto dbase
if %1 = = LOTUS goto lotus
echo The entry after the batch filename was not
echo one of the 3 allowable options. The options
echo are WS DBASE or LOTUS. They must be in
echo uppercase letters. Try again.
pause
goto end
:ws
cd \ws
if exist *.bak erase *.bak
ws
goto end
:dbase
cd \dbase
dbase
goto end
:lotus
cd \lotus
lotus
:end
cd \
cls
```

To use this batch file, a computer operator must already know what the options are. Then something like this is typed in:

```
starter DBASE
```

When this is typed in, the batch file named STARTER is used, and DBASE is plugged in anywhere the %1 parameter appears. After ECHO is turned off, the string DBASE is compared to the string WS. Since they are not exactly alike, the comparison is false, so the command following, the GOTO WS, is not executed. On the next line, DBASE and DBASE are compared. This comparison results in a true, so the following command is executed, which sends DOS to the label :DBASE. All the lines in between are skipped over. The \DBASE subdirectory is made the current one, and the dBASE program is run. When dBASE is finished, execution of the batch file is picked up where it left off. The next command is GOTO END. This causes the lines that start LOTUS to be skipped, and the last two lines after the :END label are executed.

The ECHO commands right after the three string evaluations are there just in case the operator types in something that is not one of the three valid

choices. In programming jargon, this is called "error trapping." A message about the only possible choices is displayed and there is a PAUSE so the operator will have time to read the message. Then execution is sent to the end with the GOTO END, and the batch file finishes.

FOR

Sometimes we might want the computer to carry out the same command over and over, on a number of items in a list. When this situation arises, the FOR command can save some typing. The general form of this command is:

```
FOR (variable name) IN (list) DO (command with variable)
```

A variable is a parameter whose value can be changed each time it is used. The variable name in the FOR command is preceded by two percent signs (%%). Here's a situation in which we could use FOR: We need to use the COPY command four times, to copy to drive B some dBASE II files – the data base files (ending in .DBF), the index files (ending in .NDX), the report format files (ending in .FRM), and the memory variable files (ending in .MEM). Rather than type in the command four separate times, we can let the FOR command do the work for us. Here's what the command would look like:

```
for %%file in(*.dbf *.ndx *.frm *.mem) do copy %%file b:
```

This will cause the COPY command to be used four times. The first time, the variable name FILE, which is just a filler name, is replaced by the first item in the list, which is *.dbf. Thus the command

```
COPY *.DBF B:
```

is issued. The second time, the name FILE is replaced by the second item in the list, *.ndx, and so on. The COPY command is executed again and again until all items in the list have been used up, or, as they say in data base jargon, the data have been exhausted.

The first part, the FOR %%FILE, simply established that the variable name is FILE. IN is followed by the list of items that are to be used, in the order that they will appear. DO is followed by the command that is to be executed, and somewhere in that command the %%FILE appears. Wherever it appears, each time the command is executed the next item in the list is plugged into its place, because %%FILE is the "variable," the thing that changes each time.

Realistically, the usefulness of this command can be questioned. Looking back at it later, it may be difficult to decipher what the command is actually supposed to do, whereas a file with four COPY commands, like this next one,

might be a better one to have on disk. We can tell by looking at it what it accomplishes.

Filename: COPIER.BAT

```
copy *.dbf b:
copy *.ndx b:
copy *.frm b:
copy *.mem b:
```

SHIFT

In the FOR command, the list of items was fixed, and thus the number of times the command would be executed was fixed. If there are four items, the command takes place four times. The SHIFT command allows an indefinite number of replaceable parameters to be used in a batch file.

Here's a useless example, but one that shows how the SHIFT command works:

Filename: SAMPLE.BAT

```
echo off
echo %1
shift
echo %1
shift
echo %1
```

If the command to start this batch file were entered like this:

```
sample Hello from DOS
```

The result on the screen would be:

```
A> sample Hello from DOS

Hello
from
DOS
```

The first ECHO %1 command produces the "Hello," as this word is plugged in for parameter %1. The SHIFT command causes the replaceable parameter to shift over to "from." The next ECHO %1 then displays "from" on the screen. The last SHIFT changes the replaceable parameter once more, this time to "DOS." The word "DOS" is then displayed by the last ECHO %1

command. The SHIFT command causes the next item in the list to be used to replace the %1 parameter. The "list" used is everything typed in along with the command to run the batch file.

Now, let's look at an example that could be useful. In the explanation of the FOR command, we saw how it could be used to perform the COPY command a fixed number of times. The number depended on how many items were in the list. This time, we would like to set up a batch file that gives us the flexibility to enter just one specification of what to copy (e.g., just .DBF files), or several (e.g., .DBF, .NDX, .FRM files, etc.).

By using the GOTO command to create an infinite loop, along with SHIFT to make the replaceable parameter change in each cycle through the loop, we can do this. Here is a file that would produce such results:

Filename: SHIFTER.BAT

```
:start
copy %1 b:
shift
goto start
```

The last command sends the batch file processing back to the beginning each time, creating the infinite loop. If the command starting all this is:

```
shifter *.dbf *.ndx
```

---

then the first time through it copies the .DBF files. The SHIFT causes *.NDX to be the replaceable parameter, so the second time through, the .NDX files are copied. Then the SHIFT is encountered again, but there is nothing else to SHIFT to, so the third time through, the %1 has a null value, because there is nothing to replace it with. This causes the following message to be displayed:

```
A> COPY B:
B: File not found
      0 File(s) copied
```

(This assumes that ECHO was not turned OFF. Some of this would not be displayed if ECHO was OFF.) This will continue to take place, over and over, until processing of this batch file is halted with a Ctrl-C or Ctrl-Break.

Rather than have to worry about interrupting the execution of this batch file, the IF command can be used to place checks into it that will halt it when appropriate. This can be done by using either condition, the one that checks for the existence of a file, or the one that compares two different strings of characters.

First, here's the one that checks for an existing file:

Filename: SHIFTER.BAT

```
:start
copy %1 b:
shift
if exist %1 goto start
echo Done!
```

As many filename specifications as wanted can be entered after the command SHIFTER. When they have been used up, the next SHIFT causes the %1 to have no value (no more names to be used), and the condition will be false. There is no file on the disk with a name consisting of no characters! Since the condition is false, the GOTO START is not executed, and the batch file ends after the ECHO command.

The second way looks like this:

Filename: SHIFTER.BAT

```
:start
copy %1 b:
shift
if %1 = =end goto stop
goto start
:stop
echo Done!
```

When using this method, the operator must be aware that the string "end" must be typed in as the last entry on the command line. The command must look like this:

```
shifter *.dbf *.ndx *.frm end
```

If "end" is not typed in, this file will be in an infinite loop. The first method has the advantage of not requiring that any special code word be remembered.

These subcommands can be used to create very complex batch files. Most of the time this kind of complexity is not really needed. Often, how much these subcommands are used depends on how much an individual likes to tinker with the system just to see what can be accomplished.

10.2 Changing the DOS Prompt: PROMPT

The PROMPT command is a very handy one that can be used to change the

the message that DOS presents to us every time it is ready for a command. We normally see A> or B> , and so on. By entering PROMPT and then some message, we can change the prompt.

For example, if we want to have the DOS prompt say:

Enter your command:

---

then we would change it by typing in:

---

prompt Enter your command:

Notice that we don't have to worry about finding PROMPT.COM on some disk. That is, of course, because it is an internal command, despite the fact that the IBM documentation says that it is external.

The problem with this new prompt is that it is so different, we might forget that it means we can enter a DOS command, and we might miss knowing which disk drive is the current one. The solution is to include familiar things like the A> in the DOS prompt, along with anything else we want. In order to do this, some special codes must be used. There are several of these special codes that are used only with the PROMPT command. They are always preceded by a dollar sign, so that they can be interpreted as special codes, and not as a part of the text of the new prompt line.

The code $d causes the system date to be inserted into the prompt. If the prompt is changed this way:

---

prompt The date is $d

---

the system prompt, rather than being A>, would look like this (the actual date will vary, of course):

The date is Sat 3-17-84

Again, this is probably not a good prompt, since it is not obvious that a command is to be typed in immediately after it. Let's take a look at the important codes that can be used in the PROMPT command, then we'll explore some useful ways to change the prompt line.

---

t Time. Displays the system time.
d Date. Displays the system date.
n Current drive. Enters the letter of the current drive.
g > Enters the > character.

_	New line. Enters a Carriage Return and Line Feed. (This is the underline character.)
p	Current directory. Displays the path to the current directory.

Examine the following PROMPT command and compare it to what it produces:

```
prompt The date is $d$_The time is $t$_$n$g
```

The new DOS prompt (assuming drive A is current) becomes:

```
The date is Sat 3-17-1984
The time is 14:5:12.65
A>
```

The d_ puts in the date and then Carriage Return and Line Feed. Carriage Return and Line Feed together send the cursor to the beginning of the next line. The t_ puts in the time and goes to the next line. The $n says, "Put in the current disk drive," and the $g says, "Put in the > symbol."

Something like this can be included in an AUTOEXEC.BAT file on every disk (or a hard disk that is used to start up). The $p can also be included. Rather than having separate lines for everything, the first line can have all the inforamtion desired, and the second the A> or C>, whatever it might be.

Filename: AUTOEXEC.BAT

```
date
time
prompt $d      $t      The current directory is: $p$_$n$g
cls
```

With such an AUTOEXEC.BAT file, after we are asked for the date and time, the prompt will be changed, and when the screen gets cleared by the CLS command, the new prompt to appear in the upper left corner will be:

```
Mon  3-19-1984      18:20:31.92      The current directory is: A:\
A>
```

Another possibility would be to use most of this same information and include a line that says "Enter your command now."

Filename: AUTOEXEC.BAT

```
date
time
prompt $d $t Current directory: $p$__Enter your command now.$__$n$g
cls
```

This would produce the following prompt:

```
Mon  3-19-1984      18:20:31.92      Current directory: A:\
Enter your command now.
A>
```

The other, less important, codes are:

$	The $ symbol. Allows a dollar sign to be part of the text.
v	Version. Includes a message about what version of DOS is being used.
l	< Enters the < character into the new prompt.
b	| Enters the bar (|) character into the new prompt.
q	= Enters the equal sign (=) into the new prompt.
h	Backspace. Causes a backspace and erasure of the previous character.
e	ESCape. Performs the same function as hitting the Escape key.

If any of these are used in the new prompt string, they must be preceded by the $. The fact that there are many characters that are not used as special codes after the $ can be used to our advantage. If any other character is used, it is just ignored, and characters following this are then used as the new prompt. This allows us to use characters that would otherwise have special meaning to DOS, like a semicolon (;) or slash (/). To include these characters in a prompt, they can be preceded by something like $x, which will just be ignored.

This may come in especially handy when we want some spaces after the last character in the prompt line, so that the cursor will not be right next to this last character. For example, if we establish a new prompt like this:

```
prompt Enter command:
```

then when it comes time to type in a command after this prompt – for example, the DIR command – the screen will look like this:

```
Enter command:dir
```

To clean this up and include two spaces in the prompt line, we need to use a dummy code; otherwise, the last two spaces would just be ignored:

```
prompt Enter command:$x____
```

The two underscore characters here are used to show that you hit the space bar twice. After doing this, entering commands will be neater:

```
Enter command: dir
```

10.3 Chaining Batch Files

Since any valid DOS command can be included in a batch file, it stands to reason that the command to start another batch file could be contained in a batch file itself. This is called *chaining from one batch file to another* and opens up lots of possibilities to us.

Here's an example of a batch file, START.BAT, that contains a replaceable parameter that can be used to chain to any one of a number of different batch files. Let's say that Gary, one of the users of a hard-disk system, wants to do some word processing. His subdirectory for this is \WP\GARY. He knows that there is a batch file on the disk called WPGARY.BAT that will start things for him. He uses the START.BAT file to enter the new system prompt and to start his other batch file. The command to do this is:

```
start wpgary
```

Here's what the two files might look like:

```
Filename: START.BAT
        prompt $d   $t$__$n$g
        cls
        dir/w
        %1
```

The %1 is replaced by WPGARY, which causes the batch file by this name to be executed.

```
Filename: WPGARY.BAT
        cd \ws\gary
        path \ws
        del *.bak
```

```
ws
cd \
```

Note that after running WordStar (the line that has ws on it), the next line is CD \. When WordStar is finished, this will return Gary to the root level.

It is important to realize that although execution can return to a batch file after a program like WordStar is done, execution cannot return to an originating batch file when another batch file has been started. This is called *nesting* in programming lingo and is not allowed with batch files. It means that if a batch file is started from another batch file, that is the last command that can be executed from the original batch file. Any other commands that might have been in the original batch file will never be reached.

Yet there are times when we will want to return to an originating batch file. The solution is simply to have the second run the first one again. This leads into the topic of menus.

10.4 Creating DOS Menus

Although computer operators can include items after a batch file name that can be used in place of the parameters %1, %2, and so on, this does not help when dealing with AUTOEXEC.BAT files. We want to give operators some choices, ask which choice they want, and then react to what they type in. There is a way to do this.

We can display some messages and then change the prompt to make it look as though a program is running and it is time to answer a question. On a hard-disk system that has different subdirectories for different programs, the user could be presented with some choices that are contained in the MENU.BAT file, which is started by the AUTOEXEC.BAT file:

Filename: AUTOEXEC.BAT

```
date
time
menu
```

Filename: MENU.BAT

```
echo off
prompt Enter your choice now:$x____ (____ means spaces here)
cls
echo Your choices are:
echo 1 Word Processing
echo 2 Data Base
echo 3 Spreadsheet
echo 4 Return to DOS
echo Type in the number you want and hit ENTER.
```

Because the prompt was changed, after the last ECHO line is displayed, the screen will show, instead of A>, the following:

```
Enter your choice now:
```

The batch file is finished! The operator may or may not know this, but the next thing to do is to respond with a 1, 2, 3, or 4. What happens when a 2 is typed in, followed by the ENTER key? DOS looks for a .COM, .EXE, or .BAT file by that name. So then, all we need are batch files named 1.BAT, 2.BAT, 3.BAT, and 4.BAT that will care of doing what the menu says such a choice will produce. The batch file named 1.BAT might look like this:

```
Filename: 1.BAT
    cd \ws
    ws
    cd \
    menu
```

This little batch file moves to the WS subdirectory, starts the WordStar program, and when WordStar has finished, returns to the root level and runs the MENU batch file again. The 2.BAT and 3.BAT files would be very similar. The 4.BAT file would need to look something like this:

```
Filename: 4.BAT
    echo off
    prompt $d   $t   Directory: $p$_$n$g
    cls
    echo Returning to DOS
```

In this file, the DOS prompt changes. It becomes more like the usual prompt; it just includes a few extras about date, time, and current directory.

Now it's time to get fancy and elaborate. The ECHO command is fine for one-line messages. But if we want to create a very polished menu with blank lines and messages in all the right places, we can create a text file that contains everything to be displayed, then use the TYPE command to display this text file. Text files of this nature should be done with a word processing program or the DOS editor, EDLIN.

First we create a text file that looks like this when displayed on the screen:

```
                Main Menu
                ---- ----   (these are dashes typed
                            in on the line below
                            Main Menu)
                Choices:

          1 Word Processing
          2 Data Base
          3 Spreadsheet
```

```
4 Return to DOS
   Type in the number of your choice and hit ENTER.
```

This file is saved on the disk with a name like MENU.TXT. The MENU.BAT file would then look like this:

```
Filename: MENU.BAT
echo off
prompt Enter your choice now:$x____
cls
type menu.txt
```

When this last command is executed, a neat, easy-to-read menu appears on the screen. Then the DOS prompt appears, which has been changed to read "Enter your choice now: ." Batch files named 1.BAT, 2.BAT, 3.BAT and 4.BAT must be created. Each of the first three would contain, at the end, these two commands:

```
cd \
menu
```

Let's look now at how to allow different people to use different subdirectories. These users have a hard disk on their computer, which has the tree structure shown in Figure 10-1. What is needed here is a way to allow the user to include his or her name so that the right subdirectory will be used. This can be done easily enough with a replaceable parameter.

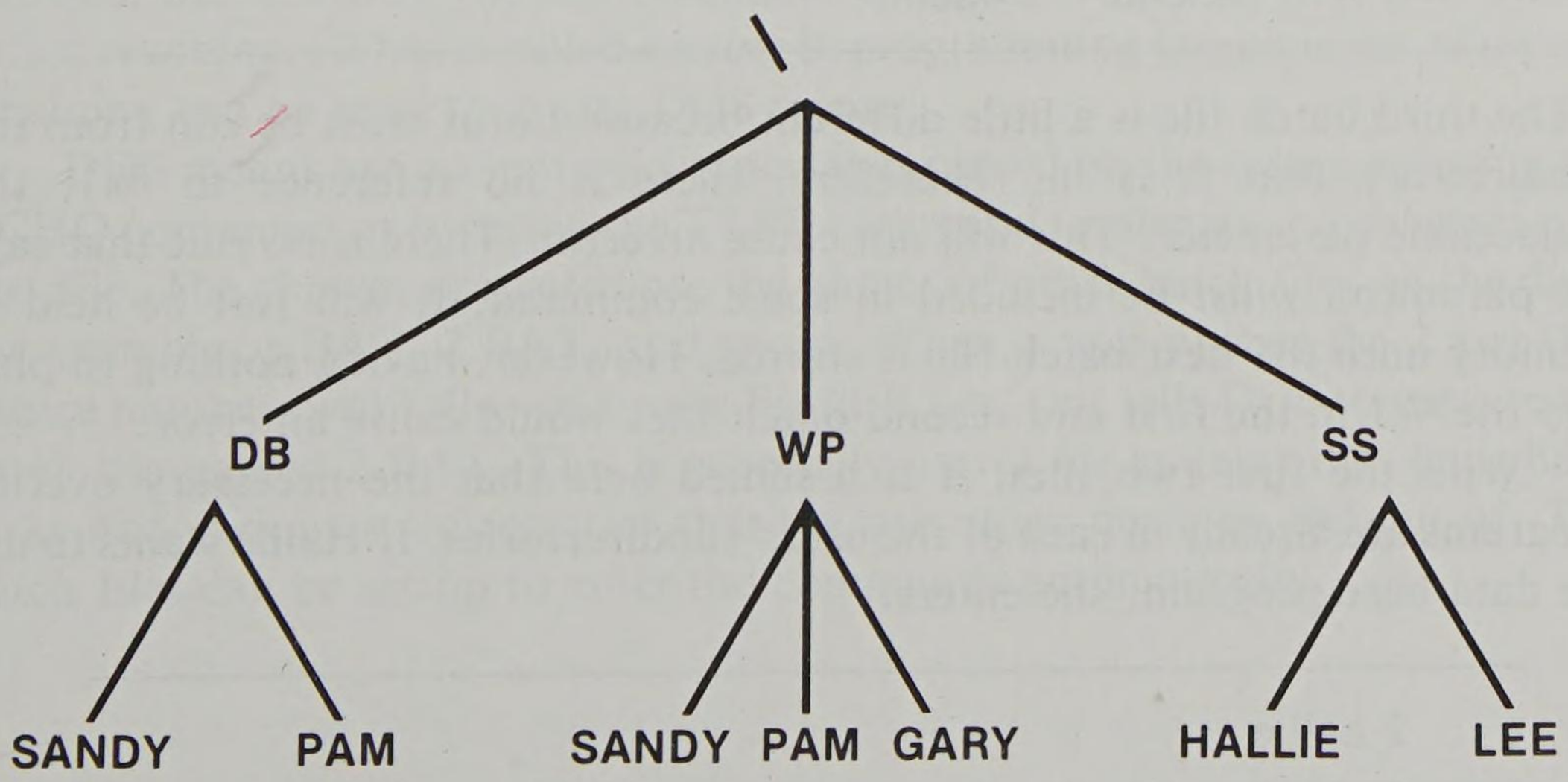


Fig. 10-1. A Tree Structure of All Users' Subdirectories

```
1 Word Processing
2 Data Base
3 Spreadsheet
4 Return to DOS
```

The MENU.BAT remains the same as in the previous example; it changes the prompt and TYPEs out the MENU.TXT file.

Filename: MENU.BAT

```
cls
prompt Enter your command now:$x____
type menu.txt
```

The user must enter one of the numbers, followed by his or her name. The name will then replace the %1 in the batch files that will be started by entering a 1, 2, 3, or 4. These batch files must contain the appropriate commands to start whatever word processing, data base, or spreadsheet program is being used. In the examples below, it is assumed that WordStar, dBASE II, and Lotus 1-2-3 are the programs being used.

Filename: 1.BAT

```
path \wp
cd \wp\%1
ws
cd \
menu
```

2.BAT

```
path \db
cd \db\%1
dbase
cd \
menu
```

3.BAT

```
cd \ss
lotus
cd \
menu
```

The third batch file is a little different because Lotus must be run from the subdirectory that it is in. Therefore, there is no reference to %1, the replaceable parameter. This will not cause an error. There is no rule that says the parameter *must* be included in some command. It will just be held in memory until the next batch file is started. However, having nothing to plug into the %1 in the first and second batch files would cause an error.

With the first two files, it is assumed here that the necessary overlay programs are already in each of the users' subdirectories. If Hallie wants to use the data base program, she enters:

```
2 hallie
```

It won't matter if she uses "hallie," "Hallie," or "HALLIE" because uppercase and lowercase are not distinguished in filenames or directory names. The last batch file, 4.BAT, again just changes the prompt back to a more normal one and clears off the screen:

Filename: 4.BAT

```
echo off
prompt $d   $t   Directory: $p$__$n$g
cls
echo Returning to DOS
```

Many batch files may end up at the root level. Careful organization of a disk, whether it be a hard disk or a floppy disk, can make it easier to keep track of where everything is.

10.5 Summary

DOS has a few special commands that are for use only in batch files. They are ECHO, REM, PAUSE, GOTO, IF, FOR, and SHIFT. ECHO is used to display lines of text on the screen. PAUSE causes the execution of the DOS commands in the file to halt until a key is struck. GOTO, IF, FOR, and SHIFT allow us to create mini-programs that control the flow of commands in a batch file. They can be used to cause the order of processing to jump around in the file instead of going from one line to the next as is normal.

The PROMPT command, an internal one, is used to change what is displayed on the DOS prompt line. The system date, system time, current subdirectory, or a message, for example, can be made a part of the system prompt.

A command in one batch file to start execution of another batch file is allowed, but it will be the last command executed from that first file. This is called *chaining*. (What is called *nesting* in programming jargon is not allowed.) Chaining can be used to create DOS menus.

DOS menus are no more than messages placed on the screen by using the ECHO command or by using the TYPE command to display the contents of a text file. The choices presented are the names of other batch files on the disk, for example, 1.BAT, 2.BAT, and so on. When a user pushes the 2 key (for choice number two) followed by the ENTER key, this tells DOS to execute the batch file named 2.BAT. This is especially useful for systems that have hard disks and various subdirectories that the user must move in and out of. The batch files can be set up to enter the commands automatically.

CHAPTER 11

Making Backups

Making backup disks is imperative. Files must be recorded on more than one disk! This is so important that the topic deserves its own chapter. Let's look at some of the reasons why making backups is so critical.

11.1 Why and When

All disk drives will eventually cease to function. They are mechanical devices with moving parts. They will not last forever. All disks will eventually wear out, too. The read/write head of a disk drive rubs on the disk when data is retrieved from or recorded onto a disk. Disks also fail for other reasons.

Is the message clear? It's just a matter of time before something fails. When it does, the backup disks have got to be there. In order for that to be the case, it is necessary to establish routines for making backup disks. Everyone using the computer must be aware of the procedures and the need to implement them.

Many new users have worked for periods of up to one year without ever making a backup disk. For example, in one office where the computer was used for word processing, the secretary did not know how to format disks nor how to make backup copies of the one disk that had been prepared for her. She worked with the same program disk in drive A and the same data disk in drive B for one year. Periodically she would erase old document files from the data disk to make room for new ones. This defeats one of the main purposes of having a computer, the ability to build a library of files that can be used over and over.

Another major reason for backups is that users occasionally make a mistake and erase the wrong file, files, or even the whole disk. Formatting a disk wipes it clean (this includes hard disks). It has happened to the wrong disk all too many times. Erasing the wrong file or set of files from a disk is very easy to do. It has happened at one time or another to almost everyone who uses a computer regularly.

So the wise person makes backup disks. Now how often must this be done? The answer is, "As often as needed to ensure that there is no irrecoverable loss of data." As a general rule, backing up disks should be done every day. It should be part of the daily routine, normally at the end of the day. It is quite possible that backup should take place even more frequently. For example, for someone writing a book, it is appropriate to make a backup copy when a chapter is completed. This may mean copying only the file that contains that chapter to a backup disk, rather than copying an entire disk.

To determine how often to back up, ask this question: "Can the data be reconstructed?' If it's a document being typed in with word processing, and the document has already been written down on paper, a normal backup at the end of the day sounds fine.

If the document is being composed at the keyboard, a pause half-way through the day (or even more often) to copy the file to another disk would certainly be in order, because if something were to happen to the disk being used to record the document while working, it would be impossible to reconstruct *exactly* what was written.

11.2 Backing Up from Floppy Disks: COPY vs. DISKCOPY

On a system with one or more floppy disk drives, the question always comes up, "Which is better to use to make backup copies, COPY or DISKCOPY?" The answer: An unequivocal "It depends." If only a couple of files have been created or modified, or if the files' names have something in common and can therefore be matched with a wildcard, then it is probably easier and quicker to use COPY. If one or more files are to be *added* to a backup disk that already has other files saved on it, then COPY must be used. DISKCOPY makes an exact copy of an entire disk. It wipes clean the target disk before any copying takes place. Only COPY can add something to a disk!

If a full disk must be reduplicated, then DISKCOPY will be faster. It copies faster because it does not go file by file as COPY does. Instead it takes into memory as much as possible from the source disk and then copies that entire large chunk to the target disk.

When using COPY, remember that it is an internal command. It will work any time that the DOS prompt is up. Typically, when work is in progress, there will be a main program disk and a second data disk to take the files. If the computer has only one disk drive, this may require some disk-swapping while working. If the computer has two drives, the main program disk will be in A and the data disk in B.

With two drives, when work has been completed it makes sense to keep the data disk in B and replace the main program disk with a backup disk in A. Then the appropriate files can be duplicated from B to A using COPY, or the entire disk in B can duplicated onto the disk in A with DISKCOPY. Keep in mind that DISKCOPY is an external program; so in order to use this last procedure, DISKCOPY.COM must be on the data disk; otherwise, a disk that already has DISKCOPY.COM will have to be inserted in one of the drives. As an alternative, all files can be copied using COPY, by entering *.* as the name of what is to be copied.

COPY can be used this way even if the computer has only one drive. Use the A and B designations to help keep track of which disk to put in as the necessary disk-swapping takes place. DOS allows reference to A and B drives even when there is actually only one drive. Messages to "insert the disk for drive B" or "insert the disk for drive A" appear as necessary.

Whatever the method chosen to do backups, it is a good idea to be consistent. Once you have determined the most effective way, it is best to use the same routine all the time. This way making backup copies will become an ingrained habit.

11.3 For Hard Disks: BACKUP and RESTORE

DISKCOPY is not employed when a hard disk is involved. The COPY command is often used, but so are two other commands that are made to work with hard disks. They are BACKUP and RESTORE. Both are external commands. Since they involve a hard disk, it is a simple enough matter to ensure that BACKUP.COM and RESTORE.COM are on that hard disk. Let's look at how these two work.

BACKUP is specifically designed to copy large numbers of files from a hard disk to one or more preformatted floppy disks. Unless a special option is used with BACKUP, the floppy disks being used to receive the files are wiped clean first. Disks being formatted to be used as backups should not have DOS put on them (i.e., the /S option should not be used). RESTORE is used to bring back files from a floppy disk that were placed there by the BACKUP program. RESTORE is the *only* correct way that such files can be brought back.

BACKUP is very similar to COPY, but there are three major differences:

1. The files copied to a floppy disk by BACKUP are not usable unless RESTOREd. They are listable on the floppy disk with the DIR command.
2. BACKUP has some extra options to it that allow some sophisticated selection of files to be included in the copying.
3. BACKUP is the *only* choice if a file takes more than 362K bytes of disk space. BACKUP has the capability of breaking up the file and copying it across one or more disks.

The general form of the BACKUP command is:

```
BACKUP WHAT DESTINATION-DRIVE/options
```

The first parameter, WHAT, means put in the filenames or subdirectory name and the drive where they are located. This specifies exactly what is to be copied. The destination is just the drive that contains the floppy disk.

All examples given in this chapter are for an IBM XT, where drive A is the floppy drive and drive C is the hard disk drive. Anyone working with different systems can just replace the A or C with the appropriate disk drive specification. Here is an example of the BACKUP command:

```
C>backup c:\lotus\*.wks a:
```

This command says, "Make a backup copy on drive A of all the files in the \LOTUS subdirectory that end in .WKS" (which happen to be Lotus 1-2-3 worksheets). In this case, we could replace the BACKUP with COPY, and it would accomplish the same thing. The only difference is we can only bring back BACKUP files with the RESTORE command. If the total space taken up by these files is not greater than 362K, then it might be just as well to use COPY.

To back up the entire hard disk, one of the options must be used. The command is:

```
BACKUP C:\ A:/S
```

The first parameter, C:\, tells BACKUP to start at the root level of drive C, the hard disk. The /S option says, "Start the process at the subdirectory specified in the command and work down into any subdirectories that branch from it, or from its subdirectories." In this case the starting point is the root level because it is explicitly included in the command (C:\), so ALL files in ALL subdirectories will be included in the backup. If the \ had not been included, then whatever subdirectory was the current one would have been the starting point, and all subdirectories that branch from it would be included. The A: says copy everything to the floppy disk in drive A. The following message appears on the screen:

```
Insert backup diskette 01 in drive A:
Warning! Diskette files will be erased
Strike any key when ready
```

When the process is started by hitting any key, the screen shows:

```
*** Backing up files to diskette 01 ***
```

The names of the files being copied are then listed while this takes place. When the first disk becomes full, the message changes:

```
Insert backup diskette 02 in drive A:
Warning! Diskette files will be erased
Strike any key when ready
```

The next disk is inserted, and hitting any key starts the process going again. Either before a disk is inserted or immediately after it is removed, it is imperative that it have a label with a number placed on it (avoid writing on labels already affixed to disks, but if this is necessary, use a felt-tip pen to avoid damage). If the hard disk has a lot on it, backing up everything can take awhile, in the range of 10 to 30 minutes.

It is not necessary to back up the entire hard disk very often, so let's look at ways to specify "less than everything." To back up everything in a particular subdirectory, all that is necessary is to give that subdirectory's name as the WHAT to copy. If it has subdirectories branching from it, they will not be included unless the /S option is tacked on. For example, if we have the tree structure shown in Figure 11-1, with subdirectories GL, AR, AP, and INVEN and various files in the ACCT subdirectory, we can get everything backed up, including all the subdirectories, with this command:

```
BACKUP C:\ACCT A:/S
```

This command will not include the WP subdirectory or any other subdirectories that branch from it. If the GL, AR, AP, or INVEN subdirectories have more subdirectories branching off from them, the extra subdirectories would also be included in the backup specified by this command. Everything that branches directly or indirectly from the \ACCT subdirectory is included. If the command is entered without the /S, then only files that are in the \ACCT subdirectory itself will be included. This is something that has confused many users, because the documentation regarding it is rather unclear.

Two more options exist for specifying what files are copied by the BACKUP command. They are /M and /D:date. The /M option says, "Include only files that have been modified since the last time the BACKUP program was used." For example:

```
BACKUP C:\ACCT A:/S/M
```

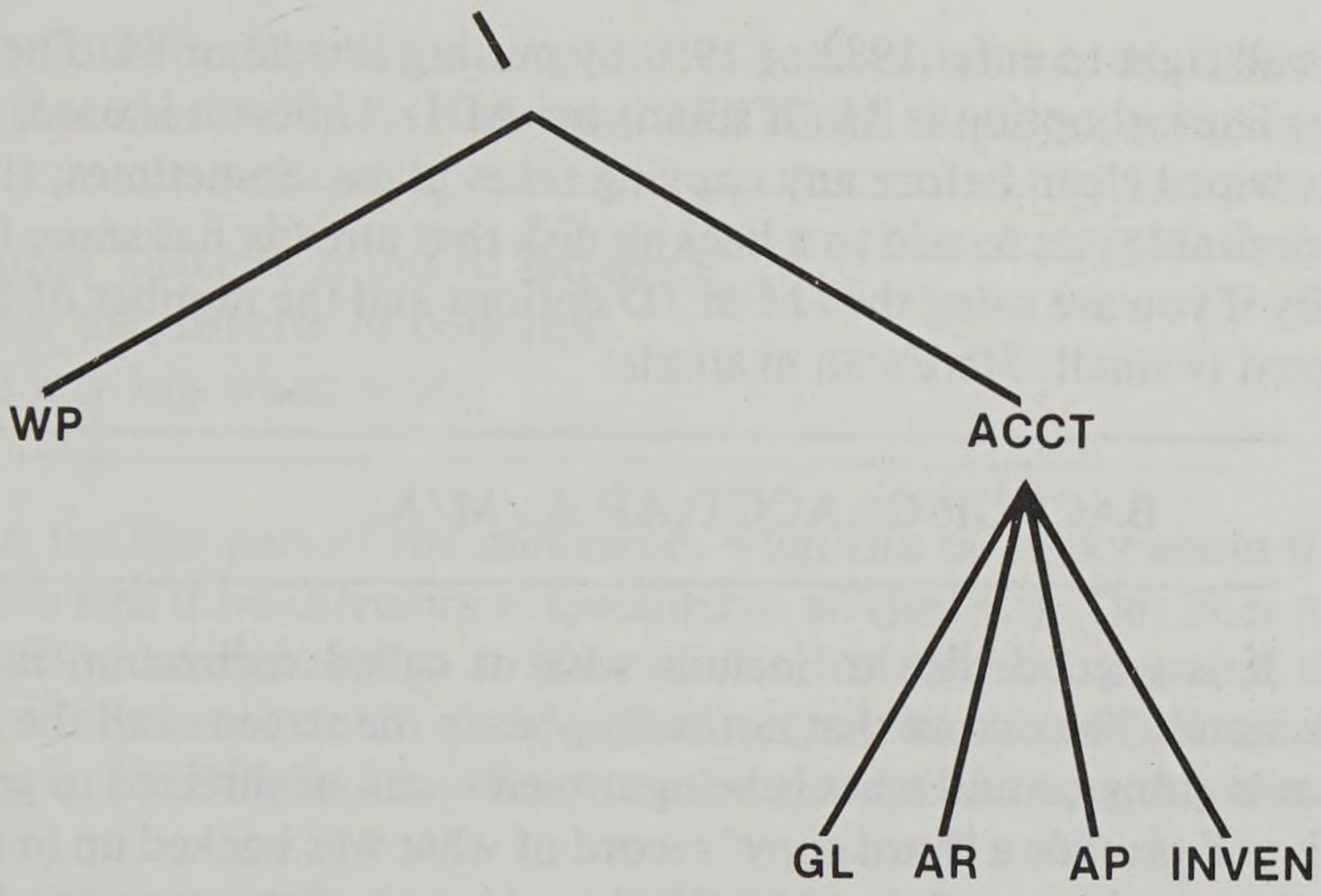


Fig. 11-1. A Tree Structure for Accounting Applications

If no files that are in any of the subdirectories included by this command have been modified, BACKUP displays this message:

```
Warning! No files were found to back up
```

The /D:date option says, "Include only files that have a date the same as or after the given date." For example:

```
BACKUP C:\ACCT A:/S/D:3-28-84
```

Notice that the D is followed by a colon (:). That's just part of the command's structure, or the syntax as it is called in computer jargon. Dates are entered just as they are for the DATE command:

```
mm-dd-yy      or      mm/dd/yy
```

One-digit numbers are accepted for the month (mm) or day (dd). It is not necessary to fill in zeroes. The following examples are all valid dates:

```
10-1-83   9-2-83   8-23-83
```

It is all right to enter 1983 or 1984 by putting just 83 or 84. The 19 is assumed.

The last option is /A. It stands for ADD. Unless it is used, the floppy disk gets wiped clean before any copying takes place. Sometimes, though, it might be desirable just to add to a backup disk that already has some files on it, especially if you are using the /M or /D options and the number of files that will be copied is small. Here's an example:

```
BACKUP C:\ACCT\AP A:/M/A
```

It is a good idea to include what is called redirection in the BACKUP command. The output that normally goes to the screen – all the messages about what is going on and what is being copied – can be directed to go to the printer. This will provide a "hard copy" record of what was backed up in that session. To do this, we just need to add >PRN to the end of the command each time.

```
BACKUP C:\ACCT A:/S>PRN
```

The concept of redirection is explained in Chapter 12.

Now that we have seen how to use BACKUP, we need to go on to the RESTORE command. RESTORE is used only if the original files are accidentally lost from the hard disk. The general form of this command is:

```
RESTORE FROM WHAT&WHERE/options
```

The FROM part will always be the drive specification of where the floppy disk is. For IBM XTs this is drive A. On systems that set up half the hard disk as A and the other half as B, then the floppy disk drive would be C. The WHAT&WHERE consists of drive-path-name. This part of the command serves two purposes. It not only says what is to be retrieved but also specifies where it is to be placed on the hard drive. Here's an example:

```
RESTORE A: C:\DB\GARY\*.DBF
```

The A: says, "Get the files from drive A." The rest of the command says to get all the files in the subdirectory named \DB\GARY that have an extension of .DBF. It also says to place them all in the same subdirectory on drive C. When the command is entered, the screen shows:

```
Insert backup diskette 01 in drive A
Strike any key when ready
```

If more than one floppy disk is involved, it is essential that the disks be inserted in the correct order. If they are not, RESTORE displays the message:

```
Warning! Diskette is out of sequence
Replace the diskette or continue
Strike any key when ready
```

In regard to the last part of the command, what can be tricky about it is where the files are sent if no directory is specified at all (just C:). This does not mean the files will automatically go to the root level of drive C. Instead, the *current* directory is assumed to be the destination if none is specified. If the current directory is C:\DB\GARY, then these two commands are functional equivalents:

```
RESTORE A: C:
RESTORE A: C:\DB\GARY
```

They both say to copy all the files that were originally in this subdirectory, and they both say to copy the files back into it. These commands as shown will include only what is in the C:\DB\GARY subdirectory. To include everything in the various subbranches of a directory, the /S option is used. For example, if the entire hard disk has been backed up, the command to bring back everything would be:

```
RESTORE A: C:\/S
```

This says, "Restore from A all files in the root directory and all dependent subdirectories and send them to C: with the same tree structure."

The only other option for RESTORE is /P, which stands for Prompt. When this is included, the RESTORE program stops and asks if it should go ahead each time that it sees that the file about to be copied still exists on the hard disk *and* has been modified since it was backed up. It "prompts" the user to answer yes or no as to whether or not this file should be restored. If the answer is no, the file will be skipped. This prevents overwriting a new version with an older version.

Now that we've seen how BACKUP and RESTORE work, let's go on to a discussion of when and how to do backups.

11.4 Backing Up from a Hard Disk: BACKUP vs. COPY

Just as there is a decision to be made between COPY and DISKCOPY when backing up floppy disks, a decision has to be made between COPY and

BACKUP when working with a hard disk. Everyday users will probably find it more convenient and simpler to use COPY. That way, just the files that were worked on that day can be either added to or updated on the backup disk(s). Updating means copying a file to a disk that already has an old version of that file on it. Doing this overwrites the old one.

When there are multiple users of the computer, each user should have a separate backup disk and be responsible for doing his or her own copying each day. This can be made easy with the use of batch files, discussed in the next section. Then if a particular file is lost, a user can simply copy it back from his or her own backup disk.

This is not enough of a backup system, though. It is best to supplement this with use of the BACKUP program. If there are several users involved, one should be appointed "system operator." This would normally be the person who is either the most knowledgeable about or is the heaviest user of the system.

Once every month or two, this person should back up the entire hard disk. If the daily usage of the computer is fairly heavy, this backup should probably be monthly. If it's sporadic, every two months is likely to be adequate. The command to use to do this backup on an IBM XT is:

```
BACKUP C:\ A:/S
```

For other systems with different drive designations for the hard and floppy disks, just replace C: with the specification for the hard drive and A: with the name of the floppy drive.

As more and more files accumulate on the hard disk, it will take more floppy disks to hold all of the hard disk's contents. The maximum number required for a 10-megabyte hard disk is 29, assuming the floppy disk drive is double sided (and thus each disk will hold 362K). Remember, this also assumes that the disks were formatted *without* using the /S option. All these disks must be formatted before BACKUP is started.

The number of disks that will actually be needed is easy enough to compute before starting BACKUP. The CHKDSK program can show how many bytes are still available on the hard disk. This number is subtracted from the total bytes available (reported on the first line of the CHKDSK output). The result is then divided by 362,000. To make it even easier, the last three digits of all three of these numbers can simply be dropped off. The number obtained in this way should be rounded up. For example, if the number is 13.5, then 14 disks will be needed.

Besides this monthly task, a less complete BACKUP should be done at least once a week. For this, a separate set of disks will be needed. In fact, two more sets of disks will be needed. Now the only question left is, what option or combination of options works best for copying only certain files?

Well, we certainly don't need to copy all the files that never change — all those application programs for word processing, spreadsheet, and so on. A

first reaction might be to think of /M. Let's delve into this possibility. We have two sets of disks. We use the following command, and set 1 of the disks (the printer must be on when the redirection >PRN is used):

```
BACKUP C:\ A:/S/M>PRN
```

This only takes the files that have changed since that last backup which, we'll say, was when we backed up the entire hard disk. We watch the list and note that only files X, Y, and Z were copied. The next week, we do the same, using set 2 of the disks. We note that files X, A, B, and C are copied. These were the only ones that had any changes made to them since we did BACKUP the week before. The following week we do it again, using set 1of the disks again, which wipes out what was on them (X, Y, and Z). We note that we got X, A, B, and C again.

Now, we have a problem. We no longer have Y and Z. So we think, "Maybe we need to use the /A option too." This causes files to be added to the disks each time. Not only is this unwieldy, but it will cause more than one version of the same file to be copied onto the disks (they will be on the disks in the order in which they were added to it). Which version will we get when trying to RESTORE? There are too many problems with this whole approach! So we don't want to use /M and /A.

That leaves us with /D. We can in fact come up with a good system using this option. Let's start again. We do the monthly backup. The date is 4-1-84. A week later, it is time for a partial backup. Using set 1 of our two extra sets of disks, we use this command:

```
BACKUP C:\ A:/S/D:4-1-84>PRN
```

This catches all files changed since the last complete backup. The next week, using set 2 of the disks, we use exactly the *same command!* This again catches all files that were changed since the last complete backup. Now set 2 has all the files that set 1 has, but it also has everything that was changed after set 1 was created.

The next week, the same command is used again, and set 1 of the disks is reused. The /A option is not included. This wipes set 1 clean of the old stuff and copies all files changed since that last full backup. This goes on, with the two sets of disks being alternated, until it is time to do a complete backup again, using that other (third) set of disks again.

Why two sets of disks for the partial backups? It's just an extra precaution, in case something happens to one of the floppy disks. Really cautious system operators will have two full sets of floppies for the complete backups and will alternate those, too; otherwise, how could everything be retrieved if something happened to the only set of complete disks? For example, recall that the

RESTORE program runs into problems if the disks get out of sequence when restoring is in progress.

One other option that can be used is to turn VERIFY on. This will make the process even slower, but it will be even more reliable. Copying, whether with COPY or BACKUP, is quite reliable with DOS 2.0 and 2.1, so using VERIFY is not critical.

Should anything happen to the hard disk, the procedure to RESTORE depends on whether the entire disk must be restored or just a portion. To put back the entire contents of the hard disk, the command is:

```
RESTORE A: C:\/S
```

Do this first with the last set of monthly backup disks that has everything. Then the same command is used again with the last set of disks that was used with the DATE option. This gets all the files up to date, as files that changed since the complete backup are copied onto the hard disk, overriding the older versions.

If only a few files must be replaced, the user's own backup disk can probably be used to COPY back just those files needed. But if something like an entire subdirectory were lost, then the system operator's disks would be used. It is not far-fetched to have to do something like this! Since a subdirectory can be removed only if it is empty, there are times when the ERASE *.* is used to clear out an entire subdirectory, so it can be taken off (maybe all the files have been consolidated into another subdirectory). But occasionally people do this to the wrong subdirectory (it has happened!). To restore the entire subdirectory named \DB\GARY, the command is:

```
RESTORE A: C:\DB\GARY
```

To restore an entire subdirectory, it is most likely that the system operator's set of disks that has a complete backup will have to be used. Then the same command can be given again with the latest set of disks that used the DATE option, in case any of the files in the subdirectory had been changed since the complete backup. The files on the dated backup disks will be searched through sequentially, starting at 01, even though it is possible that no files will match the criteria specified. As always, care must be taken to insert the disks in the correct sequence.

11.5 Using Batch Files

Using batch files can make the backup process much easier for everyday users. Let's look first at how batch files might be set up for a system using one or two floppy disk drives. Even if there is only one drive, references to A and B

drives are acceptable, even desirable. It helps to differentiate what is going where.

A batch file set up to take care of backing up must prompt the user through whatever steps are necessary, steps such as taking one disk out and putting in the backup disk. The batch file is normally saved on the data disk, since that is the disk that the procedure will start from, and because the main program disk will have to be removed from drive A so the backup disk can be put there.

Where the user ends up when finished with the program depends on which program is being used. With something like dBASE II, it is normal to have the dBASE programming in drive A and a data disk in B. When this is the case, dBASE must be started with drive A as the current drive; otherwise a message comes up saying "OVERLAY FILE (DBASEOVR.COM) CANNOT BE OPENED" because it searches for that overlay file on the current drive. When dBASE is finished, it then exits to drive A. To start a batch file on drive B, the user can either switch over to drive B and then type in the name, or just precede the name with B:. With a program such as WordStar, for example, it is normal to end up on drive B. In that case, nothing special needs to be done. The batch file can be executed immediately.

A batch file to back up dBASE work might look look this:

Filename: DOBACKUP.BAT

```
echo off
cls
echo Remove the dBASE disk from drive A.
echo Replace it with the backup disk in drive A.
pause
copy *.* a:
echo Backup copying is done. The disks can
echo be removed now.
```

Actually, this very same batch file could be used with almost any data disk that needs to be backed up. It would simply be a matter of replacing the message "Remove the dBASE disk" with "Remove the disk from drive A."

To make it even easier on the user, if the main program disk has an AUTOEXEC.BAT to autostart the program, it can "chain" to this batch file. Here's an example of what the AUTOEXEC.BAT file might look like on a disk with dBASE.

Filename: AUTOEXEC.BAT

```
echo off
cls
date
time
```

```
dbase
b:
dobackup
```

The command "dbase" runs the dBASE program. When the user "exits" from dBASE, the next command in the file is executed. The "b:" shifts the user over to drive B, and "dobackup" executes the batch file. It's really fairly straightforward when using floppy disks. Things get complicated with a hard disk, especially if there are numerous users involved. Let's look into this.

For the person designated as the system operator, it is just as simple to do the monthly and weekly backups just by typing the appropriate BACKUP command, rather than fiddling with some batch file. For the everyday users, though, batch files can be used to ensure that they do their copying.

Let's assume that we have implemented a system similar to that described in Chapter 10. The tree structure for this disk is the one shown in Figure 11-2.

Upon startup, everyone sees this menu:

```
INSTRUCTIONS

Enter the number you want, followed by a space
and then your first name.

1 Word Processing
2 Data Base
3 Spreadsheet
4 Return to DOS
```

There is a batch file named MENU.BAT that contains the command (TYPE) that causes this text to be displayed on the screen. To do word processing, Gary types in the command:

```
1 gary
```

In Chapter 10, we said the batch files that must be available would look like these:

Filename: *1.BAT*	*2.BAT*	*3.BAT*
path \wp	path \db	cd \ss
cd \wp\%1	cd \db\%1	lotus
ws	dbase	cd \
cd \	cd \	menu
menu	menu	

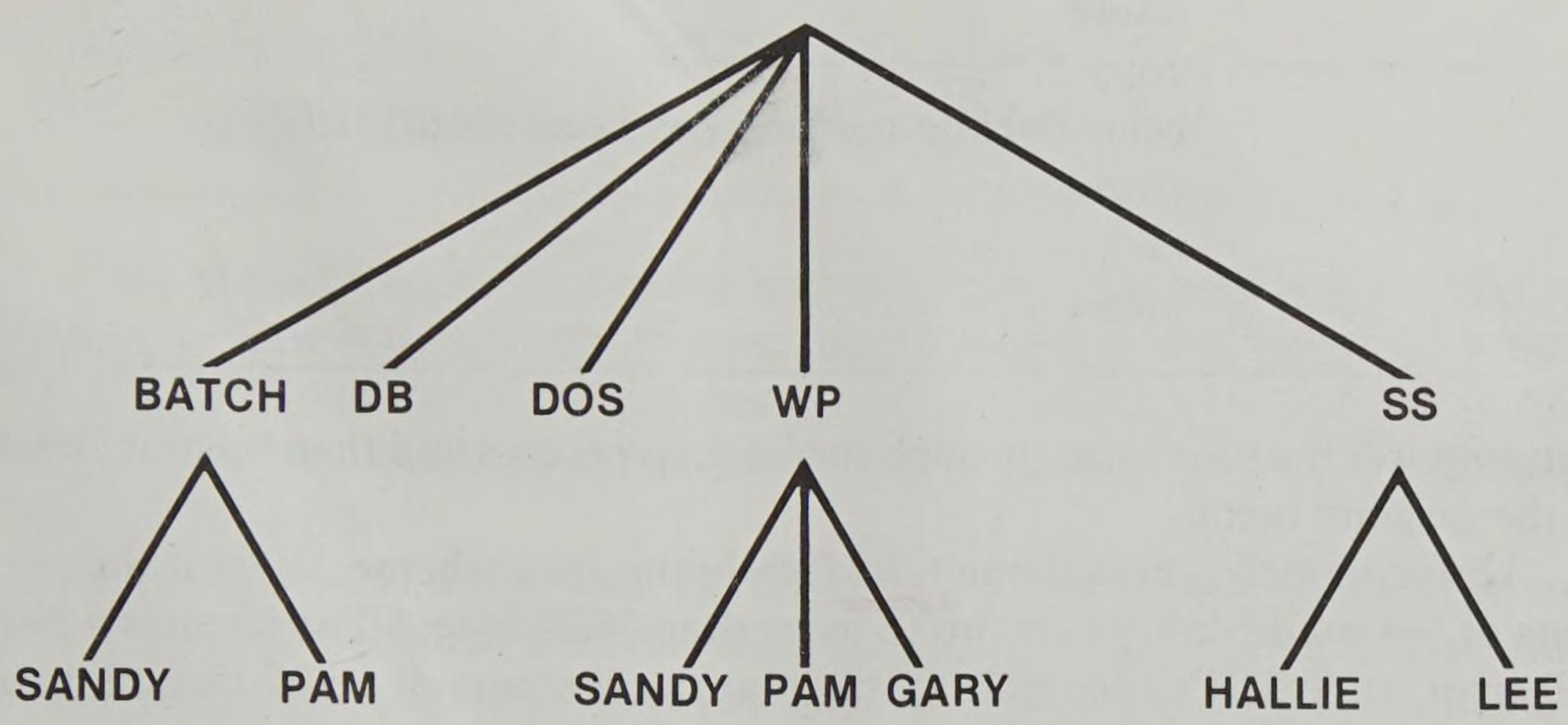


Fig. 11-2. Sample Tree Structure of a Hard-Disk System

Now to incorporate backing up, we'll change these. Rather than have them go straight back to the root level and restart the MENU batch file there, we'll have each one chain to a DOBACKUP batch file that is in each of the subdirectories. Since the subdirectory will be the "current directory," it is easy enough to write the COPY command to back up the files. The startup batch files now look like this:

Filename:	*1.BAT*	*2.BAT*	*3.BAT*
	path \wp	path \db	cd \ss
	cd \wp\%1	cd \db\%1	lotus
	ws	dbase	dobackup
	dobackup	dobackup	

The DOBACKUP batch files that must exist in each and every subdirectory look something like this (as always in this chapter, the commands here assume that an IBM XT is being used, where C is the hard disk and A the floppy disk drive):

Filename: DOBACKUP.BAT

```
echo off
cls
echo Place the backup disk in drive A.
```

```
pause
copy *.* a:
echo Backup copying has been completed.
pause
cd \
menu
```

This requires the user to go through the backup process and then "chains" back to the original menu.

Unfortunately, there is one *fatal* flaw in this nice scheme. What if the contents of the subdirectory total up to more than 362K bytes? The result is a disk full error. It could happen quite easily that the contents of an entire subdirectory grows to more than 362K. Even one single file can grow to be more than 362K.

One alternative is to have the users do the COPYing themselves. This really is not too difficult to implement. The COPY command itself is one of the easiest to use. Where problems crop up with unsophisticated users is in maneuvering in and out of subdirectories or dealing with complicated paths as part of file or subdirectory names. If these aspects of the copying can be mastered through one or two classes for everyone, this may be the easiest way to go.

A second alternative is to set up the batch files to use BACKUP instead of COPY. In this case, the users need to know only enough to be prepared with several disks. Messages in the batch file can warn them of what is about to happen and tell them how to respond.

A third alternative would be to try to set up a very complicated system with a batch file that tells the user to type in a command (the name of the next batch file that will actually do the work), followed by the name of each of the files that was worked on. The second batch file could then use SHIFT, IF, and GOTO commands to copy just those files specified. It would contain the following sequence:

Filename: DOBACKUP.BAT

```
:start
copy %1 a:
shift
if not exist %1 goto stop
goto start
:stop
echo Copying done
pause
```

The user would initiate this batch file (it would have to be in each subdirectory) with a command such as:

```
dobackup ltr2 ltr3 memo5.84 memo6.84
```

This seems to be an interesting way to do it at first glance, but it has *two* fatal flaws. One is the same as before; the files being copied may exceed 362K. The second is that the users may not know the correct names of all the files, and they may not know the names of all the files that have been created during their work session.

For example, if a user uses dBASE II and enters more records, then uses the command to reindex the records, then creates a new report format, three files will have been updated, and the user will have only typed in the name of one of them, the report format. Furthermore, dBASE automatically puts an extension on each of the files. The data file gets .DBF, the index file gets .NDX, and the report format file gets .FRM. Only somewhat knowledgeable users know this kind of information. Similarly, in Lotus 1-2-3, worksheets are given the extension of .WKS, graphs that are saved separately from the worksheets are given .PIC, and worksheets printed to disk are given a .PRN extension. Again, a user has to be fairly sophisticated to be aware of this.

One main reason for trying to create sophisticated batch files is so that the casual users will go through the backup process. In the batch file, use COPY and the wildcards or a subdirectory name if possible, and if it's not possible, then use BACKUP. The choice is based on whether there is any possibility that what is to be copied will total more than 362K. If so, BACKUP should be used, and the casual users can be instructed to get help if they actually have to RESTORE anything.

11.6 How to Avoid Losing Data

So far this book has discussed various things to do to avoid data loss but it is probably worthwhile to have one unified discussion on the topic. First, we should identify the most common occurrences that cause data loss. They are listed here, starting with the most frequent cause.

Causes of Data Loss

1. Erasing the wrong file or files. This can be caused by typing in the wrong name, using a wildcard without checking the ramifications, using *.* in the wrong subdirectory, and other such errors.
2. Disk failure. The disk goes bad. It may have had an inherent flaw from the start that quality control did not pick up.
3. Copying in the wrong direction. Instead of going from A to B, the copying is done from B to A (or vice versa), causing an older version of a file to wipe out the more recent one.

4. Formatting the wrong disk. This wipes out the contents of the disk.
5. Issuing the wrong command while using a particular program. The command happens to cause data to be erased.
6. Computer or power failure. This can cause loss of data that had not yet been saved to disk.

Now let's look at specific solutions to these potential problems. Basically, we can try to prevent them, and we can also try to be prepared to counter the consequences — to offset the results.

Ways to Avoid Data Loss

1. Be careful and pay attention. Pause for a moment and reflect on what is about to happen when the potential of erasing data exists. Don't go through an erasing procedure mindlessly at 11:45 p.m. Many of us know that feeling all too well. It is surprising how fast it wakes you up when you realize you just destroyed your entire day's work! Use the DIR command before the ERASE/DEL command to get a "preview" list of what will actually be erased.
2. Have backups. Have more than one set of backups of the same files.
3. Use batch files. This avoids copying "in the wrong direction," caused by typing in a command backward. It also makes backing up easier, and hence there is more likelihood that backing up will be done.
4. To avoid loss from the computer's memory, before the data gets recorded to disk, do frequent saves. It's hard at first to get into this habit, because it is so easy to get totally wrapped up in the work being done. It takes a conscious effort to remember to save every page in word processing, or every ten or fifteen minutes of work with another program. After one or two disasters it's amazing how we start really paying attention to this!

11.7 Organizing a Hard Disk

At this point in the book, after chapters on how to create subdirectories, how to move around in them, how to create batch files to do a lot of the moving, starting programs from here and there, and how to back up everything, it is appropriate to talk about how to organize a hard disk.

With floppy disks, this is not really an issue. Most of the time it is not necessary to create a tree structure on a floppy disk. All the files can simply go into the root directory. But with hard disks, it is really important to segment them into subdirectories. At a minimum, there should be a subdirectory for the files of every major application being performed on the computer. That means a subdirectory directly off the root directory for word processing, one for data base, one for electronic spreadsheet, one for accounting, and so on.

In fact, one good way to organize the hard disk is to have nothing but subdirectories at the root level. Unfortunately, what happens most of the time is that people start off right away copying everything from the DOS disks to the hard-disk root directory, then as they add a program to their library they copy the program there too, and slowly but surely things get out of hand. It can be very difficult to reorganize once lots of data files have been added to the root directory as well. Files have to be copied one by one to new subdirectories.

The best way to avoid this is to start from the very beginning with the needed subdirectories. All the files from the original DOS disk can go into a subdirectory. All the batch files except the AUTOEXEC.BAT can go into another. Normally a few public domain programs and other miscellaneous materials turn up over a period of time, so a subdirectory called MISC might be handy. Before all the files that make up an applications program like WordStar are put onto the hard disk, a subdirectory should be created for them. Then after changing to that subdirectory, making it the current one, it is very easy to copy all the files from the newly acquired disk or disks into it, using the COPY and *.*.

Figure 11-3 shows a tree structure for a hard disk on a computer being used for four major applications. WordStar is the word processing program chosen, and its name is usually abbreviated WS. dBASE II is the data base program, Lotus 1-2-3 the spreadsheet program, and the subdirectory named ACCT is for the accounting programs.

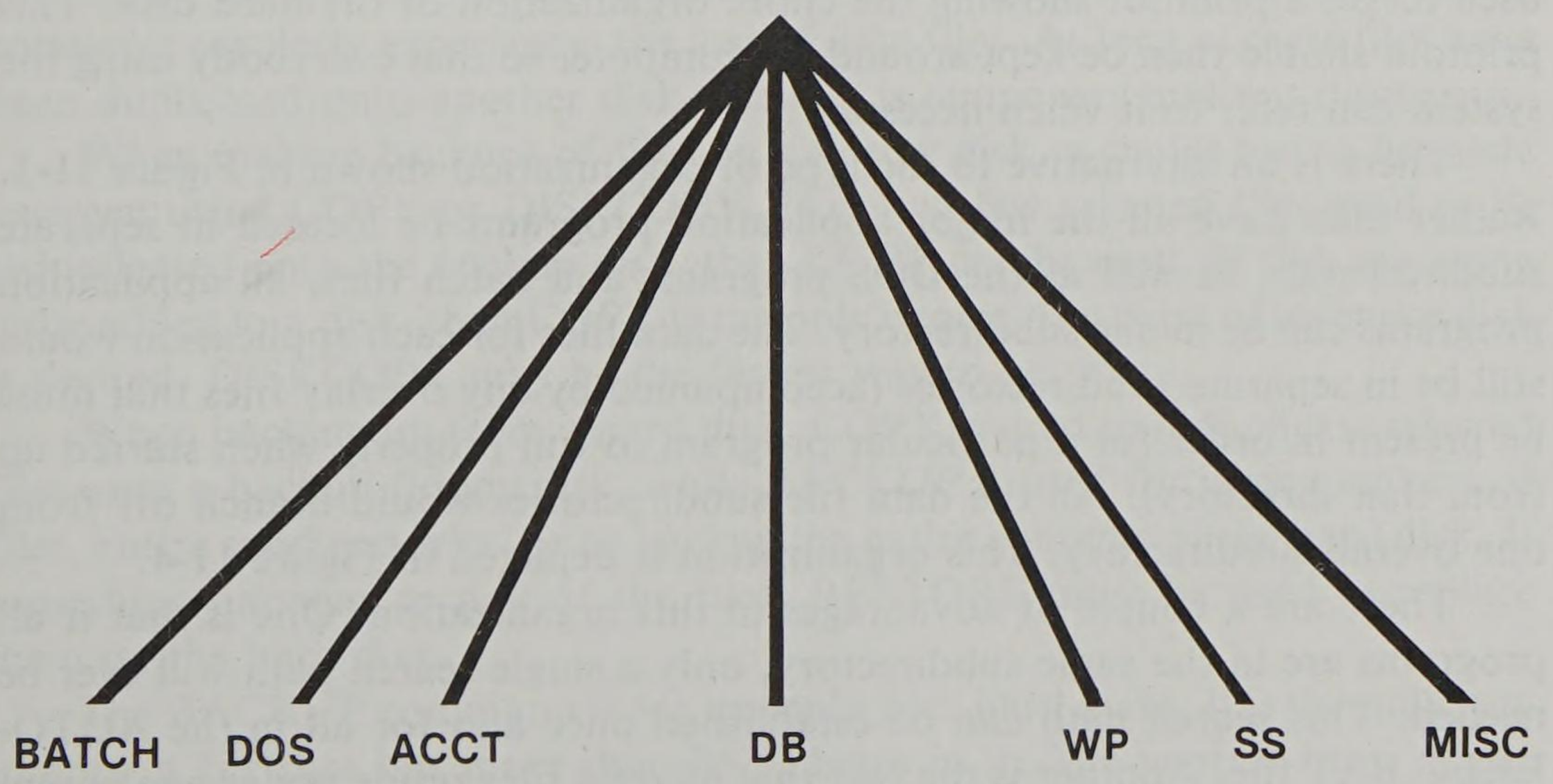


Fig. 11-3. A Tree Structure with Branches for Each Major Application

Whether or not any subdirectories will be needed underneath each of these depends on the number of files that will be created, the number of different people who will use the computer, and whether the programs

themselves have overlays that complicate the matter of keeping everything together. If everything is in some subdirectory, it will be very important to keep track of the search path or paths that are established so the batch files or DOS programs can be found when needed. The AUTOEXEC.BAT file would need to have in it a path command like this:

```
path \batch;\dos
```

The \DOS subdirectory is included because such external programs as CHKDSK and TREE may be needed from time to time. Every time executing a new batch file changes these search path specifications, it will be necessary to change them back before that batch file ends. An alternative would be to keep one main batch file at the root level, besides the AUTOEXEC.BAT file, that always sets things up properly. This could be the MENU.BAT file that underling batch files always come back to.

While it is quite necessary to have some subdirectories on a hard disk, the tree structure should be kept as simple as possible. The more branches in the tree, the more complicated it becomes to keep track of where everything is. It might be best to try to have most of the subdirectories branch off the root directory. That way they will show up when a simple DIR command is used. Subdirectories that branch off other subdirectories will not show up when the DIR command is used. Whenever the structure of the hard disk is changed, either by adding or taking away a subdirectory, the TREE command should be used to get a printout showing the entire organization of the hard disk. This printout should then be kept around the computer so that everybody using the system can refer to it when necessary.

There is an alternative to the type of organization shown in Figure 11-3. Rather than have all the major application programs be located in separate subdirectories, as well as the DOS programs and batch files, all application programs can be in one subdirectory. The data files for each application would still be in separate subdirectories (accompanied by any overlay files that must be present in order for a particular program to run properly when started up from that directory). All the data file subdirectories would branch off from one overall subdirectory. This organization is depicted in Figure 11-4.

There are a couple of advantages to this organization. One is that if all programs are in the same subdirectory, only a single search path will ever be needed. This search path can be established once and for all in the AUTOEXEC.BAT file. Another is the fact that all data files reside under one parent subdirectory, the one named FILES in Figure 11-4. This can be used in the BACKUP command, along with the /S option, for the weekly backups that work off the DATE of the last monthly (complete) backup, cutting down quite a bit on what has to be searched through by the BACKUP program. A disadvantage of this organization is that the one subdirectory with all the program files in it will be a real zoo.

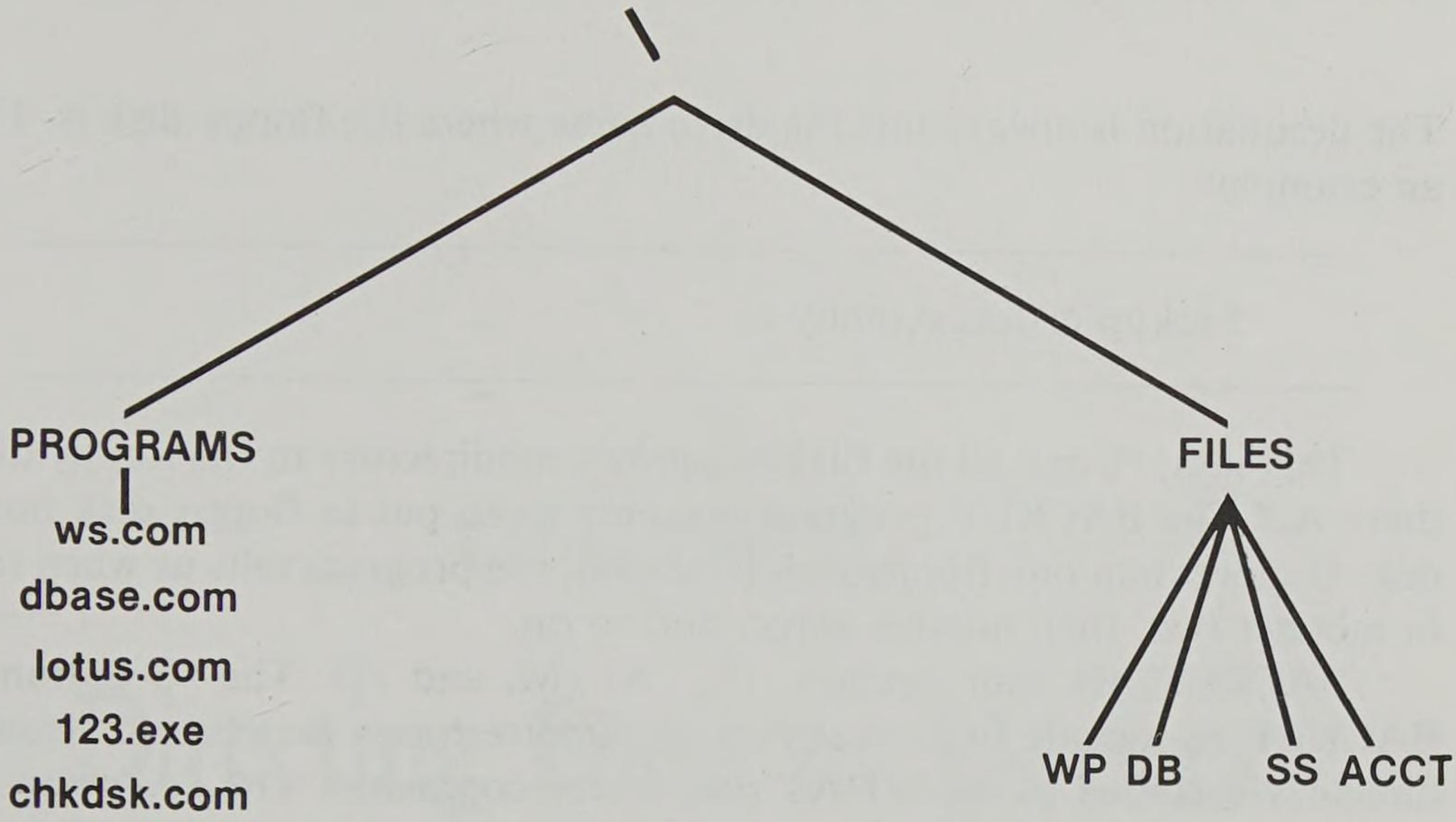


Fig. 11-4. Organizing a Hard Disk to Optimize for BACKUP

11.8 Summary

Making backup disks is an essential function that *must* become part of the routine of using a computer. At some time or another, everyone who uses a computer regularly experiences the loss of data files. As long as these files have been duplicated onto another disk, the loss is temporary and not disastrous.

When making backups of files on a floppy disk, a choice has to be made between using COPY or DISKCOPY. If only a few selected files need to be reduplicated onto the backup disk, then COPY can be used. If files are going to be added to a disk, then COPY is the only choice. If a copy of an entire disk is desired, DISKCOPY may be the fastest way to get it.

When backing up from a hard disk, COPY is used to reduplicate selected files onto a backup floppy disk, while BACKUP is used for large numbers of files, entire subdirectories, or to backup the entire contents of the hard disk. If something happens to any of the files, RESTORE must be used to replace them on the hard disk.

The BACKUP command is for use only with hard disks. It is the only way to copy a file that has more than 362K bytes in it. The general form of the command is:

BACKUP WHAT DESTINATION-DRIVE/options

The WHAT part of the command specifies exactly what is to be copied.

The destination is always just the drive name where the floppy disk is. Here's an example:

```
backup c:\lotus\sandy a:
```

This says, "Copy all the files in Sandy's subdirectory to the floppy disk in drive A." The BACKUP program prompts us to put in floppy disk number one. If more than one floppy disk is needed, the program tells us when to put in number two, then number three, and so on.

BACKUP has four options, /S, /A, /M, and /D. The /S option tells BACKUP to include in the copying all subdirectories that branch from the directory specified in the WHAT part of the command. The /A is for Add. This causes the BACKUP program to add what is being copied to disks that already have files on them. Otherwise, disks used are wiped clean first.

The /M stands for Modified, and tells BACKUP to copy only files that have been changed since the last time BACKUP was used. The /D option includes a date with it when used. It tells the BACKUP program to copy only files with a date equal to or later than the one specified. Here's an example:

```
backup c:\ a:/s/d:4-1-84
```

Since the /S is used, and the copying is to start at the root level of drive C (this command would be appropriate for an IBM XT), this command says to include all subdirectories on the disk. Each subdirectory on the disk will be checked, but only files with a date of 4-1-84 or later will be copied. Note the colon after the /D but before the date. It must be there.

Batch files should be used to make the backing up process easier, which will help ensure that it gets done. Messages prompting the user to put in the backup disk can be included, as well as the actual commands to initiate the copying.

Having backup copies of everything is absolutely imperative. Unfortunately, it usually takes one major disaster to drive this point home.

CHAPTER 12

Directing Data to Devices

12.1 Device Names

Before we can discuss sending output to different devices, we have to have an understanding of what devices are, and of the meaning of the terms standard input and standard output device.

A **device** is a peripheral, an extra piece of equipment that is attached to the computer. This is different from an interface, which is the physical connection for a device – for example, a plug or pin connector and its wire.

The types of peripherals that are of primary interest to us are printers that communicate in parallel with the computer, printers that communicate serially, modems (which are serial devices), and terminals, which are keyboard and screen devices that communicate serially.

The terms for the two different ways to communicate – in parallel or serially – refer to how the bits in a byte travel from one device to another. With **parallel** communications, all eight bits of a byte go out side by side on eight separate wires. They go out "parallel" to each other, as depicted informally in Figure 12-1. With **serial** communications, there is only one wire, and the eight bits go out one after the other, or in a "serial" line. This is depicted in Figure 12-2. In serial communications there are normally some extra bits thrown in to help keep track of things.

So bytes (which are equivalent to characters) of data may come in to a computer via a parallel connection or a serial one. This necessitates having two different kinds of interface spots. The hookup points are referred to as **ports**, like a port where a ship can dock.

Input normally comes from the keyboard, so in DOS the keyboard has been defined as the "standard input device." Unless DOS is told otherwise, it

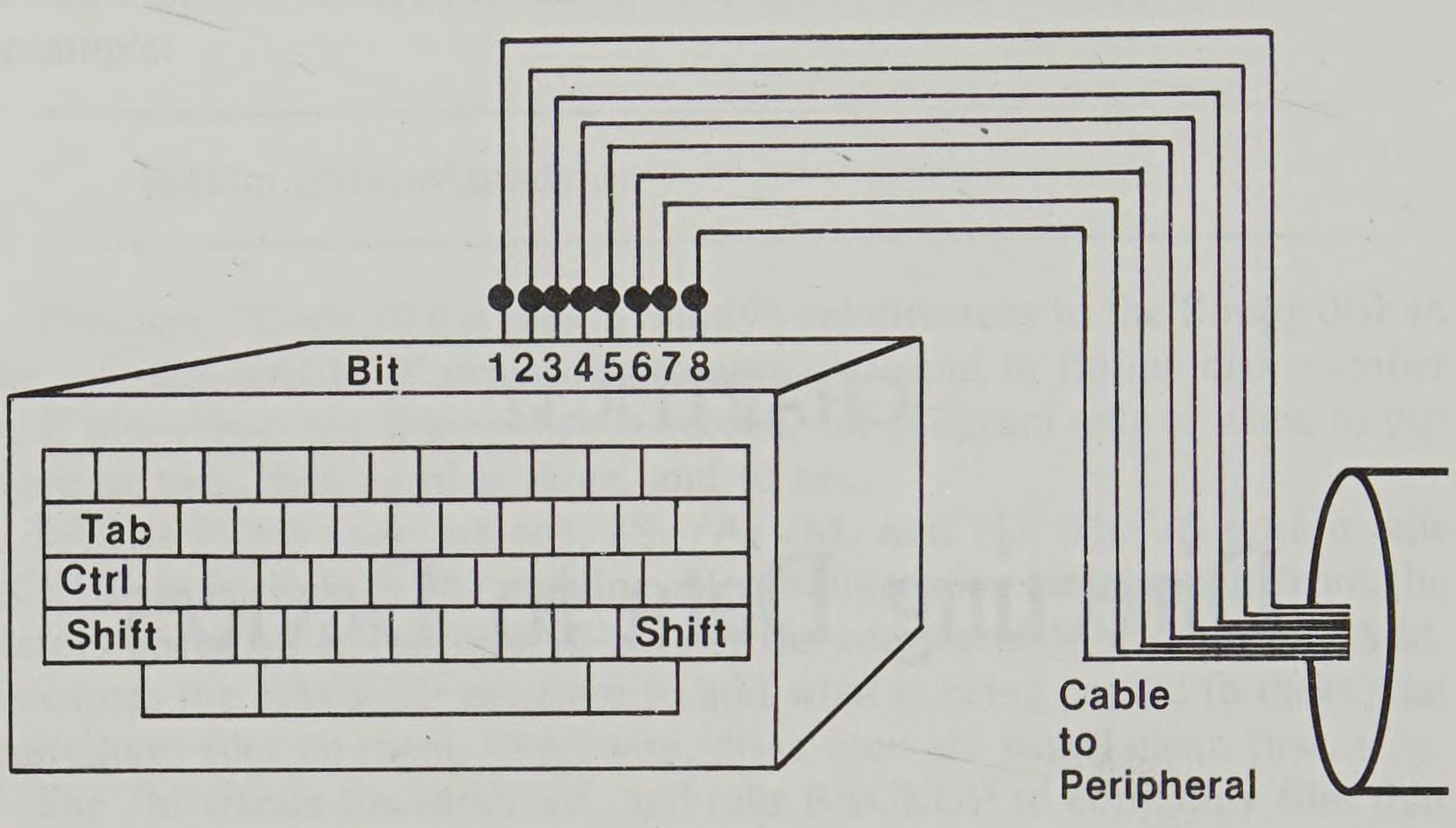


Fig. 12-1. Illustration of a Parallel Interface

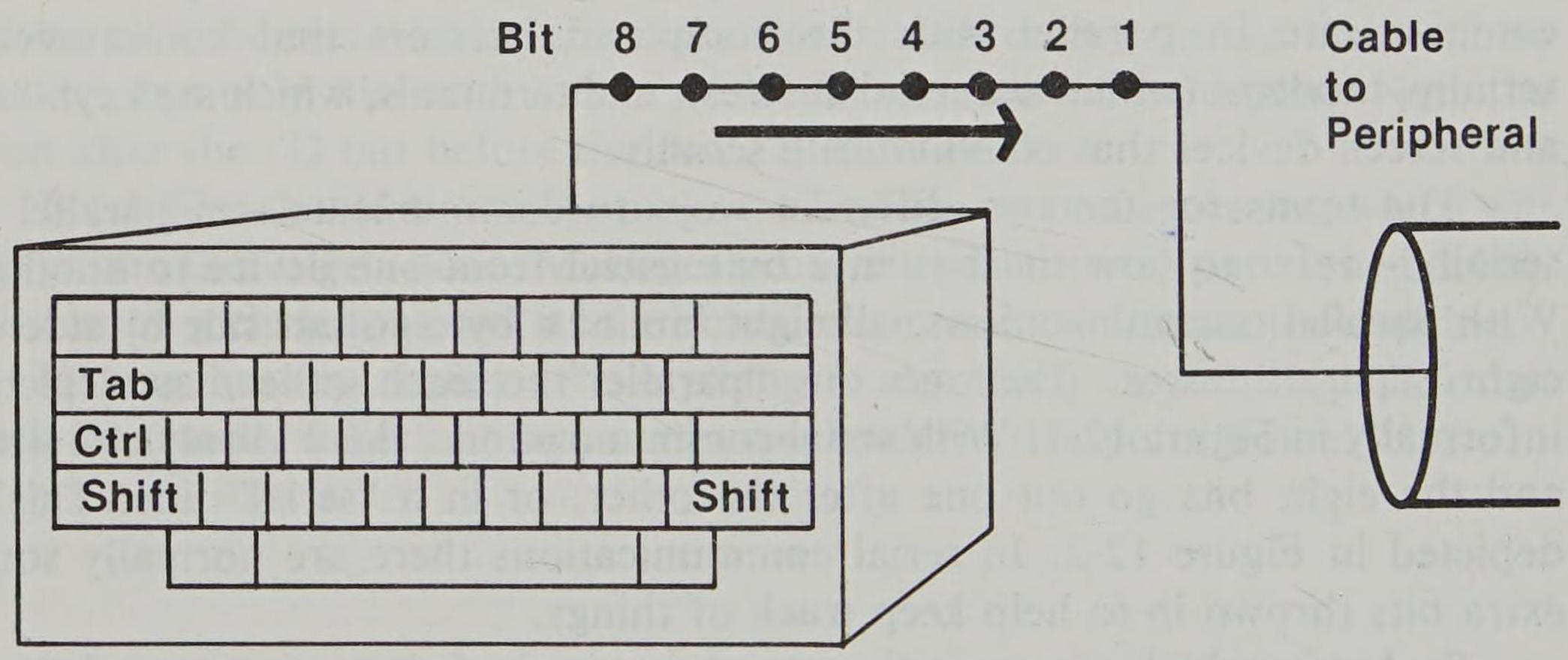


Fig. 12-1. Illustration of a Serial Interface

expects to get input from the keyboard. As for the output, the video screen is defined as the "standard output device."

Before we can tell DOS to send output to devices other than the screen, we have to know their names so that we will be able to refer to them in commands. Table 12-1 contains a list of device names.

Table 12-1. Valid device names with DOS 2.0 and 2.1

ACTUAL DEVICE	*NAME*	*WHAT IT STANDS FOR*
Keyboard & Screen	CON:	Console
Parallel Port 1	LPT1:	Line Printer 1
	PRN	Printer
Parallel Port 2	LPT2:	Line Printer 2
Parallel Port 3	LPT3:	Line Printer 3
Serial Port 1	COM1:	Communications Port 1
	AUX:	Auxiliary Port
Serial Port 2	COM2:	Communications Port 2
Nonexistent device	NULL:	Null device
Cassette Recorder	CAS1:	Cassette recorder 1

Often the colon on these names can be left off. It must be included if the device name is followed immediately by some other character, unless the character is a space. Anytime it makes sense, these device names can be substituted for filenames in commands. A good example of using a device name was shown in Chapter 8, when the COPY command was used to take input from the keyboard rather than from a file already on disk and place this input in a text file on disk.

```
COPY CON: AUTOEXEC.BAT
```

This command says, "Take whatever comes into the memory of the computer from the the console (the keyboard), and record it on disk in the file called AUTOEXEC.BAT." If an attempt is made to use device names in place of filenames in inappropriate circumstances, one of these two messages will result:

```
File not found
Illegal parameter
```

12.2 Setting Up Devices with MODE

The MODE program in PC-DOS was written by IBM, not Microsoft. It is an external command and is very complicated. It actually has four different functions and could just as easily have been broken apart into four separate commands. In fact, it really is just like four separate commands that happen to start with the same word. The four functions it performs are:

1. Setting up column width and lines per inch on the IBM parallel printer.
2. Setting up output to a serial port (setting baud rate, data format, etc.)
3. Redirecting printer output from parallel to serial.
4. Switching video display modes.

Mode Version 1

The first version of the MODE command sets up the IBM parallel printer for operation with 80 or 132 characters per line and 6 or 8 lines per inch. When the printer is turned on, its default settings are normally 80 characters per line and 6 lines per inch. Telling it to put 132 characters on a line causes a special code to be sent out that puts it into "condensed" mode of operation, where the characters are smaller, so that 132 of them fit on a line, not just 80. Since IBM parallel printers are actually manufactured by Epson, this command works with Epson printers as well.

The general form of the command is:

```
MODE LPT#:(N)(,M)(,P)
```

Here's what the symbols stand for:

#	1, 2, or 3. Typically, LPT1 for the first parallel port.
N	Characters per line. Set at either 80 or 132. Default is 80.
M	Lines per inch. Set at either 6 or 8. Default is 6.
P	Printer. Keep trying to send despite time-out errors.

If the N or M parameters are left off, then they remain unchanged from their current value. If, for example, N was set to 132 earlier and then left off from a new command, it would remain 132. Here are some examples:

```
MODE LPT1:132,8,P
MODE LPT1:132,6
MODE LPT2:132
MODE LPT2:,8,P
MODE LPT1:132,,P
MODE LPT1:,,P
```

The P option controls what happens when a time-out error occurs. A time out is needed when an "error" occurs – for example, when there is no printer hooked up to the parallel port, or the printer is turned off, or it is "off line" (the power is on but it is not ready to receive).

Time-outs exist so that the computer won't just keep trying to send to a nonexistent or not ready printer forever. When it does try to keep sending, the computer no longer responds to the keyboard and is "frozen." This is called "hanging the system" and necessitates a reboot. Having to reboot causes any data in the memory to be lost. Rather than have this happen, after a certain period of trying, the system "times out." An error message is sent to DOS saying "Too much time has elapsed; let's forget the whole thing," and the computer stops trying to send to the printer.

This is all well and good in many cases, but sometimes it is necessary to ignore these time-outs because they occur too soon. A good example is when the printer is temporarily offline so that paper can be put in.

Including the P option says, "Ignore the time-out errors and keep trying to send." If retries are taking place, and it is necessary to break out of what might be a never ending cycle, a Ctrl-Break is entered. If the P option is left off the command, it does *not* remain unchanged (as the N and M values do), but rather is turned OFF. No retries will be made in that case.

Mode Version 2 SERIAL PORT

The second version of the MODE command is used to configure certain parameters of a serial port. This is also called **initializing** the port. When a peripheral device (typically a modem or printer) is going to be hooked up using a serial port, the baud rate, parity, number of databits, and number of stopbits may need to be specified. If the hookup is for a serial printer, a special parameter needs to be included to indicate this. Here's the general form of the command, along with an example:

```
MODE
COM#:BAUDRATE(,PARITY,DATABITS,STOPBITS,P)
mode com1:300,n,8,1,p
```

Let's examine each of these parameters in detail.

The # symbol stands for a 1 or 2. If it is serial port number 1 that is being configured, the entry is COM1:. If it is number 2, the entry is COM2:. This part of the command may not be left off.

Of the various parameters, the baud rate is the only one that cannot be omitted. Baud rate refers to the speed at which the data is transmitted. A rate of 300 means 300 bits per second. If there are ten bits in every character, which is the most typical situation, then a rate of 300 baud is 30 characters per second.

The valid baud rates that may be specified in this command are:

110	1200
150	2400

300	4800
600	9600

When these are entered as part of the command, only the first two characters must be typed in, as the rest are ignored anyway. The first two differentiate each rate from any other possibility, so the MODE command does not need to bother to check further.

Modems normally run at either 300 or 1200 baud. Modems capable of working at 1200 baud cost more but may be cost effective in the long run. Since they are four times as fast as 300-baud modems, for someone paying for time on line to another computer, this could reduce the bill substantially. Unfortunately, some of the well-known data base services don't miss a trick, and charge more for the time if 1200-baud communications are being used.

Parity bits, databits, and stopbits deal with the zeroes and ones being sent out one after the other. Each character, or byte, that is sent out consists of a series of bits. A bit is the minimal unit in a computer. Each bit has a value of either zero or one. The character is a sequence of bits. Sometimes it is made up of seven bits, sometimes eight bits. The character we call uppercase K looks like this: 1001011. The "databits" portion of the command refers to the number of bits in each sequence. It has a 7 or an 8 plugged into it.

As the bits are sent along a wire one after the other, there is always a risk that one or two of them might be lost or changed in transmission. This could have dire consequences, so a procedure is built in to watch out for this. The parity check is used for this purpose.

The transmitting device first sums up the zeroes and ones and ascertains whether this sum is odd or even. If it is odd, it adds another one, making it even, and then sends it. If the result is even, it adds a zero, leaving it even, and sends it. On the other end, as each sequence is received it is totaled up to ensure that the sum is still even. If it is not, the system knows an error has occurred.

This is called EVEN parity. Alternatively, the sum may have a one or zero added to it, as appropriate, to make it always turn out odd. In this case, ODD parity is being used. Sometimes NO parity is used. It is fixed that either a one or a zero must be added in. When a one is added each and every time, it doesn't mean anything, so parity is essentially being ignored. So the entry for parity in the MODE command is either N for None, O for Odd, or E for Even.

So far so good. The only question left is about differentiating sequences (i.e., characters) in this steady flow of bits. The letter K is 1001011, the letter L is 1001100, and the letter M is 1001101. If these three are sent one after the other, we get 100101110011001001101. This might be correct, but how does the receiver know when one ends and another starts?

This segmenting problem is solved with startbits and stopbits. Extra bits are tacked on to the beginning and end. They bracket the sequence so that it can be distinguished from the preceding sequence and the following one. The convention is to use one start bit, but it varies as to whether one or two stopbits

are used. For this reason, we need to specify this in the MODE command. The number plugged into this part of the command is either 1 or 2, whichever is the number of stopbits being used.

The P option here is the same as in the version of the MODE command used for setting up a printer. When it is included, time-out errors are ignored, and the computer keeps on trying to send to the printer. If the computer is in a neverending cycle of retries, this can be aborted with a Ctrl-Break. If the P is not included, time-out errors will cause the computer to stop sending. The P option normally must be included when a serial printer is hooked up. Many people have become very frustrated trying to figure out what was wrong with the wiring of the cable to a serial printer, or the settings of the switches inside it, when all that was needed was the P that was left off the MODE command.

Let's look again at the sample given earlier:

```
mode com1:300,n,8,1,p
```

It says, "Set up communications through serial port number 1 at 300 baud, with No parity, 8 databits, 1 stopbit, and retries in effect."

The baud rate must be specified but only the first two characters actually need to be typed in. The parity, databits, and stopbits specifications can be omitted. If they are left off, they remain unchanged from a previous setup. If they were not already set up, they take their default values. The default value for parity is E for Even. For databits it is 7. For stopbits it is 1, except when the baud rate is 110, in which case it is 2.

When any one of the specifications is left off, the commas that go between them all must still be there. Here are some examples:

```
mode com1:12,,,,p
mode com1:30,n,,,p
mode com1:12,,8,,
```

MODE VERSION 3 Using a SERIAL PRINTER

Next we move on to the third version of MODE. It is used to redirect output from a parallel port (normally LPT1) to a serial port. Here's the general form of the command, along with an example:

```
MODE LPT#:=COMn
mode lpt1:=com1
```

The colon in COMn has been dropped because it is not needed. The colon in LPT#: must be included or an "Illegal device name" error message results. The n in COMn is replaced by either a 1 or a 2.

This command is typically used when a serial printer we want to use is hooked up to COM1 or COM2, and the output of commercial programs would normally be going to LPT1, a very common occurrence. The command needs to be preceded by the second version of the MODE command, so that the serial port is first set up, or "initialized." This redirection to a serial port can be negated by using the first version of the MODE command, as shown in these two samples:

```
mode lpt1
mode lpt1:80,6
```

Often these two commands must be entered every time the system is started up. When this is the case, both should be included in the AUTOEXEC.BAT file of the boot disks that have programs that use the serial printer.

MODE VERSION 4 w/ COLOR GRAPHICS MONITOR

The fourth version of MODE is used for switching between different video display modes. It deals with displays of 40 characters per line versus 80 characters, and monochrome versus color. It also allows for an adjustment that shifts the entire display on the screen right or left one character if a color graphics board is in use. This command is important for people who have both monochrome and color graphics adapters in their computer. The general form is:

```
MODE DISPLAY,DIRECTION,T
```

The valid options for DISPLAY are:

40	Color graphics board set to 40-character display.
80	Color graphics board set to 80-character display.
BW40	Switch to the color graphics board, with 40-character display and black and white mode.
BW80	Switch to the color graphics board, with 80-character display and black and white mode.
CO40	Switch to the color graphics board, with 40-character display and color mode.

CO80	Switch to the color graphics board, with 80-character display and color mode.
MONO	Switch to the monochrome display board.

It is enough to have just the DISPLAY parameter specified with the command. Examples:

```
mode mono
mode bw40
```

The error message "Invalid parameters" appears if an attempt is made to access a nonexistent device; for example, a color graphics board when only a monochrome adapter is present.

The DIRECTION and T options are used to shift the display on the screen. They work only with a color graphics adapter, not with a monochrome adapter and video unit. Valid options for DIRECTION are R and L for right and left. If the T is included, a test pattern is placed on the screen. Here are some examples:

```
mode 80,r
mode 80,l
mode 80,r,t
mode 80,l,t
```

The first causes the entire display to shift one character to the right; the second causes a one-character shift to the left. The DOS prompt then reappears, and the command is finished. The third example shifts the display one character to the right, but the process is not finished. A test pattern appears on the screen so that the right-left alignment can be checked and shifted some more if necessary. The pattern consists of 40 or 80 digits (depending on whether 40- or 80-character mode is in use) across the top of the screen and is followed by a message:

```
Resident portion of MODE loaded
01234567890123456789.....8901234567890

Do you see the leftmost 0? (Y/N)
```

A response of n for No causes the screen to clear, everything to shift one character to the right, and then the same pattern and question to reappear. As long as the n key is used to repond No, this continues. When the y key is pushed, the process ends, and the DOS prompt comes back on the screen.

Using the fourth example initiates a similar procedure, except each shift is to the left, and the message is:

```
01234567890123456789.....890123456789

Do you see the rightmost 9? (Y/N)
```

Answering with the n key causes a one-character shift to the left, and so on until the y key is used to stop the process.

As with other versions of MODE, if a shift or display change must be entered at each startup, the command should be made a part of an AUTOEXEC.BAT file. If the screen must be shifted two characters, the command must be entered twice, as each time it is used it shifts everything only one character.

12.3 Remote Terminals: CTTY

It is possible to hook up a remote terminal to a serial port so that the computer can be controlled from a distant keyboard and screen (which is what a "terminal" is), rather than its own. The terminal is connected to the serial port, either directly or through a modem. The serial port must be initialized for the correct baud rate (normally 9600 for a direct link to a terminal) using the second version of the MODE command. Control can then be turned over to the remote terminal by using the CTTY command. The general form of this internal command is:

```
CTTY DEVICENAME
```

The actual possibilities are:

```
ctty aux
ctty com1
ctty com2
ctty con
```

When the command is entered, the main console – the computer's own keyboard and screen – goes dead, and the remote terminal takes over. A reboot will return control to the main console, but this might cause the loss of data. The CTTY command can be used to return control:

```
ctty con
```

(Remember that CON is the device name for the computer's own keyboard and screen, its CONSOLE.)

In reality, this command has some problems and limitations. It is recommended that a commercial program be purchased to run a remote terminal if this is really necessary.

12.4 Redirecting Output

Normally, output goes to the "standard output device," which is the video screen. So far we've seen some commands that can alter this setup, but they implement changes that stay in effect as long as the computer stays on, or until they are changed with another command.

Sometimes we may want to redirect output just once. For example, we might want a directory listing to be recorded (saved) in a file on disk, or we might want the message displayed by the CHKDSK program to be saved in a file. Also, rather than have the listing of what files are being copied by the BACKUP program go only to the screen, we might want this history printed on paper. These two types of redirection, to a file on disk or to the printer, are the most commonly used.

They can only be used to redirect standard output, which means output that commands or programs think is going to the screen. The "greater than" symbol, also known as the right angle bracket (>), is used to redirect output. Examples:

```
DIR > PRN
DIR > DIRLIST.APR
DIR > B:DIRLIST.MAY
```

These cause a directory listing to be routed to a device (here PRN) or a file (here DIRLIST.APR or B:DIRLIST.MAY), instead of to the screen. The general form of such a command is:

```
DOSCOMMAND > DEVICENAME
        or
DOSCOMMAND > FILENAME
```

DOSCOMMAND means any valid command that can be entered at the DOS prompt, including a command for running a program by typing in its name (e.g., CHKDSK > PRN). FILENAME is used here in the full sense that it has been throughout the book. That is, the filename will consist, at a minimum, of a corename but may include a drive specification, path, and extension as well.

When output is redirected to a file that does not yet exist on disk, it is created at that time. If the file exists on the disk already, it is overwritten by

the new one. So if we need to add to a file, not overwrite it, then double angle brackets are used (> >). For example, we might want to combine the directory listings of both drive A and B into a file for archival purposes. The commands would be:

```
dir a: > dirlist.apr
dir b: > > dirlist.apr
```

If the file does not exist, it is created the first time it is referenced, regardless of whether single or double angle brackets are used. After that, rather than the file being destroyed, the new one gets tacked on to the end of it. The file DIRLIST.APR will contain both directory listings, first that of the A drive, then the one from the B drive.

In Chapter 11, it was recommended that when the BACKUP command is used, the output should be redirected to the printer so that a record would exist of what had been backed up. The command might be:

```
backup c:\ a:/s > prn
```

Actually, if it is desired that the output go to both the screen and the printer, the Ctrl-P toggle can be used. By first entering a Ctrl-P, we tell DOS to open the port to the printer so that the output can flow to the printer as well as the screen. Normally this port will be LPT1, but the output can, of course, be redirected to COM1 with the MODE command. Another Ctrl-P must be entered later to shut off the flow of the output to the printer. But if the output is being redirected to a file on disk, then the only way is to use the redirection symbols > and > >.

Let's look now at an example of how to redirect the output of a program in BASIC from the screen to a text file on disk. This little program in BASIC does nothing more than count from 1 to 15 and display each number on the next line. Here's how to type it in and save it to disk. With a disk that has BASIC.COM on it in the current drive, the word "basic" is typed in to start BASIC running, followed by the ENTER key, of course. When the Ok prompt is on the screen, the program can be typed in exactly as shown:

```
10 cls
20 for x = 1 to 15
30 ? x
40 next x
50 ?
60 ? "Done!"
```

When this typing is finished, the run command should be typed in, followed by ENTER, to test the program. If the program works properly (no syntax errors, and the numbers 1 through 15 are displayed down the screen), then the program can be saved by typing in:

```
save "count
```

Then to return to DOS, the command is:

```
system
```

Now we're ready to do some redirection. When the COUNT program is run, it normally puts the numbers on the screen. To have the numbers put in a text file, the command is:

```
basic count > result
```

When the program is done, BASIC is still in effect, so it is necessary to type in SYSTEM to return to DOS. Then the TYPE command can be used to see what is in the file on the disk called RESULT.

```
type result
```

These are fairly simple examples of what can be done with redirection of output. Once the technique is mastered, it's just a matter of applying a little imagination to figure out what other uses it can be put to.

12.5 Redirecting Input

Normally input comes from the "standard input device," which is the keyboard. When we want to specify that input will be coming from another device or from a file, we use the "less than" symbol, also known as the left angle bracket (<).

The input might be coming in serial port number 1. The general form of the command in that case would be:

```
DOSCOMMAND < COM1
```

It's more likely that this redirection of input will be used to take input from a file on disk. For example, we can use BASIC to write a little program that asks the

user to type in three different numbers and then shows their sum on the screen. Instead of those numbers coming from the keyboard, they can come from a file on disk. Here's a sample program in BASIC that can be typed when BASIC is up and running.

```
10 cls
20 ? "Enter the first number:"
30 input a
40 ? "Enter the second number:"
50 input b
60 ? "Enter the third number:"
70 input c
80 ? "Thank you."
90 ? "The sum of the numbers is:"
100 ? a + b + c
```

This can be saved with the command SAVE "SUM. To return to DOS, the command is SYSTEM.

Whenever this program is run from the DOS prompt with the command BASIC SUM, it asks for three numbers to be typed in and then shows the sum on the screen. Before these three numbers can come from a file instead of the keyboard, there has to be a file on disk with the numbers in it. Make this file by using the COPY command:

```
copy con: numbers
13
8
21
^Z or F6
```

^Z means enter a Ctrl-Z at that point. Then when the command to run the program is entered, it can include directions to take input from the file called NUMBERS:

```
basic sum < numbers
```

As the program runs, the numbers are plugged in from the file, instead of being typed in at the keyboard.

12.6 Summary

Different kinds of devices, such as printers and modems, can be hooked up

to a computer. Some communicate in parallel, others serially. The hookup points for these devices are called ports. DOS has its own names for the various ports as we saw in Table 12-1.

These device names can be plugged in to replace filenames in DOS commands. Example:

```
COPY CON: AUTOEXEC.BAT
```

The MODE command is used for four different functions. The first is for setting up the IBM printer, and its form is:

```
MODE LPT#:N,M,P
```

The second MODE function is to initialize a serial port. The command's form is:

```
MODE COM#:BAUD,PARITY,DATABITS,STOPBITS,P
```

The third MODE function is to redirect output from a parallel port to a serial port. It is usually used to tell the system that a serial printer is hooked up, rather than a parallel one, as the system often assumes. The form of the command is:

```
MODE LPT#: = COM#
```

The fourth function is to switch video modes between color and monochrome displays and between 40 and 80 characters on the screen. The form of the command is:

```
MODE DISPLAY,DIRECTION,T
```

Control can be turned over to a remote terminal hooked up via a serial port using the CTTY command. It has limitations and a few problems, so for any serious reconfiguring of this nature, it is recommended that a commercial piece of software be acquired.

The standard input and output can be redirected using these symbols:

```
>      >>      <
```

These various redirecting capabilities are enhanced even further by pipes and filters as discussed in Chapter 13.

CHAPTER 13

Filters, Pipes and More

Pipes and filters interact to allow us to take data, do something to it, pass the changed output to another program, do something else to it, and pass it along again. Programs in the middle of such a chain of processing are referred to as **filters**, because the data is "filtered" through them and refined. **Pipes** pass the data along.

13.1 Filters: SORT, FIND and MORE

DOS 2.0 and 2.1 have three filter programs – SORT, FIND, and MORE, all three external. Unlike other DOS commands, which have only output, filters require input as well. They take their input from the standard input device (the keyboard), unless redirection is used. Each of the filter programs modifies the data and sends it to the standard output device (the screen), unless it is redirected.

SORT

The SORT program rearranges lines of text into alphabetical order. Unless otherwise specified, the lines are entered from the keyboard. Here's an example of what appears on the screen when a few lines are entered and sorted:

```
A> sort
lions
tigers
bears
```

```
anteaters
monkeys
^Z
anteaters
bears
lions
monkeys
tigers
```

Ctrl-Z (or the F6 key) is used to end entry of the lines. As soon as it is typed in and the ENTER key is pushed, the lines reappear in sorted order. Note that input came from the keyboard and output went to the screen.

We can use the techniques of redirection learned in the previous chapter, to have the input come from a text file on disk. The COPY command can be used to place a small sample file of names on the disk:

```
copy con: names
Hallie
Charlie
Pam
Gary
Darlene
Lee
Sandy
Dave
^Z                    (or F6)
```

Once this is done, we can use the input redirection symbol in a SORT command:

```
sort < names
```

This says, "Run the SORT program and take the input from the file called NAMES instead of from the standard input device." The screen shows:

```
A> sort names
Charlie
Darlene
Dave
Gary
Hallie
Lee
```

```
Pam
Sandy
```

If we want the output to be saved in a separate file on disk, this can be accomplished with the following command:

```
sort < names > names.sor
```

With the SORT program, sorting is done one line at a time. The first characters of each line are compared and ordered. If necessary to break "ties," the second character, the third character, and so on are also compared. The sort is according to the ASCII value of the characters being compared. A glance at the chart in Appendix D shows that the character A has a decimal value of 65, B is 66, Z is 90, lowercase a is 97, and b is 98. The character representing the number one, 1, has a value of 49. What this means to us is that the sort may not come out exactly as we might anticipate. All uppercase letters come before lowercase letters, and number characters come before both of those. So, a mixed list like this one comes out sorted as shown:

```
lions                          255
Glenn                          477
255              = =>          Glenn
anteater                       Zelda
Zelda                          anteater
477                            lions
```

If we want, we can reverse the normal sorting order by including the /R option. For example:

```
A> sort/r < names
Sandy
Pam
Lee
Hallie
Gary
Dave
Darlene
Charlie
```

FIND

The FIND program is used to extract lines from the input, lines that meet

a certain criterion. The criterion is that they contain the string of characters specified. A simplified form of the command is:

```
FIND "string" FILENAME
```

Redirection for input is not needed, as this program is specifically designed to find the string in a file on disk. If a filename is not included, the input is expected to be from the standard input device. The string of characters to be found is enclosed in quote marks.

Using our now-familiar file called NAMES, we can have the FIND program extract the lines that contain the sequence "ar." The command, and the results it gives, are shown here:

```
A> find "ar" names

----- names
Charlie
Gary
Darlene
```

The name of the file is always specified with the output, because more than one file can be searched. The full, general form of the command is:

```
FIND "string" FILENAME FILENAME .... etc.
```

Working on a disk with three different files of names, the command and its results might look like this:

```
find "ar" names1 names2 names3

----- names1
Charlie
Gary
Darlene

----- names2
Marlene
Darla

----- names3
Mark
Marcus
```

The FIND command includes three options, /V, /N, and /C. The /V option specifies that lines that meet the criterion are to be excluded. Only lines that do not meet the criterion will be included in the output. Here's an example:

```
find/v "ar" names

---------- names
Hallie
Pam
Lee
Sandy
Dave
```

The /N option says, "Number the lines and show these line numbers." An example, using our trusty file called NAMES, looks like this:

```
find/n "ar" names

---------- names
[2]Charlie
[4]Gary
[5]Darlene
```

The /C option says, "Give me a count of how many lines met the criterion." The actual lines are not shown, just the count of how many there are. Example:

```
find/c "ar" names

---------- names: 3
```

The /V and /N options can be used together, but the /C is exclusive. Using the /V and /N together produces:

```
find/v/n "ar" names

---------- names
[1]Hallie
[3]Pam
[6]Lee
[7]Sandy
[8]Dave
```

MORE

The MORE command has a function identical to the /P option that goes with the DIR command. It stops the output to the screen every time the screen is full, and waits for any keystroke to continue. With the file DIRLIST.APR on disk, the following command displays it on the screen:

```
more < dirlist.apr
```

The lines of this text file scroll up the screen until the first one reaches the top. The listing halts, and at the bottom of the screen, the message is:

```
 – More –
```

Pushing any key allows the scrolling to continue.

13.2 Passing Data via Pipes

There are times when we use one command to create some output, and then take that output and use it as the input to a second command. Here's a simple example:

```
sort < names > names.sor
find "ar" names.sor
```

The SORT command takes input from the file called NAMES, sorts the lines, and sends the output to a file called NAMES.SOR. The FIND command is then used to extract and display the lines that include the characters "ar."

Instead of typing in the two commands separately, we can use a PIPE to take the output of one command and pass it along to be used immediately as the input to another command. The bar symbol (|), also known as the "up and down line," is used to initiate this function.

These two commands above can be combined with a pipe:

```
sort < names | find "ar"
```

In this case no filename is specified with the FIND command, causing it to expect to get input from the standard input device. The pipe provides input from the output of the SORT command, just as if this input had come from the standard input device. If the list of names in the NAMES file is quite long, the MORE filter can be included so that the display goes one screenful at a time.

```
sort < names | find "ar" | more
```

The classic use of piping is to get a sorted directory listing:

```
dir | sort
```

And since directory listings often are more than a screenful, the MORE filter can be used:

```
dir | sort | more
```

Since the nature of the MORE command is to act on the way the final output gets to the screen, it is always the last command in any sequence that it appears in.

The FIND command can be used to filter a directory listing to extract only files with a certain date:

```
dir | find "4-07-84"
```

This type of piping of output for another program's input creates and saves temporary files at the root level on the current disk drive. These temporary files always have a name like this:

```
%PIPEx.$$$
```

DOS assigns a number in place of the x shown here. These files will sometimes show up in a directory listing, since they exist at the time this listing is pulled off the disk. They are erased from the disk at the very end of the piping process and do not show up on normal directory listings invoked after that.

Some of these commands can be useful when dealing with a large number of files in one subdirectory on a hard disk. With accounting applications, this situation arises frequently. To get a look at all the names of the data files (let's say their names end in .DAT, which is common) and have this listing sorted, the command is:

```
dir | find ".DAT" | sort | more
```

If commands like this are used frequently, they should be put in a batch file to make it easier to use them. Replaceable parameters (%1, %2, %3, etc.) can be used to allow for great flexibility in getting exactly the kind of results desired.

13.3 Printing Text Files: PRINT

Printers cannot print out characters nearly as fast as a computer can send them. Because of this, there is an ongoing process of telling the computer to start and to stop sending. Printers have a little bit of memory of their own, referred to as a buffer. This buffer can store small numbers of characters. How much it holds depends on the brand.

When starting, the computer sends a special code to the printer, a "request to send" message. If the printer is ready, it sends back an acknowledgment code that says, "Go ahead." When the buffer gets full, the printer sends a code that says, "Halt the flow." As printing continues, characters that are printed are no longer held in the buffer. This frees up space. When enough buffer space is available, the printer sends another "go ahead."

This sending of start and stop messages is called **handshaking** or a **protocol.** Different sets of codes are used for different protocols. XON/XOFF is one; ETX/ACK is another. Variations in the way codes are used and which wire carries which signal are what make interfacing printers one of the big problems in the computer industry, because there is no one standard way to do it.

While the computer is on hold so the printer can catch up, the microprocessor chip is idle. The PRINT command is an attempt to allow for more efficient use of the processor. It can be used to start a printout of a text file and then go on to something else. When the processor must send more characters to the printer, it does so. When it is free from this task (which is a large part of the time) it proceeds with any other task that we have given it. This may sound very efficent, but as implemented with the PRINT command, it has severe limitations.

No matter how fast a microprocessor chip is, it can only do one thing at a time. When we try to issue other DOS commands while the PRINT command is working, there are times when the computer does not respond immediately, because the processor is busy taking care of the printout. It is quite disconcerting to most people to type in five or six characters and see nothing on the screen, then have them appear all of a sudden after a one- or two-second lag.

The most important of the limitations has to do with the printing itself. Only standard text files can be used. If the file has Control characters in it (which files created by many word processing programs do), these characters can wreak havoc on what happens on the printer. Word processing programs often use control characters as special codes, for underlining and bold print, for example. The word processing program's text formatter responds to them while printing is taking place, telling the printer how to underline or print in bold. The PRINT command has no provisions for such things. No underlining, bold print, superscripts, subscripts, and so on are possible.

Further, PRINT does not take into account stopping between pages; the printout just chugs along until it is done, regardless of whether it's one page or ten. It does not know when it goes from one page to the next; lines are just printed one after the other. A word processing program keeps track of the

number of lines and leaves blank ones at the tops and bottoms of the printed pages (the number of blank lines is something the user controls). So, to accomplish page breaks using PRINT, the text file must already have blank lines built in right where the break between pages occurs, and printing must be started at exactly the right place on the first piece of paper.

Such documents exist. There are commercial and public domain programs that come with their documentation in a text file on disk. A well-known example of this is the Freeware program PC Talk III, a communications program. It has 70 pages of text on disk, preformatted to print properly using the TYPE or PRINT commands.

The general form of this command is:

PRINT FILENAME1 FILENAME2 . . . FILENAME10

When more than one filename is given, they are put in a queue. "Queue" is the word the British use for a line. "Queueing up" is getting in line. Files that are queued up are on hold and will be printed in the order in which their names were entered.

The PRINT command is most useful when we have a group of text files to be printed out. Rather than have to use the TYPE command over and over, we can list all the files at once. If their names have something in common, wildcards can be used. Batch files are a good example of this. If we have four batch files on a disk (1.BAT, 2.BAT, 3.BAT, and 4.BAT) that we want printed out, the most efficient way is with this command:

---

print *.bat

The first time PRINT is used, it comes back with a query:

Name of list device [PRN]:

It's asking which port to direct the output to. The fact that it says PRN in brackets is a cryptic but commonplace way (among programmers, anyway) to say, "If you just hit ENTER, I'll know you want to accept PRN as your answer." PRN is the device name for parallel port number one (it's the same as LPT1). If the output needs to go out a serial port, this should be set up in advance with the MODE command (using the MODE version that initializes the port for baud rate, and so on, and then using the version that redirects from LPT1 to COM1 or COM2). Then when this question comes up, all that has to be done to begin the PRINT process is to push the ENTER key. After that has been done, the screen shows:

Resident portion of PRINT installed

```
A:1.BAT is currently being printed
A:2.BAT is in queue
A:3.BAT is in queue
A:4.BAT is in queue
```

The program does try to go to a new page each time it starts to print a different file, but whether this actually occurs depends on the printer. The printer must be capable of responding to the special code normally associated with this function (called a "page eject").

If there are more than ten batch files on the disk (let's say 1.BAT through 12.BAT), the PRINT program shows something like this on the screen:

```
A> print *.bat
PRINT queue is full

A:1.BAT is currently being printed
A:2.BAT is in queue
A:3.BAT is in queue
A:4.BAT is in queue
A:5.BAT is in queue
A:6.BAT is in queue
A:7.BAT is in queue
A:8.BAT is in queue
A:9.BAT is in queue
A:10.BAT is in queue
```

The last two files, 11.BAT and 12.BAT, would still need to be printed. Even while the PRINT program is still in the process of printing the first ten, the following command could be entered:

```
print 11.bat 12.bat
```

These would then be added to the end of the queue, provided that PRINT has already finished printing two files, leaving room for two to be added on.

PRINT has three options, /T, /C, and /P. The /T is for Terminate. Entering a PRINT command with this option (PRINT/T) causes the printing to stop and all the files in the queue to be cleared. The message "All files canceled by operator" is printed out on paper.

The /C is for Cancel. It says, "Don't print the file whose name precedes this option and files whose names follow in this command." The name of the file appears on the printout, followed by the message "File canceled by operator." For example, if we want to print all the batch files except 7.BAT, 8.BAT, and 9.BAT, we can enter:

```
print *.bat 7.bat/c 8.bat 9.bat
```

If the command to cancel a file is issued while it is actually being printed, that printout is aborted immediately, and PRINT goes on the next file in the queue.

The /P is for Print. It says, "Include the file whose name precedes this option and files whose names follow in this command." A /P cancels a /C, and vice versa. The way to describe their scope of operation (what files they include) is to say, "Include/cancel the preceding file and all files following the option, or until another option is specified."

We can use this to have selected files put in the queue. Let's say the printing has already been started with the command:

```
print ??.bat
```

This will match up to and thus include our batch files named with numbers 1 through 12 but will not include AUTOEXEC.BAT, SAMPLE-.BAT, and TEST.BAT. We decide after printing has commenced that we want to include these last three and we want to tack on 11.BAT and 12.BAT and at the same time cancel 6.BAT and 7.BAT. The following command will do this:

```
print 6.bat/c 7.bat 11.bat/p 12.bat autoexec.bat sample.bat test.bat
```

When the PRINT command is issued with no filenames or options accompanying it, it reports back the status of the printing operation with "PRINT queue is empty," or something like:

```
A:5.BAT is currently being printed
A:6.BAT is in queue
A:7.BAT is in queue
A:8.BAT is in queue
A:9.BAT is in queue
A:10.BAT is in queue
```

13.4 Printing Graphics: GRAPHICS

Anytime there is text on the screen, the Shift-PrtSc command tells DOS to print everything on the screen. This is called a "screen dump." If a graphics image is on the screen, this same procedure can be used to dump the contents of the screen to an IBM printer, but *only* if the GRAPHICS command has been invoked previously.

The GRAPHICS command is external, and it has no parameters or options. It is another program that comes from IBM, not Microsoft's generic version of MS-DOS 2.0. When it is used, the GRAPHICS.COM program loads a printer driver into the computer's memory. A "driver" is a little program that controls exactly how the information is sent out to a specific brand of printer. Once this programming is resident in the computer's memory, DOS knows how to send a graphics image on the screen to the printer in a way that the printer is prepared to handle.

The GRAPHICS program differentiates between two modes, called Screen1 and Screen2. The first has a resolution grid of 320 X 200. When this is in use, the screen image is printed with up to four shades of gray (corresponding to the four colors allowed on the screen). The second has a resolution of 640 X 200 and allows only black and white. The screen image is printed out sideways on the paper. It is rotated 90 degrees to the left, so that the upper right corner of the screen image is in the upper left corner of the paper.

If a BASIC program is running, using Shift-PrtSc to dump a graphics image causes the program to halt until the printout is done, and then the BASIC program continues. There are commercial programs available that are designed to allow graphics dumps which are more sophisticated than those provided by GRAPHICS.

13.5 Electronic Spoolers and Disks

Because too much time is wasted while the computer waits for the printer to finish a job, the PRINT program was devised to allow other work to take place at the same time that printing goes on. Unfortunately, its practicality is very limited.

A good way to free up the microprocessor from the doldrums of waiting for a printer is to use a print buffer, or spooler. A print buffer is something that takes the characters being sent to the printer and temporarily stores them. Once all the characters have been sent to this temporary storage area, the computer thinks the printing has been completed and can go on to the next task. The buffer device continues to feed the characters to the printer until the job is done.

There are two kinds of buffers, external and internal. An external buffer is a box with memory chips in it. The cable coming out of the computer that would normally go to the printer is attached to this box instead, and a separate cable goes from the box to the printer.

The computer does not even know the buffer exists. Besides the memory chips, there are some chips in the box that have programming built in. This programming is necessary to do the "handshaking" that controls the flow of characters to the box, and also controls the flow of characters from the box out to the printer.

How much the buffer can hold depends on the amount of memory it has. If it has something like 16K (16,000 bytes, or characters), it can hold quite a few pages of material, but it still might not hold the entire document that is to be printed.

If it cannot hold the entire document, the first 16K of characters are sent out immediately. When the buffer gets full, a signal is sent to the computer to stop sending. As characters are printed out, the buffer has room again, and so a signal is sent to the computer to send more. This is just the normal handshaking kind of procedure that a printer would do on its own. The difference here is that when the final 16K of characters get to the box, the computer thinks it has finished printing and it is free to go on to another task.

With 64K of memory, most documents to be printed are going to fit in their entirety into the buffer. Of course, the more memory in the buffer, the more expensive it is.

The other kind of buffer, the internal one, is part of the computer's memory. If the computer has enough memory to hold all the programming and data necessary and still has some left over, it may be possible to allocate some of that memory as a print buffer area. To do so requires a program. This kind of program is called a print spooler. It "spools" the characters out to the printer one at a time. It is called an electronic spooler because it is in the electronics of the computer, as opposed to outside in some separate box.

The first thing such a program must do is partition off some of the computer's memory to hold the data being sent to the printer. Then it must redirect any output that is supposed to go to the printer to this segment of memory. Next, it must control the flow of characters from this memory to the printer.

A similar process can be used to set up a portion of memory to seem like a disk drive to the computer. This can be beneficial because storing and retrieving data on a disk is by far the slowest kind of operation a computer performs. Work slows down considerably when a program must constantly go out and get a piece of data from the disk, bring it into the working memory, process it, and then go out to the disk again and get the next piece. If the disk drive has stopped spinning in the interim, it takes a good portion of a second to get most drives restarted and up to full speed.

Manipulating data in the memory of the computer goes very, very fast. So one good way to speed things up is to have some memory in a computer blocked off, and trick the microprocessor into thinking that this portion of memory is a disk drive. The most common name for this is electronic disk. It is also known as a RAM disk. Some of the RAM, Random Access Memory, is being used as if it were a disk drive. Since it is behaving like a disk drive, it is occasionally referred to as a pseudodisk as well.

The application program being used tells the system to record some data, and the system sends the data to what it thinks is a disk drive. In actuality the "disk drive" is the portion of memory that was blocked off. The process takes very little time because it is memory rather than a mechanical device that is involved. There is no motor to be started, no read/write head moving across a disk, no bit-by-bit changing of magnetic fields.

There are three things to note here. First, the computer must have extra memory installed in it, as well as some software that handles blocking off a portion of that extra memory.

Second, each time the system is restarted, all programming and data files must be copied into the electronic disk (using COPY). If any modifications are made to the files, then they must be copied back to disk before the computer is turned off, or the changes will be lost.

Third, because working memory is volatile (that's the fancy jargon for "not permanent"), there is some danger of losing what has been recorded in the electronic disk. If a power failure occurs while work is in progress, the contents of the electronic disk will be lost. If the work being done is critical, it may be better to sacrifice that extra speed for the assurance that the data is being recorded on a disk. Power failures normally do not affect what is on a disk.

An electronic disk is most useful when information that already exists on a floppy disk can be put into it to be manipulated. Maybe a giant list of names has to be sorted and then printed. With many data base programs, an electronic disk can speed up such an operation significantly.

Batch files can run the program or programs that set up electronic spoolers and disks. The same batch file can contain both commands to copy the appropriate programs and data files to an electronic disk, and commands to start up immediately whatever program is to be used. This can be done in different ways depending on the hardware and software.

An extra board can be plugged into one of the available slots of the computer to extend the memory to more than 256K. With IBM PC or PC-compatible computers, there are many add-on boards on the marketplace that provide this memory expansion. In fact, there are many multifunction plug-in boards. They contain extra memory for the computer to use, serial and parallel ports, and print spooling capabilities (you get the programming with it on disk).

The SixPakPlus from AST Research is one of the well-known multifunction boards. This particular board includes extra memory, a serial port, a parallel port, and a clock function (with battery). It comes with programs on disk to set up electronic print spooling and electronic disks and to read the date and time from the clock. The program to read in the date and time is ASTCLOCK.COM, and the program to set up an electronic disk is SUPERDRV.COM. The program to set up an electronic print spooler is SUPERSPL.COM.

Here is an example of a batch file for a computer with a SixPakPlus board. It creates an electronic disk, copies the dBASE II program into it along with its data files, then, when the dBASE session is over, copies everything back to drive A (assumed here to be the current drive) and, finally, copies the data files to a backup disk in drive B.

Filename: START.BAT

```
echo off
rem      Read the system date and time:
astclock
echo Creating electronic disk drive C:
```

```
superdrv c:/1
echo Copying the dBASE programming and data to C:
copy db*.* c:
copy *.prg c:
copy *.dbf c:
copy *.ndx c:
copy *.frm c:
rem     Now start up dBASE.
dbase
rem     dBASE is done, clear the screen and go on.
cls
echo Copying the files back from the
echo electronic disk to the floppy disk.
rem     Assume current drive is A, so no "destination" needed.
copy c:*.dbf
copy c:*.prg
copy c:*.ndx
copy c:*.frm
cls
echo Done updating the floppy disk.
echo Please place the backup floppy disk in drive B.
pause
echo Copying the files to the backup disk.
copy *.dbf b:
copy *.prg b:
copy *.ndx b:
copy *.frm b:
cls
echo Done. You can shut off the computer now or go on
echo to whatever you want to do next.
```

In the old days (two or three years ago), memory was expensive. Now, multifunction boards are available with everything except the ability to do the typing for us. More and more commercial programs require large amounts of memory – 256K or even 512K – to run. Having lots of memory will prove worthwhile sometime, so there is no reason to put off getting it. A board purchased to add this memory should include print spooling and electronic disk software. Electronic spoolers and disks are something everyone should be taking advantage of.

13.6 Summary

Filters are programs that process text given to them and pass it along in a different form. Pipes pass along the output of one program to be used as input

to another program. The three filter programs that come with DOS are SORT, FIND, and MORE.

SORT sorts lines of text into an order based on the ASCII values of the characters.

FIND searches through text, normally in a file, to find a specified string.

The search will exclude, rather than find, all lines that meet the stated criterion if the /V option is included. The MORE program simply stops the scrolling every screenful when a file is displayed on the screen.

Piping is done with the bar (|) symbol. To take a directory listing, sort it, select only files with an extension of .DAT, and then list this screenful by screenful, the command is:

```
dir | find ".DAT" | sort | more
```

The PRINT command prints out text files. Its practicality is limited, because it does not take into account where page breaks need to occur, nor can it do such formatting as underlining, bold print, or superscripting.

The GRAPHICS program is used to prepare the computer to do a screen dump of a graphics image instead of text. To initiate a graphics screen dump, the Shift-PrtSc command is used.

Electronic spoolers and disk drives are portions of RAM that act as buffers, storing data in a temporary holding area. A print spooler continues to feed this data out to a printer one character at a time. An electronic disk holds data just as a disk would.

APPENDIX A

Summary of DOS Commands Covered in the Book

ASSIGN

Task: Diverts input and output from one disk drive to another.
Type: External
Syntax: ASSIGN drive1 = drive2, etc.
Sample: ASSIGN B = C
ASSIGN A = C B = C

BACKUP

Task: To make backup copies of files from a hard disk to one or more floppy disks.
Type: External
Syntax: BACKUP WHAT DESTINATION/options
Options: /S Include all subdirectories.
/M Include only modified files.
/A Add to the floppy disks, rather than overwrite.
/D: <date> Include only files made or modified after a specified date.
Sample: BACKUP C:\MISC*.TXT A:

BATCH FILE SUBCOMMANDS

ECHO Display a one-line message.
FOR.IN.DO Do a command until the list is exhausted.
GOTO label Jump to a line with a special label on it.
IF Execute a command only on certain conditions.
PAUSE Halt execution until a key is pushed.
REM Remark.
SHIFT Shift over to the next replaceable parameter.

CHDIR or CD

Task: Change the current directory, or display what the current directory is.
Type: Internal
Syntax: cd drive-path
Sample: CD C:\LOTUS\FILES84

CHKDSK

Task: Report on file space remaining, memory remaining, and the directory of the disk.
Type: External
Syntax: CHKDSK drive
Sample: CHKDSK B:

CLS

Task: Clear the screen display.
Type: Internal
Syntax: CLS
Sample: CLS

COMP

Task: Compare two files; report on whether they are identical or not.
Type: External
Syntax: COMP FILENAME1 FILENAME2
Sample: COMP A:FY84.WKS B:FY84.WKS

COPY

Task: Copy files from one disk to another.
Type: Internal
Syntax: COPY FILENAME DESTINATION/option
Options: /V Verify ON. Double-checks the copy.
Sample: COPY B:*.DBF C:\DBASE\PAM
COPY *.* B:

CTTY

Task: Switches control of the computer from the console to a remote terminal, or brings control back to the main console.
Type: Internal
Syntax: CTTY DEVICENAME
Sample: CTTY COM1:

DATE

Task: Allow the user to change the system date.
Type: Internal
Syntax: DATE
Sample: DATE

DEL

Task: Delete a file or files from disk. Same as ERASE.
Type: Internal
Syntax: DEL FILENAME
Sample: DEL FY84.WKS
DEL C:\LOTUS\PAM\FY84.WKS

DIR

Task: Produce a directory listing of what is on a disk.
Type: Internal
Syntax: DIR DRIVE-PATH-NAME/options
Options: /W Wide. Display the listing with 5 columns.
/P Pause. Halt the listing every screenful.
Sample: DIR/W
DIR B:
DIR C:\LOTUS\GARY
DIR B:*.BAS

DISKCOMP

Task: Compare two disks to ensure that they are identical.
Type: External
Syntax: DISKCOMP DRIVE1 DRIVE2/options
Options: /1 Compare only one side.
/8 Compare eight tracks per sector (DOS 1.1 format)
Sample: DISKCOMP A: B:

DISKCOPY

Task: Copy the entire contents of one disk onto another. Includes formatting if the disk is not already formatted.
Type: External
Syntax: DISKCOPY DRIVE1 DRIVE2/options
Options: /1 Copy only one side.
Sample: DISKCOPY A: B:

EDLIN

Task: Edit a text file.
Type: External
Syntax: EDLIN FILENAME
Sample: EDLIN AUTOEXEC.BAT

ERASE

Task: Delete a file or files from disk.
Type: Internal
Syntax: ERASE FILENAME
Sample: ERASE CHAP11
ERASE C:\BOOK\CHAP11
ERASE *.BAK

FIND

Task: A filter program that extracts lines from a text file that meet a specified criterion.
Type: External
Syntax: FIND/options "string" FILENAME
Options: /V Exclude. Find only lines without the string.
/C Count the number of lines found.
/N Number each line that is found and displayed.
Sample: FIND "ar" NAMES.TXT

FORMAT

Task: Place the IBM format on the disk.
Type: External
Syntax: FORMAT DRIVE/options
Options: /S System. Record DOS on the disk after formatting it.
/1 Format only one side of the disk.
/8 Use only eight sectors per track.
/V Allow a volume name to be entered.
/B Format eight sectors per disk and leave room for DOS to be added later.
Sample: FORMAT B:
Note: USE CAUTION! This command wipes a disk clean, including a hard disk. Remove this program from a hard disk. Put in a floppy that has it when it is needed.

GRAPHICS

Task: Allow a screen dump of a graphics image. The Shift-PrtSc command is used to get the printout.
Type: External
Syntax: GRAPHICS
Sample: GRAPHICS

MKDIR or MD

Task: Create a subdirectory.
Type: Internal
Syntax: MD DRIVE-PATH
Sample: MD C:\MISC\BOOK

MODE

Task: Four tasks.
1. Set parallel printer characteristics.
2. Set up a serial port.
3. Redirect from parallel port one to a serial port.
4. Change the video display mode.

Type: External
Syntax:
1. MODE LPT#:N,M,P
2. MODE COM#:BAUD(,PARITY,DATABITS,STOPBITS,P)
3. MODE LPT#:=COM#:
4. MODE DISPLAY,DIRECTION,T

Sample:
1. MODE LPT1:132,8,P
2. MODE COM1:12,N,8,1,P

3. MODE LPT1:#COM1:
4. MODE BW80,R,T

MORE

Task: A filter program. Stops output to the screen every screenful.
Type: External
Syntax: MORE
Sample: MORE < NAMES
DIR | SORT | MORE

PATH

Task: Establish one or more search paths. When an attempt is made to run a program, if it is not found immediately in the current directory, the directories specified earlier with this command are searched.
Type: Internal
Syntax: PATH path1;path2;path3;etc.
Sample: PATH \DOS;\BOOK;\LOTUS\FILES

PRINT

Task: Print up to ten text files while the computer is allowed to go on to another task.
Type: External
Syntax: PRINT FILENAME1/option,.....,FILENAME10/option
Options: /P Print. Add the file to the queue.
/T Terminate. Abort the printing.
/C Cancel. Remove the specified files from the queue.
Sample: PRINT AUTOEXEC.BAT,START.BAT,DOBACKUP.BAT
PRINT *.BAT

PROMPT

Task: Change the system prompt line.
Type: Internal (the IBM documentation is wrong)
Syntax: PROMPT new prompt line
Codes: $t System time.
$d System date.
$p Current directory.
$n Current drive.
$g >
$_ Go to new line (carriage return and line feed).
$l <
$b |
$q =
$h Destructive backspace.
$e Escape.
Sample: PROMPT Current directory: p_ng
PROMPT $d $t p_ng

RENAME

Task: Rename a file.
Type: Internal
Syntax: RENAME OLDNAME NEWNAME
Sample: RENAME FY84.WKS FY84DONE.WKS

RESTORE

Task: Bring back files that were copied using the BACKUP command.
Type: External
Syntax: RESTORE FROM WHAT&WHERE
Options: /S Include all subdirectories.
/P Prompt the user that a file is about to override one that has been changed since it was backed up.

Sample: RESTORE A: C:\LOTUS\FILES
RESTORE A: C:\/S
RESTORE A: C:\BOOK

RMDIR or RD

Task: Remove an empty subdirectory.
Type: Internal
Syntax: RD path
Sample: RD \BOOK
RD \LOTUS\82

SORT

Task: Filter program. Sorts lines of text into alphabetical order.
Type: External
Syntax: SORT/option
Options: /R Reverse. Sort in reverse order.
Sample: SORT < NAMES > NAMES.SOR
DIR | SORT | DIRLIST.TXT

SYS

Task: Record DOS on a disk. Only works when the space is available on the disk.
Type: External
Syntax: SYS drive
Sample: SYS B:

TIME

Task: Allow the user to change the system time.
Type: Internal
Syntax: TIME
Sample: TIME

TREE

Task: Display a list of all subdirectories. Does not include the root level.
Type: External
Syntax: TREE/option
Options: /F Include all the files in each subdirectory.
Sample: TREE
TREE/F

TYPE

Task: Display the contents of a text file on the screen. No wildcards allowed in name.
Type: Internal
Syntax: TYPE FILENAME
Sample: TYPE AUTOEXEC.BAT
TYPE STARTER.BAT

VERIFY

Task: Turn ON, OFF, or show status of the verification process. When ON, double checks that a file is readable anytime a file is recorded on disk.
Type: Internal
Syntax: VERIFY
Options: ON Turns it on.
OFF Turns it off.
Sample: VERIFY (shows status)
VERIFY ON
VERIFY OFF

APPENDIX B

Summary of DOS Commands Not Covered in the Book

BREAK

Task: ON/OFF Changes how often DOS checks the keyboard to see if a Ctrl-Break has been entered.
Type: Internal

CONFIG.SYS If this file is on the boot disk, DOS will find it at bootup and read the following:

BREAK Same as a BREAK command on its own.
BUFFERS Changes the number of disk buffers held in memory.
DEVICE Specify a file that has a custom device driver.
FILES Change maximum number of files that can be open at once.
SHELL Specify that your own command processor be loaded instead of COMMAND.COM.

Type: Internal (all of them)

DEBUG

Task: A programmer's tool.
Type: External

EXE2BIN

Task: Convert .EXE files to .COM files.
Type: External

FDISK

Task: Prepare a hard disk, called a fixed disk by IBM, for use. Create partitions for different operating systems. Does not include formatting.
Type: External

LINK

Task: Programmer's tool.
Type: External

RECOVER

Task: Recover what is left of file when portions of it are on bad sectors. Recover a damaged disk directory. It is very risky business trying to recover a blown directory. It can easily wipe out the entire directory!

SET

Task: Sets the system environment.
Type: Internal

VER

Task: Shows the Version number, 2.0 or 2.1.
Type: Internal

VOL

Task: Shows the Volume name of the disk, which is also shown every time we get a directory listing.
Type: Internal

APPENDIX C

Summary of Internal Versus External DOS Commands

Internal	External
BREAK	ASSIGN
Subcommands in a Batch file:	BACKUP
ECHO	CHKDSK
FOR	COMP
GOTO	DEBUG
IF	DISKCOMP
PAUSE	DISKCOPY
REM	EDLIN
SHIFT	EXE2BIN
CHDIR (CD)	FDISK
CLS	FIND
Subcommands in a CONFIG.SYS	FORMAT
BUFFERS	GRAPHICS
BREAK	LINK
DEVICE	MODE
FILES	MORE
SHELL	PRINT
COPY	RECOVER
CTTY	RESTORE
DATE	SORT
DEL	SYS
DIR	TREE
ERASE	
MKDIR (MD)	
PATH	
PROMPT	
RENAME	
RMDIR (RD)	
SET	
TIME	
TYPE	
VER	
VERIFY	
VOL	

APPENDIX D

ASCII Code Chart

ASCII stands for American Standard Code for Information Interchange. This chart shows the ASCII symbols (what we think of as characters) and the numbers that are associated with them in a computer. The numbers are in three different counting systems: a decimal-based one (a base ten counting system), a hexadecimal one (base eight) and a binary one (base two). The chart will allow conversions to be made easily by just checking across the chart. Some of the ASCII characters have symbols most people are not familiar with, for example, NUL, and SOH. These are explained at the end in a separate table.

ASCII	Decimal	Hexadecimal	Binary	ASCII	Decimal	Hexadecimal	Binary
NUL	0	0	0000000	!	33	21	0100001
SOH (CTRL-A)	1	1	0000001	"	34	22	0100010
STX (CTRL-B)	2	2	0000010	=	35	23	0100011
ETX (CTRL-C)	3	3	0000011	$	36	24	0100100
EOT (CTRL-D)	4	4	0000100	%	37	25	0100101
ENQ (CTRL-E)	5	5	0000101	&	38	26	0100110
ACK (CTRL-F)	6	6	0000110	'	39	27	0100111
BEL (CTRL-G)	7	7	0000111	(	40	28	0101000
BS (CTRL-H)	8	8	0001000	)	41	29	0101001
HT (CTRL-I)	9	9	0001001	*	42	2A	0101010
LF (CTRL-J)	10	A	0001010	+	43	2B	0101011
VT (CTRL-K)	11	B	0001011	,	44	2C	0101100
FF (CTRL-L)	12	C	0001100	-	45	2D	0101101
CR (CTRL-M)	13	D	0001101	.	46	2E	0101110
SO (CTRL-N)	14	E	0001110	/	47	2F	0101111
SI (CTRL-O)	15	F	0001111	0	48	30	0110000
DLE (CTRL-P)	16	10	0010000	1	49	31	0110001
DC1 (CTRL-Q)	17	11	0010001	2	50	32	0110010
DC2 (CTRL-R)	18	12	0010010	3	51	33	0110011
DC3 (CTRL-S)	19	13	0010011	4	52	34	0110100
DC4 (CTRL-T)	20	14	0010100	5	53	35	0110101
NAK (CTRL-U)	21	15	0010101	6	54	36	0110110
SYN (CTRL-V)	22	16	0010110	7	55	37	0110111
ETB (CTRL-W)	23	17	0010111	8	56	38	0111000
CAN (CTRL-X)	24	18	0011000	9	57	39	0111001
EM (CTRL-Y)	25	19	0011001	:	58	3A	0111010
SUB (CTRL-Z)	26	1A	0011010	;	59	3B	0111011
ESC (CTRL-[)	27	1B	0011011	<	60	3C	0111100
FS (CTRL-\)	28	1C	0011100	=	61	3D	0111101
GS (CTRL-])	29	1D	0011101	>	62	3E	0111110
RS (CTRL-^)	30	1E	0011110	?	63	3F	0111111
US (CTRL-_)	31	1F	0011111	@	64	40	1000000
(SPACE)	32	20	0100000	A	65	41	1000001

ASCII	Decimal	Hexadecimal	Binary
B	66	42	1000010
C	67	43	1000011
D	68	44	1000100
E	69	45	1000101
F	70	46	1000110
G	71	47	1000111
H	72	48	1001000
I	73	49	1001001
J	74	4A	1001010
K	75	4B	1001011
L	76	4C	1001100
M	77	4D	1001101
N	78	4E	1001110
O	79	4F	100111
P	80	50	1010000
Q	81	51	1010001
R	82	52	1010010
S	83	53	1010011
T	84	54	1010100
U	85	55	1010101
V	86	56	1010110
W	87	57	1010111
X	88	58	1011000
Y	89	59	1011001
Z	90	5A	1011010
[	91	5B	1011011
\	92	5C	1011100
]	93	5D	1011101
^	94	5E	1011110
_	95	5F	1011111
`	96	60	1100000
a	97	61	1100001
b	98	62	1100010
c	99	63	1100011
d	100	64	1100100
e	101	65	1100101
f	102	66	1100110
g	103	67	1100111
h	104	68	1101000
i	105	69	1101001
j	106	6A	1101010
k	107	6B	1101011
l	108	6C	1101100
m	109	6D	1101101
n	110	6E	1101110
o	111	6F	1101111
p	112	70	1110000
q	113	71	1110001
r	114	72	1110010
s	115	73	1110011
t	116	74	1110100
u	117	75	1110101
v	118	76	1110110
w	119	77	1110111
x	120	78	1111000
y	121	79	1111001
z	122	7A	1111010
{	123	7B	1111011
|	124	7C	1111100
}	125	7D	1111101
~	126	7E	1111110
DEL	127	7F	1111111

Table of Symbols

SYMBOL	*MEANING*
NUL	Null.
SOH	Start of heading.
STX	Start of text.
ETX	End of text.
EOT	End of transmission.
ENQ	Enquiry.
ACK	Acknowledge.
BEL	Bell.
BS	Backspace.
HT	Horizontal tabulation.
LF	Line feed.
VT	Vertical tabulation.
FF	Form feed.
CR	Carriage return.
SO	Shift out.
SI	Shift in.
DLE	Data link escape.
DC1	Device control 1.
DC2	Device control 2.
DC3	Device control 3.
DC4	Device control 4.
NAK	Negative acknowledge.
SYN	Synchronous idle.
ETB	End of transmission.
CAN	Cancel.
EM	End of medium.
SUB	Substitute.
ESC	Escape.
FS	File separator.
GS	Group separator.
RS	Record separator.
US	Unit separator.

APPENDIX E

Summary of DOS Editing Keys and EDLIN Commands

The Editing Keys

F1	Bring back one character.
F2x	Bring back everything up to but not including the character which replaces x.
F3	Bring back the remainder of the line. If the line is still intact in the buffer, the entire line is brought back.
F4x	Skip over everything up to but not including the character which replaces x.
F5	Put the current line in the buffer.
Del	Skip over one character.
Ins	Insert mode. Allow characters to be typed in while still leaving the line in the buffer as it is.
Esc	Cancel editing the current line.
BkSpc	Back up and erase one character on the screen and put it back in the buffer.

EDLIN Commands

nA	Append. Bring in n number of lines for editing from the file. Used when the entire file cannot fit into memory at one time.
x,y,zC	Copy. Copies a line or lines to a different place in the text file.
x,yD	Delete. Delete a range of lines.
n	Edit. Edit line number n.
E	End. End the edit session and save the file to disk.
nI	Insert. Insert a line at line number n.
x,yL	List. List a range of lines.
x,y,zM	Move. Move a range of lines to line number z.
x,yP	Page. List a range of lines. Make the last line shown be the current line.
Q	Quit. Leave the EDLIN program without saving the file on disk.
x,y(,?)R<string1> ^Z<string2>	Replace. Search through the range of lines to find string1 and replace it with string2. If the ? option is used, stop and ask if the string should indeed be replaced.
x,y(,?)S<string>	Search. Search through the range of lines for string1. When it is found, display that line. If the ? option is used, stop and ask whether the search should continue.
nT<filename>	Transfer. Merge a file from disk into the file currently being edited.
nW	Write. Record on disk the lines being edited, starting at line number n. Used to free up memory so more lines can be brought in, when a file will not fit into memory all at once.

APPENDIX F

About the Keyboard

The IBM PC keyboard can be very confusing at first. It is ironic that the makers of the Selectric typewriter, whose keyboard is recognized as the standard, are also the manufacturers of one that has symbols that are obscure, an almost microscopic ENTER key, and a left shift key that is woefully misplaced.

Working from left to right across the keyboard, let's look at the names and functions of the keys that require an explanation. The ten keys labeled F1 through F10 are called **function keys.** With applications such as Lotus 1-2-3 or WordStar, they will do specific tasks. When you are using DOS, they produce some editing functions (see Chapter 9 and Appendix E). The most important one used with DOS is F3. It retypes the last command used.

In the upper left corner is the Escape key, labeled Esc. At the DOS level it will cancel a command being typed in. When you are using applications, it will do whatever the programmers have set it up to do, which sometimes is nothing. Below it, the key with the two arrows on it is the TAB key. Below that is the Control key, labeled Ctrl. The Control key is another shift key. When it is held down and a second key is pushed once, a special code goes into the computer. These are called control codes. Many programs use them extensively. The Control key does nothing by itself. Just like the Shift key, it is held down while some other key is pressed. Throughout the book, a control character is notated this way: Ctrl-S. This means hold down the Control key and push the S key once. The fact that an uppercase S is shown here is not significant. It just means push the S key. Below the Control key is the infamous left shift key, labeled with an arrow pointing up. To its right is the backslash (\) key. When you want to shift, you'll get the backslash, and when you really do want the backslash, somehow you'll manage to miss it. It just takes a little getting used to. Below the shift is the Alternate key, labeled Alt. This is yet another shift-type key. Some programs make use of it; many do not.

Moving over to the right of the alphabetic characters, we find the Backspace key in the upper right. It has a left-pointing arrow on it. The common abbreviation for this key is BkSpc. When you are using DOS, it backs up and erases one character at a time. When you are using a word processing program, it may not do this. Instead, it may just backspace without erasing. This varies from program to program. Below the BkSpc key is the ENTER key. Each command that is typed in is followed by pushing the ENTER key once. This is the only way the computer knows that we have finished typing in our entry, and that it is then supposed to go on and process what we typed in.

Directly below the ENTER key is the Print Screen key. In the unshifted mode it produces an asterisk, but when a shift key is held down and it is pressed, it does a "screen dump." Try it sometime when the printer is turned on. To the left of this key is the right shift key, labeled with an arrow pointing up. Below that is the Caps Lock key. This is used to change between two different modes of operation. At startup, letter keys produce lowercase when not shifted. After pushing Caps Lock once (sorry, no indication anywhere that the mode of operation has been changed), the same keys produce uppercase letters. This is normally called an Alpha lock, because it locks only the letters A through Z, not any of the other keys. This is different from a shift lock on a typewriter, which locks all the keys. The only way to get an exclamation mark for example, no matter which way the Caps Lock is set, is to use the shift key.

Pushing the Caps Lock key again unlocks it. A key that goes back and forth between two modes, or some sort of ON-OFF situation, is called a toggle key. The Number Lock key, labeled NumLock, is another toggle key. At startup, the ten-key pad is not set up for entering numbers; rather, it responds with cursor movement commands, according to its HOME, END, PgUp, PgDn, and arrow labels. Exactly what these do varies from program to program. To get into the number entry mode, press the NumLock key once. Then the keys can be used to enter the numbers shown on them. To get back to cursor movement mode, press the NumLock key one more time.

At the bottom of the ten-key pad is the Delete key, labeled Del. When you are using DOS, this key does not back up and delete a character. In some word processing programs, it does. To the left of the Delete key is the Insert key, labeled Ins. In many programs this will put the user into an Insert mode – characters typed in do not wipe out characters that already exist. Finally, ending our tour of the keyboard, in the far upper right corner, we find the Scroll Lock, or Break key. In some programs it is a toggle key that changes the mode of operation of the ten-key pad. Mostly it is used to interrupt operations. To do this, it is used in conjunction with the Control key. This is normally referred to as a Control-Break sequence, abbreviated Ctrl-Break.

Index

Non-alphabetic characters

% symbol; see parameters (replaceable)
%PIPEx.$$$ filenames, 172
. symbol, 63, 67
.. symbol, 63-64, 67
.BAK files, 89
.BAS files; *see* BASIC
.BAT files; *see* batch files
.COM programs, 71-72, 76, 81, 88, 126
.EXE programs, 71-72, 76, 88, 126
.OVR files; *see* overlay files
> symbol; *see* redirecting output
>> symbol; *see* redirecting output
< symbol; *see* redirecting input
\ symbol, 55
  see also root directory
\ key, 192
| symbol, 171-172
  see also pipes

A

Alt key; see Alternate key
Alternate key, 192
  see also Ctrl-Alt-Del
American Standard Code for Information Interchange; *see* ASCII code
ASCII code, 24
  and batch files, 90-91, 100
  chart, 189
  definition, 47-48
  sorting by, 168
ASSIGN program, 78-79, 182, 188
  general form, 79
AST Research, 93, 179
AUTOEXEC.BAT files, 92-94
  and the PROMPT command, 122-123
  definition, 92
AUX: device name, 153

B

backing up disks, 130-145
  COPY versus DISKCOPY, 131-132
  using batch files, 140-145
  recommended procedure for hard disks, 139-140
Backspace key, 77, 102, 191-192
BACKUP program, 132-136, 182, 188
  general form, 133
  options, 134-136
BASIC, 81-85
  .BAS extension, 83
  definition, 81
  prompt, 83
  redirecting input, 164
  redirecting output, 162-163
  return to DOS, 83
  sample program, 82
  save a program, 83
BASIC.COM, 82-83
  see also BASIC
BASICA.COM, 82-83
  see also BASIC
batch file, 87-99, 112-129
  .BAT, 90, 126
  and replaceable parameters, 94-99
  AUTOEXEC.BAT file; *see* AUTOEXEC.BAT
  chaining, 124, 144
  definition, 90
  how to create one, 90-92
  how to execute one, 88
  interrupting execution, 89
  subcommands, 112-120
  what they do, 87-90
baud rate, 155-156
binary code, 48
  see also ASCII code
BIOS, 4
bit, 156
BkSpc key, 108, 191-192
  see also Backspace key
bootable disk, 11-12, 44
  see also formatting
booting, 5, 92
  see also AUOEXEC.BAT
bootstrap loader, 5, 92
BREAK command, 187, 188
Break key, 193
  see also Ctrl-Break
buffer
  in a printer, 173
  in RAM, 178-180
    see also spooler

C

Caps Lock key, 192-193
Carriage Return, 48, 103, 122
CAS1: device name, 153
Cassette BASIC, 82
  see also BASIC
CD command, 60-62, 182, 188
  general form, 60
  show current directory, 61-62
chaining batch files; *see* batch files
CHDIR command; *see* CD command
CHKDSK program, 43-46, 182, 188
  checking the File Allocation Table, 45
  the /F option, 46
clock board, 92
CLS command, 50, 94, 182, 188
COMMAND.COM, 12, 71
command files, 72
command processor, 71
COMP program, 1, 38-39, 182, 188
compiler, 81
COM1: device name, 153, 157, 174
CON: device name, 91, 153, 160-161
  used in creating batch files, 91
CONFIG.SYS file, 187, 188
CONSOLE; *see* CON: device name
contiguous sectors; *see* sector
control characters, 24, 192
  definition, 24
  in text files, 173
Control key, 24, 192
COPY command, 14, 32-39, 64-67, 91, 117, 183, 188
  COPY versus DISKCOPY, 131-132
  general form, 32, 64
  used to create batch files, 91
  used to create text files, 167
copying files, 32-39
  copy all files, 35, 66
  see also star dot star
  copy and change name, 36
  copy from current disk, 33
  copy selected files, 34-35
  see also wildcard
  copy to and from subdirectories, 64-67
  copy to current disk, 33
  duplicating a file, 36-37
  using COPY versus DISKCOPY, 131-132
  verifying with /V, 37-38
copying disks, 35, 40
  see also DISKCOPY
Ctrl key; see Control key
Ctrl-Alt-Del sequence, 5
Ctrl-Break, 50
  to stop a TYPE listing, 50
  to stop batch file execution, 89, 119
  to stop printer retries, 155, 157
Ctrl-C, 89, 119
Ctrl-NumLock, 21, 49, 79, 83, 105
Ctrl-P, 23, 30, 49, 81, 162
Ctrl-PrtSc, 23, 30
Ctrl-S, 105
Ctrl-Z, 48, 91-92, 103, 110, 164, 167
  see also F6 key
CTTY command, 160, 183, 188

D

databits, 155-156
date function, 6, 92
DATE command, 10, 50, 92, 183, 188
  in AUTOEXEC.BAT files, 92
dBASE, 8, 95-96, 116-117, 128, 141-142, 145, 147, 179
DEBUG program 187, 188
DEL command, 46-47, 183, 188
  general form, 46
  delete certain files, 46
  delete all files, 47
  delete files in a subdirectory, 67
  delete all files in a subdirectory, 67-68, 140
  previewing files to be deleted, 68, 146
Del key, 107, 191, 193
destination, 32, 36, 39, 69
  see also copying
device names, 151-153
DIR command, 18-22, 62-64, 183, 188
  including paths, 62-64
  options, 21
  to the printer, 22
directory listing, 18-23
  example, 19
  hidden files, 12, 44
  indication of current directory, 62
  of another drive, 21
  partial, 29
  used to preview deletion, 68, 146
disk, 7
  backing up disks, 130-145
  floppy disk capacity, 10, 12-13
  formatting, 9-15
  naming the disk volume, 14

switching disks, 8
DISKCOMP program, 1, 42-43, 183, 188
DISKCOPY program, 1, 40-42, 183, 188
general form, 41
not with hard disks, 132
using DISKCOPY versus COPY, 131-132
with the /1 option, 42
disk drive, 7
current disk drive, 7-8, 33-34
default disk drive, 8
disk drive specification, 22-25
see also filename
double sided, 13
read/write head, 130
single sided, 13
switching between disk drives, 8
Disk operating system; see DOS
DOS, 1
and the PCjr, 1
Editor; *see* EDLIN
hidden files, 12, 44
menus, 125-129
what it is, 1, 3-5
where it is, 5-6
versions, 1-2, 13
drive specification, 25-26, 56
see also filename
duplicating files; *see* copying files
duplicating disks; *see* copying disks

E

ECHO command, 112-113, 182, 188
editing keys, 106-108, 191
EDLIN functions
Copy, 109
Delete, 108-109
Insert, 101
List, 105
Move, 109
Referring to lines, 103
Replace, 109-110
Search, 111
EDLIN program, 100-111, 183, 188
editing keys, 106-108, 191
the prompt, 101
electronic disk, 178-180
electronic spooler, 178-180
END key, 193
ENTER key, 192
ERASE command; *see* DEL command
Esc key, 108, 191-192
ETX/ACK protocol, 173
EXE2BIN program, 187, 188
extension, 25
see also filename
external command, 19
definition, 70
list of external commands, 188

F

F1 key, 108, 191
F2 key, 107-108,191
F3 key, 77, 107-108, 191-192
F4 key, 108, 191
F5 key, 191
F6 key, 91, 167
see also Ctrl-Z
FAT; *see* File Allocation Table
FDISK program, 1, 187
File Allocation Table (FAT), 44-45
filename, 24-26, 31, 34, 57-58
characters not allowed, 24
complete explanation, 57-58
corename, 25
definition, 24-25
disk drive specification, 25-26
example, 26
extension, 25
filename versus program name, 73
general form, 31, 57
maximum number of characters, 58
no distinction between upper and lower case, 10, 128
path in a filename, 57
see also path
filters, 166-171
FIND program, 168-170, 184, 188
MORE program, 171, 185, 188
SORT program, 166-168, 186, 188
FIND program, 168-170, 184, 188
general form, 169
options, 170
floppy disk; *see* disk
FOR command, 117, 182, 188
FORMAT program, 9-15, 19, 184, 188
general form, 10
formatting, 9-15, 131
definition, 9-10
including volume name, 14
options, 12-15
with DOS, 11-12
see also bootable disk
without DOS, 10-11
Freeware, 174

G

GOTO command, 114, 120, 182, 188
GRAPHICS program, 1, 176-177, 184, 186

H

handshaking, 173, 177
hard disk
  backing up, 132-137
  BACKUP program, 132-137
  booting from, 6
  capacity, 14
  formatting, 14, 16
  how to organize, 146-148
  recommended backup procedure, 139-140
  sectors, 9-10
hexadecimal code, 48
hidden files, 12, 44
  see also DOS
hierarchical tree structure, 52
HOME key, 193

I

IBM printer, 176
IBM XT, 5, 6, 11, 78, 82, 84-85, 93, 133, 143, 150
IBM PCjr, 1, 5
IF command, 115-117, 119-120, 182, 188
initializing
  alternate word for formatting, 9
  initializing ports, 155, 158
Ins key, 107, 191, 193
interfacing printers, 173
internal command, 19
  definition, 70
  list of internal commands, 188
interpreter, 81
label (batch file), 114

L

Line Feed, 48, 103, 122
LINK program, 187, 188
Lotus 1-2-3, 98-99, 116, 128, 145, 147, 192
  and .WKS extensions, 133, 145
LPT1: device name, 153, 157
LPT2: device name, 153
LPT3: device name, 153

M

MD command, 58, 184, 188
  general form, 58
Microsoft, 1, 153, 177
  Microsoft BASIC, 81
MKDIR command; *see* MD command
MODE program, 1, 87, 93, 153-160, 184, 188
  Version 1 (setup parallel port), 154-155
  Version 2 (setup serial port), 155-157, 174
  Version 3 (redirect parallel to serial), 157-158, 162
  Version 4 (switch video modes), 158-160
modem, 4, 151, 156
monochrome, 158
MORE program, 171, 185, 188
MS-DOS, 1, 5, 177
multifunction board, 92-93, 179
  SixPakPlus, 179
MultiMate, 91

N

nesting in batch files, 125
NULL: device name, 153
Number Lock key, 193
NumLock key; see Number Lock key

O

object code, 81
Operating system, 1, 3-5
  see also DOS
overlay files; *see* programs

P

page eject, 175
parallel, 151
  parallel communications, 151-152
  parallel port, 151
parameters
  as add-ons to commands, 21
  replaceable parameters in batch files, 94-99, 116, 124, 128, 172
parent directory, 59, 63
  definition, 59
  referring to it in commands, 63
  see also .. symbol

parity, 155-156
PATH command, 73-76, 84, 185, 188
  general form, 75
  multiple search paths, 75
path, 53, 55-56
  definition, 53
  how to notate, 55-56
  in a filename, 57
  in the CD command, 60
  in the MD command, 58-59
  with BASIC, 84
PAUSE command, 113-114, 182, 188
PC TALK III, 174
PC-DOS; *see* DOS
peripheral device; *see* device
PgDn key, 193
PgUp key, 193
pipes, 171-172
port, 23, 151
PRINT program, 173-176, 185, 188
  options, 175-176
  used to print batch files, 174
print spooler, 178-180
PRN device name, 139, 153, 174
programming language, 81
programs, 4
  .COM programs, 72
  .EXE programs, 72
  definition, 4
  filename versus program name, 73
  in BASIC, 81-85
  overlay files, 76-78, 89
  running programs, 10, 70-86
prompt, 7
  EDLIN prompt, 101
  system prompt, 7
PROMPT command, 120-124, 185, 188
  in AUTOEXEC.BAT files, 122-123
protecting data, 15
  see also backup
protocol, 173
pseudodisk, 178

Q

queue, 174-175

R

RAM, 5, 70
RAM disk; see electronic disk
RD command, 67-68, 186, 188
  general form, 67
RECOVER program, 187, 188
redirecting input, 163-164
redirecting output, 161-163
REM command, 113, 182, 188
remote terminal, 160
REN command; *see* RENAME command
RENAME command, 47, 185, 188
replaceable parameters; *see* parameters
RESTORE program, 133, 136-137, 185, 188
  the /P option, 137
RMDIR; *see* RD
ROM, 5, 82
root directory, 52, 55
  creating, 53, 58
  definition, 52
  its name (\), 55
  move into it, 61

S

screen dump, 23
  for graphics, 176
  for text, 23
scrolling, 21
  stop scrolling, 21, 49, 79
sector, 9-10, 13, 44
  contiguous, 45
    see also CHKDSK
  in blocks, 44-45
  in clusters, 44
serial, 4, 151
  serial communications, 151-152
serial port, 4, 93, 151
  setup, 155-157
    see also MODE program
SET command, 187, 188
SHIFT command, 118-120, 182, 188
shift keys, 192
Shift-PrtSc, 23, 176-177, 192
SixPakPlus multifunction board, 179
SORT program, 166-168, 186, 188
source code, 81, 100
standard input device, 151, 163
standard output device, 152, 161
star dot star, 28, 35-36, 47, 66
  see also wildcard
stopbits, 155-157
subdirectory, 20, 52-69
  creating, 58-60
    see also MD command
  definition, 52
  in a directory listing, 20
  naming, 59
syntax, 32, 89

SYS program, 14, 186, 188
system disks, 12
system prompt, 7

T

TAB key, 192
ten-key pad, 193
time function, 7, 92
TIME command, 10, 50, 92, 186, 188
  in AUTOEXEC.BAT files, 92
toggle, 23, 193
TREE program, 1, 79-81, 186, 188
  general form, 79
  the /F option, 81
TYPE command, 47-50, 174, 186, 188
  general form, 49
  used in batch files, 94, 126, 142

V

VER command, 187, 188
VERIFY command, 38, 140, 186, 188
VOL command, 187, 188
Volkswriter, 90

W

wildcard, 26-31
  anything, 28
  comparing files, 38-39
  copying all files, 35-36
  copying selected files, 34-35
  erasing files, 46-47
  erasing files in a subdirectory, 67-68
  in the PRINT command, 176
  partial directories, 29
  single character, 26-27
WordStar, 8, 76-78, 87-90, 91, 98, 147, 192
  and batch files, 91, 115, 125, 126, 128, 141
  .BAK files, 89, 115
write-protect notch, 15
write-protect tab, 15

X

XON/XOFF protocol, 173

Notice

Thank you for purchasing MS-DOS and PC-DOS on the IBM PC. We hope it has been a valuable learning tool.

Because of the constant and rapid changes in the field of personal computers, book publishers are forced to maintain tight, and often rushed schedules in order to stay in step with the industry.

We at Microtrend Books wish to keep you informed of any updates, changes, or errors in our books. We also welcome your comments, suggestions and corrections. Please send us your name and mailing address on a postcard, and we will send you an Errata sheet free of charge. Be sure to mention the title of this book.

Mail to:

MICROTREND BOOKS
3719 Sixth Avenue
San Diego, CA 92103

COMPUTERS

The fantastic new developments in IBM PC software and languages can make computing easier and more powerful than ever before. But it's your disk operating system (DOS)—the buffer between you and the microprocessor—that makes it all possible. Now, even if you are totally unfamiliar with DOS, this complete, painless guide to understanding and using it lets you master the commands that will take care of disk "housekeeping" tasks, so you can move on to bigger and better computing.

Microtrend's **MS-DOS & PC-DOS on the IBM PC** tells you exactly what operating systems are, how they work, what they can do, and how to use them. Complete with a section on special considerations for hard-disk users, a summary of DOS commands, and a full chapter on the PCjr's major features, the book covers:

- booting up
- preparing disks and formatting
- accessing directories
- copying files
- fundamental commands
- running programs
- batch files
- DOS menus

. . . and everything you need to know to harness the power of your IBM PC.

PRENTICE-HALL, Inc.
Englewood Cliffs, New Jersey 07632

3

ISBN 0-13-604281-3

0 21898 60428 6